Z8000 HANDBOOK

Z8000 HANDBOOK

MARTIN L. MOORE

Prentice-Hall, Inc.
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Moore, Martin L.
Z8000 handbook.

"A Spectrum Book."
1. Zilog Model Z8000 (Computer) I. Title.
QA76.8.Z55M66 001.64 82.7485
ISBN 0-13-983874-0 AACR2
ISBN 0-13-983866-X (pbk.)

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Editorial/production supervision by Frank Moorman
Cover design by Jeannette Jacobs
Manufacturing buyer: Barbara A. Frick

ISBN 0-13-983874-0

ISBN 0-13-983866-X {PBK.}

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

CONTENTS

CHAPTER 5

PREFACE

ABOUT THIS BOOK

The Z8000 Handbook describes the operation of the Z8001 and Z8002 16-bit microprocessors. In addition, the operation of various support devices, and implementation of the Z8001 and Z8002 in a minimum microprocessor system are discussed.

AIM OF THIS BOOK

The Z8000 Handbook is designed to support a variety of readers. If you're a hardware engineer, you'll find timing specifications and interface descriptions. If you're a software engineer, you'll find a complete description of the Z8000 instruction set.

This book is not written in a tutorial manner, and expects that you know the basics of microprocessors, numbering systems, binary logic, and assemblers.

No book can be all things to all people, but an honest effort has been made to make the Z8000 Handbook as thorough and complete as possible.

Organization

The Z8000 Handbook is separated into five chapters, an index, and an appendix:

Chapter 1 — The first chapter is an introduction to the Z8000 family of 16-bit microprocessors. An overview of the Z8001 and Z8002 are given here.

Chapter 2 — The second chapter deals with the Z8000 instruction set, and includes information on registers, data types, addressing modes, and memory management.

Chapter 3 — This chapter deals exclusively with interfacing the Z8000 family to the outside world. Device lines are described, as are timing requirements, interrupt structure, and memory implementation.

Chapter 4 — In this chapter, we'll discuss designing and building a minimum Z8000 system, from a block diagram viewpoint.

Chapter 5 — The fifth chapter deals with interfacing the Z8000 family to Z80-type support devices, as well as the new Z8000-type support devices.

Index — The index contains a cross-reference of all major subjects found in the book.

Appendix — The appendix contains a summary of Z8000 instructions.

Conventions/Presentation

The conventions used in the Z8000 Handbook are few. Hexadecimal notation is used throughout, except where otherwise noted. Negative-true logic is indicated by a signal name with an overbar (NAME).

In this book, a bit is a single binary on or off, a nybble is a 4-bit unit, a byte is an 8-bit unit, a word is a 16-bit unit, and a long word is a 32-bit unit.

Z8000 HANDBOOK

1

INTRODUCTION

THE Z8000 FAMILY

The Z8000 family of 16-bit microprocessors was introduced by the Zilog Corporation, a subsidiary of Exxon. The Zilog Z8000 family consists of the Z8001 and the Z8002. Both are 16-bit microprocessors. The Z8001 has an 8 million word address range, while the Z8002 is limited to 64 K-words of memory. Both devices are second-sourced by Advanced Micro Devices as the AmZ8001 and AmZ8002.

In this chapter, we explore both the Z8001 and Z8002. We deal with device architecture, register usage, address and data functions, control and interrupt capabilities, and an overview of the instruction set. All these subjects are covered briefly. Later chapters provide detailed information about each area.

General Information

Operationally, the Z8001 and Z8002 closely resemble each other. The greatest difference is in the address range of the devices. The Z8001 can directly address 8 M-words of memory, while the Z8002 is limited to 64 K-words of memory. This difference in memory range shows up in the Z8001's expanded register set. And the instruction set differs between the

devices in that the Z8001 instruction set includes instructions specific to the extended memory range and provides memory management capability in its instruction set.

Software written for the Z8002 is directly transportable to the Z8001, but the reverse is not true.

In the next paragraphs, we discuss the Z8001 and Z8002 in terms of the "Z8000" device. The Z8001 or Z8002 is discussed only when a feature specific to either device is described.

ARCHITECTURE

Registers

The Z8000 is a register-oriented device. The Z8000 contains sixteen 16-bit registers (R0 through R15) which, with one exception, are never implied in an instruction. This frees all registers for general purpose use. Register flexibility is one of the device's strengths, in that the Z8000 does not contain an "accumulator" register. Any of the sixteen general purpose registers can serve as an accumulator. This also allows a great deal of programming flexibility. Figure 1-1 illustrates the Z8001 register set, and Figure 1-2 illustrates the Z8002 register set.

General Purpose Registers. The Z8000 register structure allows manipulation of 8-bit, 16-bit, 32-bit, or even 64-bit data words. Registers R0 through R7 can be divided in two, creating sixteen 8-bit registers. Registers R0 through R13 can be doubled up to create 32-bit registers. Finally, R0 through R11 can be used as two 64-bit registers during multiply and divide operations.

Stack Pointer. You'll notice in Figures 1-1 and 1-2 that there is a difference between the Z8001 and Z8002 as far as the stack pointer registers are concerned. The Z8001 has, in effect, two 32-bit stack pointers. The Z8002 uses two 16-bit stack pointers.

Why two? This brings up another feature of the Z8000 microprocessor. You can designate memory space as being either "system" memory or "normal" memory. "Normal" memory is the memory you would normally use to contain the current program or programs under development. "System" memory, however, can be set aside for system operations. For example, in a multi-user environment the operating system would be set off into "system" memory, while memory designated for users would be referred to as "normal" memory space. This division of memory resources is supported in the Z8000 instruction set. In the system mode, all instructions are supported, while in the normal mode, certain system instructions are prohibited.

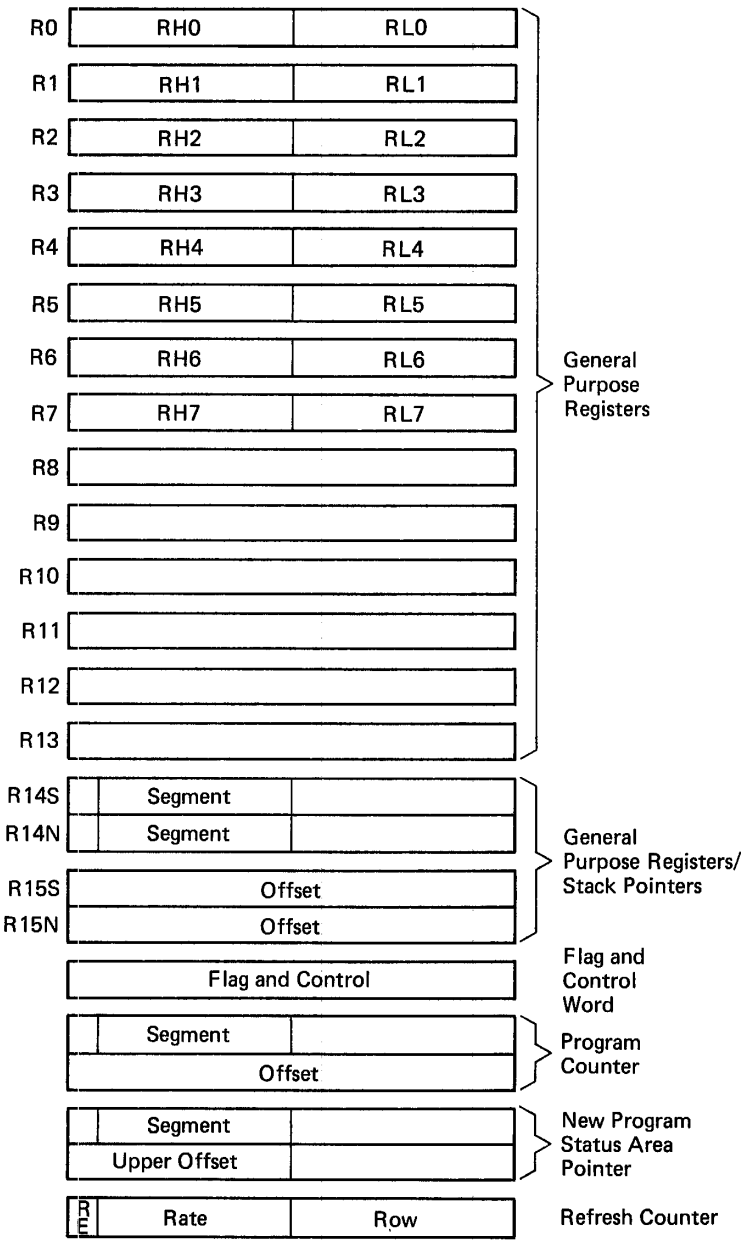


Figure 1-1 Z8001 register set.

System and normal instructions are discussed later in this chapter, and in detail in Chapter 2.

Program Status. Refer again to Figures 1-1 and 1-2. You'll notice that the Z8001 has a greater number of program status registers than does the Z8002. The Z8001, instead of using the usual program counter register, uses two program counter

registers, "PC Segment No." and "PC Offset". This program counter structure allows the Z8001 its 8 M-words of memory space. The PC Offset register points to one of 64 K-words within what is termed a memory segment, while the PC Segment No. register points to one of 128 different memory segments.

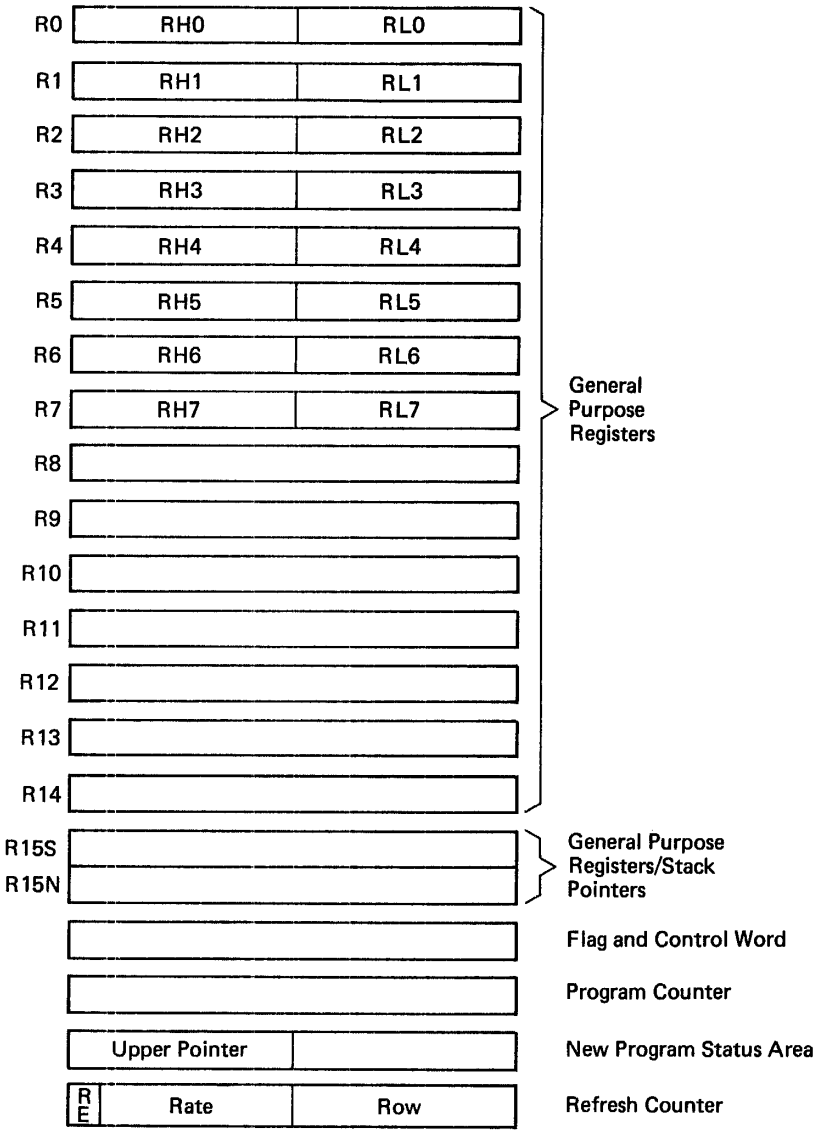


Figure 1-2 Z8002 register set.

New Program Status Area Pointer. The next point of interest is the New Program Status Area pointer. When an interrupt or instruction trap occurs, the Z8000 pushes old status information onto the stack and fetches new status information from the location pointed to by the New Program Status Area

pointer. For the Z8001, new program status information includes the flag and control word and a two-word segmented program counter value. The Z8002 new program status information includes the flag and control word and the value of the 16-bit program counter register. In addition to status information, a data word is pushed onto the stack to tell you what caused the interrupt or instruction trap. More about interrupts and traps later in this Chapter.

Refresh Counter. Finally, we come to the refresh counter register. The Z8000 contains a register that automatically refreshes dynamic RAM. The refresh rate is programmable (from 1 to 64 usec with a 4 MHz clock). The refresh row counter is nine bits wide, and allows 256 rows of RAM to be refreshed.

This completes the overview of the Z8000 register set. More information about the Z8000 registers is given in the first part of Chapter 2.

ADDRESS AND DATA

Remember that the Z8002 is limited to 64 thousand memory locations. The Z8001, however, can access over 8 million memory locations. Look at Figure 1-3 to see how this is accomplished. Both the Z8001 and Z8002 have sixteen pins, labeled AD0 through AD15. These pins take care of the Z8002's memory location and data word requirements.

The Z8001 has seven pins called SN0 through SN6. These seven pins are used by the Z8001 to point to 128 different segments. The 48-pin Z8001 uses, in effect, a 23-bit address.

This capability can be increased even further with the use of a memory management device. Notice in Figure 1-3 that there are four pins called ST0 through ST3. The memory management device (called a Z8010 MMU by Zilog) connects to ST0 - ST3, AD8 - AD15, and SN0 - SN6. The memory management unit then provides transparent segment relocation and memory protection. The Z8001's address space thus increases to 48 M-words of memory. More information on the Z8010 can be found in Chapters 4 and 5. The memory capabilites of the Z8001 will be discussed in more detail in Chapter 3.

Addressing Modes

Eight addressing modes are used in the Z8000:

Register — In the register mode, the instruction's operand is located in a register. The register type (byte, word, register pair or register quad) is always

implied in the instruction.

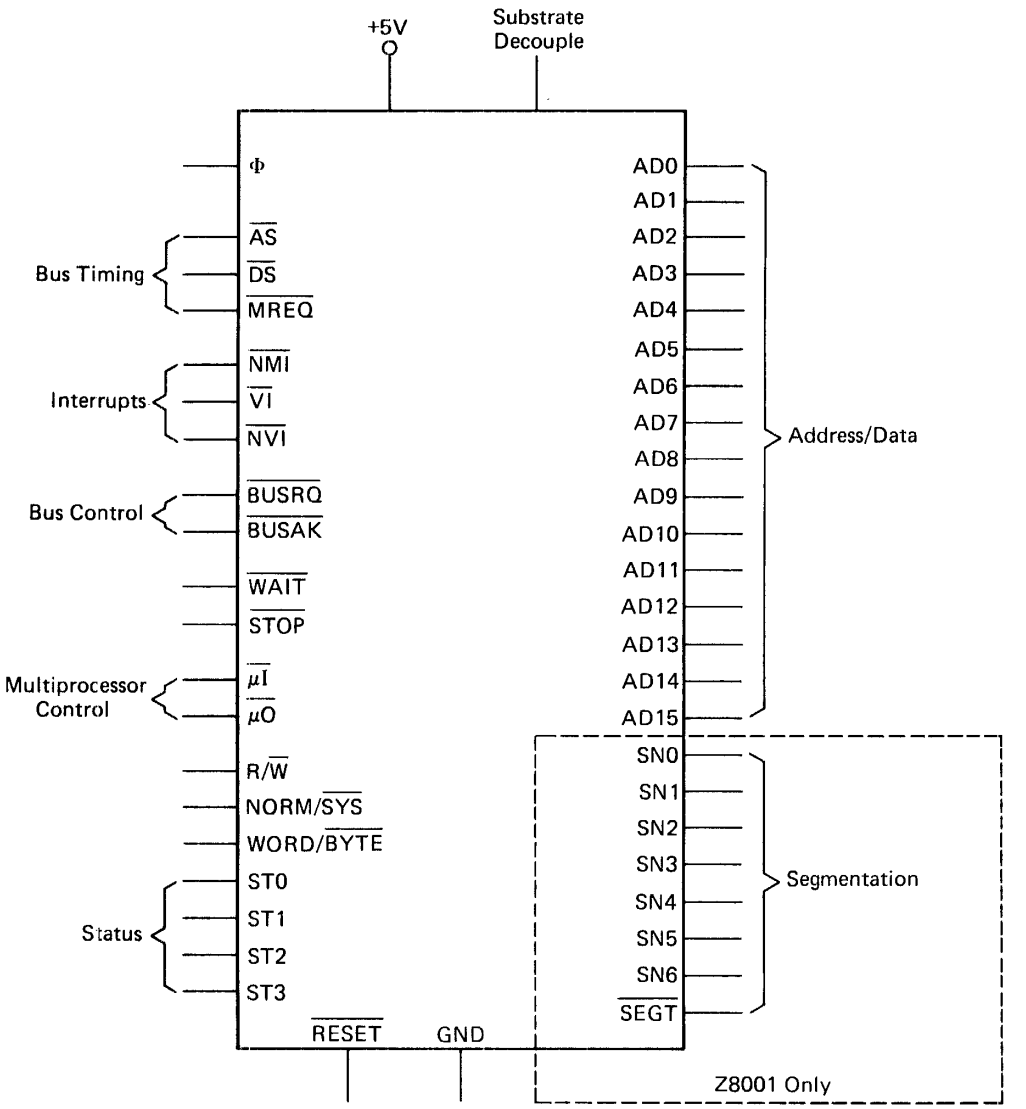


Figure 1-3 Z8000 pinout.

Indirect — When an instruction is used in the Indirect Register mode, the instruction's operand is pointed to by a register. Any 16-bit register or register pair (Z8001 only) can be used to point to the operand location.

Direct — The Direct Address mode requires that the address to be accessed be included within the instruction.

Indexed — The Indexed addressing mode uses a register as an offset to an address given immediately following the instruction. The offset value contained in the register is added to the address in the address field of the instruction, thus pointing to an indexed location.

Immediate — The Immediate addressing mode requires that the operand for an instruction be contained within the instruction itself. In other words, the operand is implied by the instruction.

Base — The Base Address mode can be considered to be the logical equivalent of the Indexed address mode. In the Base Address mode, the instruction contains the offset value, while a register contains the base address. The two are added to determine the operand's location.

Indexed — The Base Indexed mode is an extension of the Base Address mode. In this case, the offset value is contained in one register, while the base address is contained in another. The instruction itself does not contain either the offset or base address.

Relative — In the Relative Address mode, the instruction contains a 2's complement offset value. The 2's complement value is added to the value of the Program Counter register to form the operand's address.

Data

The Z8000 data lines are multiplexed with the address lines. The Z8000 can handle bits, BCD digits (four-bit nybbles),

bytes, 16-bit words, 32-bit words, and byte and word strings. Individual data bits can be set, reset, or tested. 4-bit nybbles are used in BCD arithmetic operations. Bytes are used in character or small data manipulation. The 16-bit words are typically used as integer values, instructions, and non-segmented addresses. The long words are used for large integer values and segmented addresses. Data words 64-bits long are used in extended multiply and divide instruction.

INTERRUPTS AND TRAPS

The Z8000 handles three types of interrupts: non-maskable (NMI), non-vectored (NVI), and vectored (VI). Five software traps are available: system call, illegal instruction, privileged I/O, privileged instruction, and segmentation. Both traps and interrupts are handled in a similar manner.

Of the three interrupt types, NMI has the highest priority, VI has the second highest priority, and NVI has the lowest priority. Whenever an interrupt occurs, the Z8000 generates an interrupt acknowledge on status lines ST0 through ST3. In addition, both the VI and NVI interrupts must be enabled by setting their respective bits in the Flag and Control Word register.

When an interrupt or trap occurs, the Z8000 pushes its current status information onto the system stack, along with one or more data words that indicate the reason for the trap or interrupt. New program status is fetched from the location pointed to by the new program status register. Then the interrupt or trap is processed. Of course, reset is handled in a slightly different manner. When a reset occurs, a four-word (Z8001) or two-word (Z8002) program status is fetched from the bottom of memory, and no old information is saved on the system stack.

CONTROL AND STATUS

Control

The Z8000 is controlled by writing to the Flag and Control Word register (Figure 1-4).

There is a significant division between the upper eight bits and lower eight bits of the Flag and Control Word. Notice that the lower eight bits are what might be termed "normal" program status information. Carry (C), Zero (Z), Sign (S), Parity/Overflow (P/V), Decimal Adjust (DA), and Half Carry (H) flags are manipulated during normal program

operations.

The upper eight bits, Segment (SEG), Extended Processing Enable (EPE), System/Normal (S/N), Vectored Interrupt Enable (VIE), and Non-Vectored Interrupt Enable (NVIE) control the operation of the Z8000 itself. The only way these bits can be changed is through the use of privileged instructions.

For example, if the Z8000 is operating in the "system" mode the Flag and Control Word register can be altered at will. If, however, the Z8000 is operating in the "normal" mode, any instruction that will affect the upper eight bits of the Flag and Control Word register generates a software trap, and processing is interrupted. This action protects the integrity of the system.

In fact, this system protection feature goes a bit further than just preventing access to the Flag and Control Word. The Z8000 won't allow you to perform certain I/O operations or use instructions that affect inter-processor communications unless you're in the "system" mode.

Status

The current status of the Z8000 can be monitored with the use of four output lines called ST0 through ST3. These lines report the status of the processor, as shown in Table 1-1.

Table 1-1
Z8000 Status Lines

ST3	ST2	ST1	ST0	Operation
L	L	L	L	Internal Operation
L	L	L	H	Memory Refresh
L	L	H	L	Normal I/O Operation
L	L	H	H	Special I/O Operation
L	H	L	L	Segment Trap Acknowledge (Z8001) (Reserved in Z8002)
L	H	L	H	NMI Acknowledge
L	H	H	L	NVI Acknowledge
L	H	H	H	VI Acknowledge
H	L	L	L	Memory Transaction for Operand
H	L	L	H	Memory Transaction for Stack
H	L	H	L	Reserved
H	L	H	H	Reserved
H	H	L	L	Instruction Fetch from Memory (subsequent word)
H	H	L	H	Instruction Fetch from Memory (first word)
H	H	H	L	Reserved
H	H	H	H	Reserved

I/O

The Z8000 uses device lines AD0 through AD15 to address I/O ports. Instead of using a single memory-I/O line indicating a memory or I/O access, the Z8000 uses the status lines listed in Table 1-1. Notice in Table 1-1 that two kinds of I/O operations can be specified: Normal I/O and Special I/O. Both Normal and Special I/O operations use the lower eight bits (AD0--AD7) of the address/data lines to transfer data to the selected port. The difference between Normal and Special I/O can only be seen on the Status lines.

The Z8000 can output or input data in word or byte format. In word format, two bytes are written or read one after another. In addition, the Z8000 instruction set supports autoincrement and autodecrement. That is, the I/O instruction can specify a memory range from which data is to be written to a port or to which input data is to be stored. Specify the memory range, and the Z8000 will input or output that many items.

INSTRUCTION SET

The following offers some idea of the strength of the Z8000 instruction set. There are 110 distinct instruction types which, when combined with data and addressing modes, add up to 414 instructions. Most of the instructions use any of the five main addressing modes and operate on byte, word, and long word data types.

The Z8000 supports nine groups of commands:

- load and exchange
- arithmetic
- logical
- program control
- bit manipulation
- rotate and shift
- block transfer and string manipulation
- input/output
- CPU control

Load and Exchange

This instruction group includes instructions like CLR (clear destination address), EX (exchange register data with addressed data), LD (load immediate data into addressed location), PUSH (push data onto the stack), and POP (pop data

off the stack).

Most of the load and exchange instruction group operate on bytes, words, or long words. In all cases, any general purpose register except R0 can be used as a stack pointer. Source and destination information is determined by the specific instruction, but in most cases depends on the addressing mode used. There are twenty-one instructions in this group.

Arithmetic

This instruction group includes instructions to add, subtract, compare, decrement, and increment. The Z8000 also supports hardware multiply and divide instructions. That is, no subroutine is required to perform multiply and divide processing. The DIV command gives a 16-bit quotient and 16-bit remainder, while the DIVL command gives a 32-bit quotient and 32-bit remainder. The multiply command works with either a 32-bit multiplicand (MULT) or a 64-bit multiplicand (MULTL). Both the multiplicand and multiplier are treated as signed, 2's complement integers. Altogether, there are twenty-nine arithmetic instructions.

Logical

The logical operators include AND, COM (complement), OR, and XOR. In addition, the Z8000 can test bytes, words, long words, and condition codes with the TEST and TCC instructions. There are thirteen instructions in this group.

Program Control

The program control group includes calls and jumps. One of the call instructions is SC (system call). This instruction produces a system call trap and allows you to include an identifier in the instruction that will tell the system program what caused the trap. There are nine program control instructions.

Bit Manipulation

The BIT instruction is used to test a specific bit within a byte or word. The RES instruction allows a specific bit within a byte or word to be reset to zero. The SET instruction sets a specific bit within a byte or word to one. The TSET instruction loads the sign bit of a byte or word into the S flag and sets all remaining bits to one. There are fourteen instructions within this group.

Rotate and Shift

The rotate and shift group is used to rotate and shift data logically or arithmetically left or right. There are twenty-eight rotate and shift instructions.

Block Transfer and String Manipulation

There are thirty-two block transfer and string manipulation instructions, ranging from compare and decrement (CPD) through load, decrement, and repeat (LDDR), to translation instructions. The translation instructions are used to translate a byte or byte string by using the source byte as a table pointer. The translation instructions are flexible enough to offer translate and autodecrement; translate, autodecrement and repeat; translate, autoincrement, repeat, and test; and more.

Input/Output

The Z8000's I/O instructions are as varied as its block transfer and string manipulation instructions. The I/O instructions are subdivided into two groups: normal I/O and special I/O. There are thirty I/O instructions.

CPU Control

The sixteen CPU control instructions control the operation of the Z8000 itself. Instructions included in this group are flag control, halt, and control register manipulation.

Summary

This has been a brief description of the Z8000 instruction set. The next chapter in this book describes each instruction in full detail.

MULTI-PROCESSOR SUPPORT

Both the Z8001 and Z8002 have two pins (μO and μI) dedicated to multi-processor systems. The μO pin disables the Z8000 while another processor is using the bus. The μI pin lets the Z8000 prevent another processor from taking the bus while performing critical functions.

SUPPORT DEVICES

Zilog has designed the Z8000 to be fully compatible with almost the full line of Z80 support devices. In most cases, the Z80 devices can be interfaced by simply using them in pairs. More about interfacing the Z8000 with Z80 support devices in Chapters 4 and 5.

The new Z8000 support devices include the Z8010 Memory Management Unit, the Z8034 Universal Peripheral Controller, the Z8036 Counter/Timer and Parallel I/O Unit, and the Z8030 Serial Communications Controller.

The massive memory accessing capabilities of the Z8001 almost demand that a prospective designer use the Z8010 Memory Management Unit (MMU). The Z8010 MMU offers dynamic memory segment relocation, allowing multiple users to access memory without exceeding memory bounds. This means that users need not specify where in "physical" memory their program lies, but instead can rely on the MMU to keep track of its location. The Z8010 can also prevent protected areas of memory from being manipulated by unauthorized users. Specific segments of memory can be assigned protection attributes so that, if an unauthorized access attempt is made, a software trap is generated.

The Z8034 Universal Peripheral Controller (UPC) is actually a microprocessor that works as a "slave" to the Z8000. The Z8034 UPC is very similar to Zilog's Z8 eight-bit microprocessor and uses the Z8 instruction set. The Z8034 has three programmable I/O ports. Port 1 and Port 2 are 8-bit bidirectional I/O ports. Port 3 has two input and two output lines. The Z8034 UPC has 2 K-bytes of ROM onboard and can be instructed to execute from the internal ROM, or from instructions fed to it by the Z8000. This flexibility allows the Z8034 to perform data manipulation tasks such as reformatting or arithmetic operations.

The Z8036 Counter/Timer and Parallel I/O Unit (CIO) externally resembles the Z8034. The Z8036 has two 8-bit bidirectional I/O ports and one 4-bit bidirectional I/O port. The two 8-bit ports can be linked to create a single 16-bit port. The counter/timer section of the Z8036 contains three 16-bit fully programmable counters. Two of the three counters can be linked to create a 32-bit counter. Each of the counters can be programmed to act as a timer.

The Z8030 Serial Communications Controller (SCC) contains two independent serial data channels. Each channel can transfer data at rates ranging from 0 to 1 megabit/second. Each channel has its own crystal oscillator, baud rate generator, and phase-locked loop. The channels are full-duplex and can operate asynchronously or bisynchronously. Each channel can be programmed to transmit and receive data in NRZ, NRZI, or FM formats.

This is a brief look at the new devices developed by

Zilog to support the Z8000. We'll look in more detail in Chapters 4 and 5.

SUMMARY

This has been a short look at the Z8000 family and its capabilities and resources. In the next four chapters, we examine in detail each of the features mentioned in this chapter.

Good reading!

2

Z8000 INSTRUCTION SET

INTRODUCTION

In this chapter, we discuss the Z8001 and Z8002 register set, data types, addressing modes (including a brief description of segmentation and memory management), and the instruction set.

REGISTERS

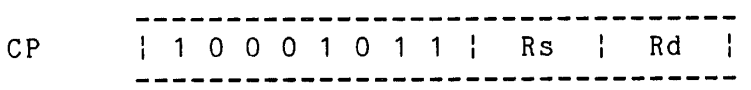
Before reading on, you should be familiar with the general register architecture of the Z8000 as outlined in Chapter 1. If you haven't read Chapter 1, please go back and do so.

General Purpose Registers

As already described, the Z8000 employs a very flexible register set. Recall that this flexibility allows the Z8000 to access registers in byte, word, long word, and extended word mode. The specific register used by an instruction is described by a register field in the instruction.

Let's use the Compare Register With Word (CP) instruction (shown on the next page) as an example. In register mode, the

CP instruction compares a word contained in one register with another word contained in another register. Notice that a source register pointer (Rs) and a destination register pointer (Rd) are shown as part of the instruction. (The words "source" and "destination" have no real meaning in this instruction.)



Both the Rs and Rd pointers use four bits of the instruction as register designators. The register(s) selected depend on the instruction's register mode. In the case of the CP instruction, registers R0 through R15 are used. If the instruction were:

```
CP      R6,R2
```

the Z8000 would compare the word contained in register R6 with the word contained in register R2. The result of the comparison would change the state of the Carry flag in the Flag and Control Word register.

Table 2-1 lists the register designations used in the Z8000. Any of the sixteen general purpose registers may be used as the accumulator.

Stack Pointers

Any of the sixteen general purpose registers may be used as stack pointers--except when using CALL and RET (return) instructions. Both CALL and RET imply that Z8001 register pair RR14 or Z8002 register R15 is the stack pointer register.

Z8001. When register pair RR14 is used as the Z8001 stack pointer, a seven bit segment number (bits 8 - 14) is contained in R14, and a 16-bit offset value is contained in R15. The Z8001 stack pointer is shown in Figure 2-1.

Table 2-1
Z8000 Register Designations

Designation Field				Byte Mode	Word Mode	Long Word Mode	Xtnd-Word Mode
0	0	0	0	RH0	R0	RR0	RQ0
0	0	0	1	RH1	R1		
0	0	1	0	RH2	R2	RR2	
0	0	1	1	RH3	R3		
0	1	0	0	RH4	R4	RR4	RQ4
0	1	0	1	RH5	R5		
0	1	1	0	RH6	R6	RR6	
0	1	1	1	RH7	R7		
1	0	0	0	RL0	R8	RR8	RQ8
1	0	0	1	RL1	R9		
1	0	1	0	RL2	R10	RR10	
1	0	1	1	RL3	R11		
1	1	0	0	RL4	R12	RR12	RQ12
1	1	0	1	RL5	R13		
1	1	1	0	RL6	R14	RR14	
1	1	1	1	RL7	R15		

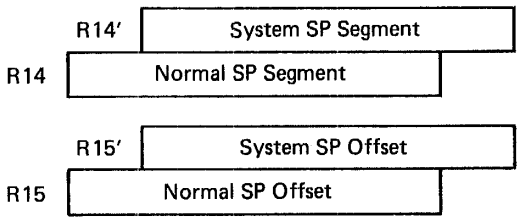


Figure 2-1 Z8001 stack pointer.

Notice in Figure 2-1 that there are actually two R14 and R15 registers. R14' and R15' make up the "system" stack pointer, while R14 and R15 make up the "normal" stack pointer. This dual stack pointer system allows the Z8001 to maintain two distinct stack pointers for system and normal operations. Also note that, even though register pair R14 is used as the stack pointer in CALL and RET instructions, it may also be used as a general purpose register. When used as general purpose registers, the phantom R14' and R15' are not accessible.

Again keep in mind that register pair RR14 is only implied as a stack pointer in CALL and RET instructions. Any other general purpose register pair may be designated as the stack pointer for PUSH and POP instructions.

Z8002. The same conditions apply concerning the Z8002 stack pointer. The difference is that, since the Z8002 does not require a segment number, only R15 and R15' (RR15) are used as the implied stack pointer.

Program Counter

The program counter registers are similar for the Z8001 and Z8002. Their only difference is that the Z8001 has an additional segment register. Figure 2-2 illustrates the Z8000 program counter registers.

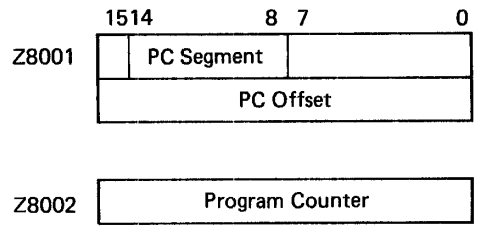


Figure 2-2 Z8000 program counter registers.

Z8000 instructions are always word aligned. Each instruction is expected to be a multiple of two addresses away from the old instruction. Therefore, the Z8000 program

counter is incremented by multiples of 2.

Any increment of the Z8001 PC offset register will not affect the PC segment register. Any time the PC offset register is incremented past FFFF it will simply wrap around to 0000 without affecting the segment register. There is no carry from the offset register to the segment register.

Upon reset, the Z8001 PC segment register is loaded from segment 0, location 0004, and the PC offset register is loaded from segment 0, location 0006. When the Z8002 is reset, its PC register is loaded from location 0004.

Flag and Control Word

The Z8000 Flag and Control Word (FCW) contains flag bits indicating the results of processor operations and control bits that set the operating mode of the processor.

The FCW, with one exception, is the same for the Z8001 and Z8002. That exception is the SEG bit, which, when set to 1, indicates that the Z8001 is operating in the segmented addressing mode. Figure 2-3 illustrates the Z8000 Flag and Control Word.

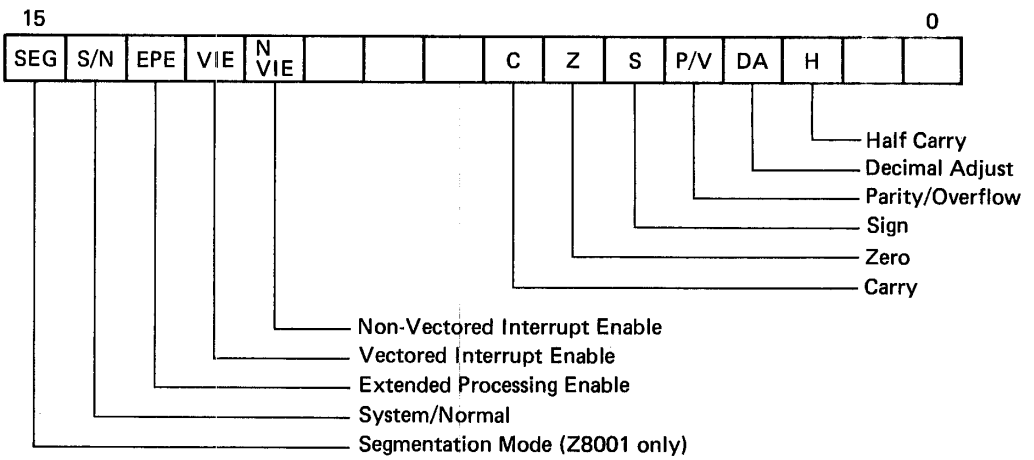


Figure 2-3 Flag and control word.

Notice in Figure 2-3 that the FCW is effectively divided into two separate bytes: flag (bits 2 through 7) and control (bits 11 through 15). The flag bits are modified as the Z8000 is running, according to the current operation. The control bits, however, define the way the Z8000 is to operate -- and are therefore privileged. That is, the Z8000 control bits can only be changed by using privileged instructions in the Z8000's "system" mode. There are a variety of software and hardware traps that may be used to prevent access to the control bits by non-privileged users. These methods are discussed later in this chapter.

Flag Bits. The Z8000 flag bits provide the standard types of information about processor operations. The following describes each flag bit of the Z8000 FCW.

H Half Carry (Bit 2)--The half carry flag is modified during arithmetic operations using byte operands. The half carry flag allows binary-coded decimal (BCD) operations. The flag is set when a carry from the least significant digit into the most significant digit occurs.

DA Decimal Adjust (Bit 3)--Like the half carry flag, the decimal adjust flag is used during BCD operations. Both the half carry and decimal adjust flags have little use in normal Z8000 operations.

P/V Parity/Overflow (Bit 4)--This bit acts as both a parity indicator and an overflow indicator. The P/V flag indicates whether byte data has an even or odd parity. If parity is even, the P/V flag is set to 1; if odd, to 0.

When used as an overflow indicator, the P/V bit is set to a 1 if both operands of an ADD operation have the same sign, but the result has the opposite sign. Overflow usually occurs if the result of the operation is too large to be represented as a signed 2's complement number. Certain operations set the P/V flag to 1 if a high-order operand bit changes value, or reset the P/V flag to 0 if the high-order operand bit does not change.

S Sign (Bit 5)--The Z8000 represents most numbers as 2's complements. Therefore, the most significant bit of a number is treated as that number's sign. When the most significant bit is a 1, the number is negative. If the most significant bit is a 0, the number is positive. The sign flag indicates the state of the most significant bit of a number following an arithmetic or logical operation.

Z Zero (Bit 6)--The zero flag is set to a 1 when all bits (including the sign bit) of a result are zero. The zero flag is reset to a 0 otherwise.

C Carry (Bit 7)--The carry flag indicates that a bit has been carried out of the high-order position of a byte, word, or long word following an arithmetic operation. The carry flag also indicates if a borrow occured during subtraction, by setting the flag to 1 if no borrow

occured or resetting the flag to 0 if a borrow did occur.

The Z8000 flags are set during normal Z8000 operations. The last part of this chapter describes each instruction and indicates which flags are modified during the execution of each instruction.

Control Bits. As was stated before, the control portion of the Z8000 FCW can only be altered by privileged instructions. The following describes each bit of the control byte.

NVIE Non-Vectored Interrupt Enable (Bit 11)--The Z8000 allows three types of external interrupts: non-maskable (highest priority), vectored (next highest priority), and non-vectored (lowest priority). The NVIE bit enables non-vectored interrupts. Non-vectored interrupts are detected when the NVI pin of the device goes low and acknowledged by the ST0-ST3 pins of the Z8000.

VIE Vectored-Interrupt Enable (Bit 12)--This bit of the FCW enables vectored interrupts. When a peripheral device issues a vectored interrupt (VI), and the VIE bit is set, the Z8000 acknowledges the interrupt. The interrupting device can then send a 16-bit "identifier" to the Z8000. See the discussion of the New Program Status Area Pointer later in this chapter for more information about vectored interrupts.

EPE Extended Processing Enable (Bit 13)--This bit enables seven "extended" Z8000 instructions. Extended instructions are instructions used to communicate with what is called an Extended Processing Unit. See the last part of this chapter for more information.

S/N System/Normal (Bit 14)--This control bit determines the Z8000's operating mode. If the S/N bit is set to 1, the Z8000 is in the system mode, and all instructions are accepted. If the S/N bit is reset to a 0, the Z8000 is in the normal mode. In this mode privileged instructions generate a software trap.

SEG Segmentation (Bit 15) (Z8001 only)--This bit allows the Z8001 to emulate the Z8002. If the SEG bit is set to 0, the Z8001 operates in a non-segmented mode. If set to a 1, the SEG bit allows segmented addressing.

New Program Status Area Pointer

In order to explain the function of the New Program Status Area Pointer (NPSAP), let's examine what happens when a vectored interrupt occurs.

1. When a vectored interrupt occurs (assuming the VIE bit of the Control and Status Word is set), the Z8000 completes the current instruction and fetches the next instruction. The new instruction is then discarded and the program counter is not incremented.

2. The Z8000 then begins the interrupt acknowledge sequence. Status lines ST0-ST3 acknowledge the vectored interrupt. The interrupting device then generates a 16-bit "identifier" word and sends it to the Z8000. The upper-order byte of the identifier is actually used to identify the interrupting device. The lower-order byte points to one of 256 interrupt servicing routines. (Other interrupts use the identifier in a different fashion. See Interrupts later in this chapter for more information.)

3. At this point, the Z8000 pushes the program counter, Flag and Control Word, and finally the identifier onto the current stack.

4. The Z8000 now reads the NPSAP register for the location of new processor status information. New processor status information can be located anywhere in memory. For this reason, the Z8001 NPSAP contains both a segment number and a high-order offset number (see Figure 2-5 later in this chapter). The Z8002 NPSAP contains only a 16-bit value.

5. Once the NPSAP is read, the Z8000 reads the FCW and program counter value from the new status area. In the case of a vectored interrupt, the program counter value is actually calculated from the value stored in the new processor status area and the low-order byte of the interrupting devices identifier. At this point, the Z8000 begins the interrupt handling routine.

6. When the interrupt handling routine is finished, the Z8000 returns to the stack and pops off the identifier, the old FCW, and the old program counter value. The

identifier is discarded, and the Z8000 continues with the original program.

As you can see from the discription, the New Processor Status Area Pointer is used to point to an area in memory containing new operating parameters for the Z8000. The NPSAP is changed by the system instruction LDCTL. Figure 2-4 illustrates the contents of the NPSAP, as well as the stack contents at interrupt time.

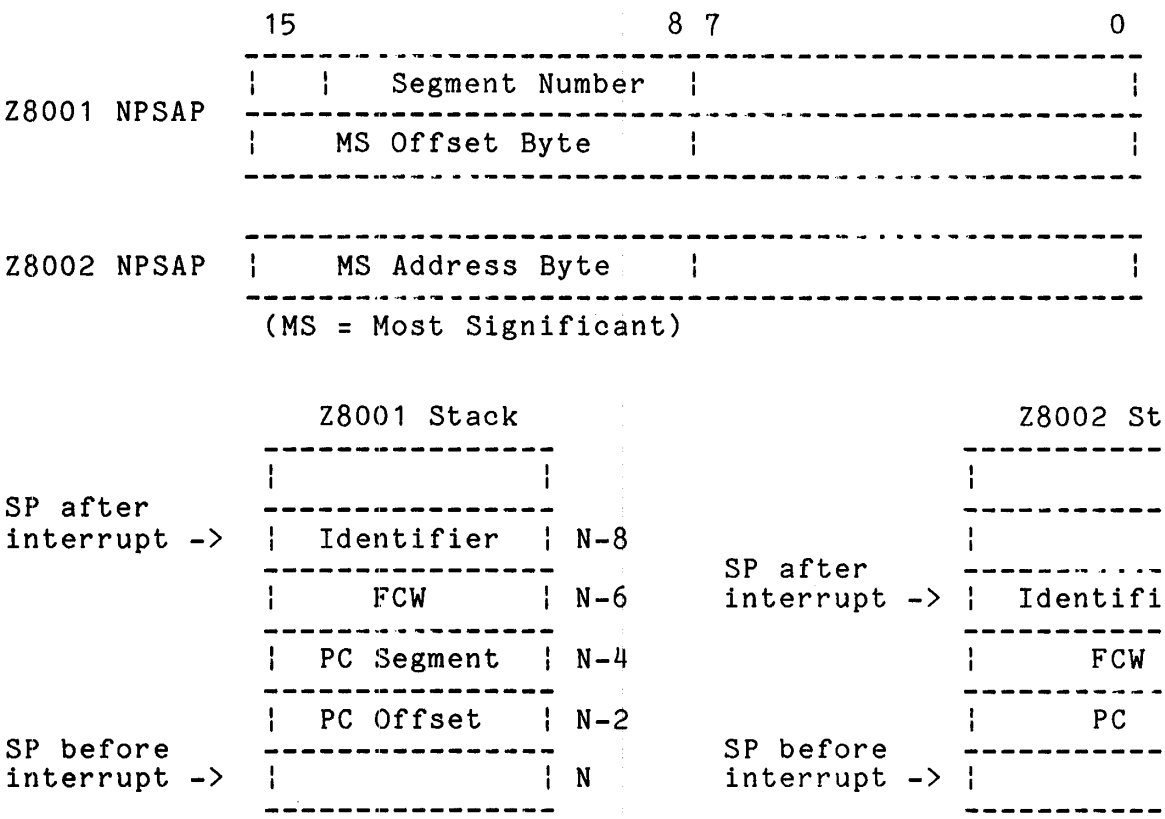
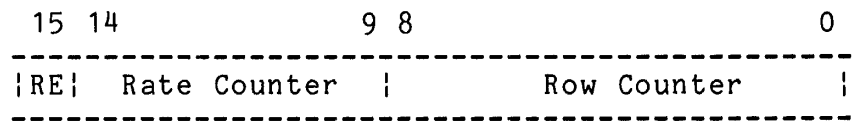


Figure 2-4

Refresh Counter

The Z8000 has a counter dedicated to dynamic memory refresh. The refresh counter exists as a Z8000 register. The counter has the following format:



The Z8000 refresh counter has a 9-bit memory row counter, a 6-bit rate counter, and a single enable bit (RE).

The rate counter is a programmable modulo-64 counter and is clocked at 25% of the CPU clock frequency. The row counter is clocked each time the rate counter overflows.

The automatic refresh feature of the Z8000 can be disabled by using the LDCTL instruction to write a 0 to the RE bit of the refresh counter. When the Z8000 is reset, the RE bit is reset to 0 (refresh is disabled) and must be set to 1 by initialization software.

ADDRESSING MODES

Z8000 instruction operands can be located in one of three places: in a register, in a memory location(s), or at an I/O location. Discounting I/O locations for the moment, there are eight explicit addressing modes and two implied addressing modes. The two implied addressing modes involve autodecrement and autoincrement in block and string manipulation instructions. The eight explicit addressing modes are described next in this chapter.

Explicit Addressing Modes

The following text makes use of the Z8000 EXB and LD commands. The EXB command causes the content of a Z8000 register to be exchanged or swapped with the contents of another register or a memory location. The LD command moves data to or from registers.

As you read the text, note that mention is made of "Zilog syntax." There are currently two manufacturers of the Z8000 devices: Zilog and Advanced Micro Devices (AMD). Each of these companies has adopted a different syntax for their assemblers. Later in this chapter, when we discuss each instruction in the Z8000 instruction set, these differences will be defined.

Register (R) Mode. In the register mode, the instruction's operand is located in one of the Z8000 registers. For example, the exchange byte instruction (EXB) exchanges a byte in one register with a byte in another register. Both registers are called out in the instruction, e.g.:

```
EXB         RH2,RH7
```

In this case, the contents of byte registers RH2 and RH7 are swapped.

Each Z8000 instruction specifies the length of the operand--in this case the length of the registers involved in the instruction. The EXB instruction operates on byte-length registers. The EX instruction performs the same operation, but on word-length registers.

Indirect Register (IR) Mode. When an instruction is given in the indirect register (IR) mode, the operand is contained at a memory address pointed to by one of the Z8000 general purpose registers (register pairs in the Z8001). The register or register pair containing the address of the operand is called out in the instruction.

Using the EXB instruction as an example again, if the instruction were given in the IR mode, the contents of the memory location pointed to by RH7 would be swapped with the contents of RH2. The syntax of the EXB instruction for this mode is the same as that for the register mode.

Any general purpose register (Z8002) except R0 or any general purpose register pair (Z8001) except RR0 can be designated in the IR addressing mode.

Direct Address (DA) Mode. In the direct address (DA) mode, the instruction itself contains the address of the operand. For example, (using Zilog syntax) the EXB instruction in DA mode looks like this for the Z8002:

```
EXB         R3,%13F0
```

As you can see, the address of the byte to be exchanged with R3 is located at address 13F0. Note also that "%" is used in Zilog syntax to indicate an address. Data values have no indicator.

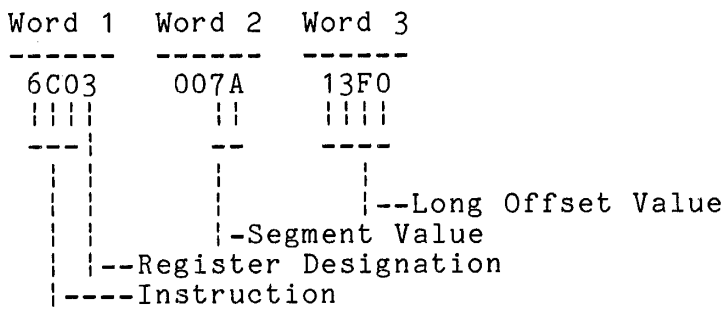
When the Z8001 is operated in segmented mode, the syntax is slightly more complicated. In the DA mode, the memory location or locations immediately following the instruction contain the 7-bit segment value and the offset of the source memory location. The Z8001 will recognize either a "long" offset (16-bits), or a "short" offset (8-bits).

With a long offset, the memory word immediately following the instruction contains the segment value, while the next memory word contains the 16-bit offset. Zilog syntax for the EXB command in the Z8001 looks like this:

```
EXB         R3,<<%7A>>%13F0
```

In this example, the contents of register R3 are swapped with the contents of segment 7A, offset location 13F0. The

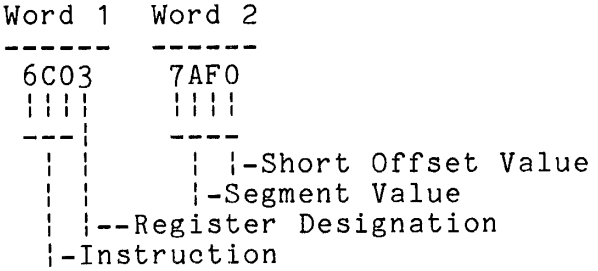
hexadecimal representation of the EXB command using a long offset looks like the following:



If the source offset address is from 0000 through 00FF, the Z8001 will allow a "short" offset to be used. Zilog syntax for the EXB instruction using a short offset looks like this:

```
EXB       R3,|<<%7A>>%F0|
```

Note the vertical bars indicated in the syntax. Zilog uses the bar to indicate a short offset value. The short offset form looks like the following:



Immediate (IM) Mode. In the immediate addressing mode, the instruction's operand is contained in the instruction itself. The EXB instruction does not have an immediate addressing mode. So let's look at an instruction that does--ADD.

The ADD instruction adds a source word to the contents of one of the Z8000 registers. Its syntax form in IM mode is:

```
ADD       R5,3055
```

In other words, ADD hexadecimal value 3055 to the contents of register R5. The sum will remain in register R5.

If the value to be added to R5 were a byte, the instruction would be changed to:

```
ADDB    RH5,4A
```

Finally, if a long word (32-bits) were to be added, the syntax would appear as follows:

```
ADDL    RR4,7FAC8970
```

Indexed (X) Mode. The indexed addressing mode uses the contents of a general purpose register and an address provided within the instruction to create the actual operand. The indexed mode is most easily explaned in terms of the Z8002.

Using the EXB instruction again, the address of the byte we want to exchange with one register is made up of the sum of the contents of another register and the address provided within the instruction. For example:

```
EXB     RL4,%30FF(RH7)
```

In this example, the contents of register RL4 are swapped with the memory location pointed to by the sum of the contents of RH7 and value 30FF (discounting any carry that might occur).

The Z8001 uses indexed addressing in the same way, except that you have the option of using a short or long offset. The segment value in the instruction is **not** involved in the addition. Let's look at the syntax using a long offset.

```
EXB     RL4,<<%1F>>%30FF(RH7)
```

In the long offset example, the actual address of the value to be exchanged with RL4 is located by adding 30FF to the contents of RH7. Any carry is ignored, and the segment value (1F) does not change.

Base Address (BA) Mode. Only a few of the Z8000 instructions use the BA mode, and all are in the register load group.

The base address mode makes use of a 16-bit register (Z8002) or register pair (Z8001) to contain a "base" address. The actual address of the instruction's operand is found by adding the base address to an unsigned integer contained within the instruction.

Let's use the LD instruction for this example. The LD instruction causes the contents of a Z8000 register to be loaded into memory. The following syntax is used by a Z8002 or by a Z8001 in non-segmented mode.

```
LD      R2,%30FF(R6)
```

In this case, the contents of register R2 are stored in the memory location determined by adding 30FF to the "base" address stored in register R6. The real usefulness of the BA

mode is that the content of R6 is not altered.

Z8001 syntax in segmented mode is very similar.

```
LD        R2,<<%04>>%30FF(R6)
```

The contents of register R2 are stored in segment 04, memory location R6 plus 30FF. There is no short offset version of the BA addressing mode.

Base Indexed (BX) Mode. The base indexed mode is similar to the base address mode, except that the actual memory location of the operand is generated by summing the contents of two registers designated within the instruction. For example, the syntax used by the Z8002 or the Z8001 in non-segmented mode is:

```
LD        R2,R4(R6)
```

In this case, the content of register R2 is stored at the address indicated by the sum of R4 and R6. Again, R6 is not altered.

The segmented Z8001 looks for the base address in a register pair that contains both the segment value and the offset value. For example:

```
LD        R2,RR4(R6)
```

The contents of R2 are stored within the memory segment pointed to by the high-order word of RR4, at the address specified by the addition of the offset (low-order word of RR4) plus the value of R6.

There is no short offset form of base indexed addressing.

Relative Addressing (RA) Mode. The relative addressing mode uses the Z8000 program counter plus a displacement value contained within the instruction to determine the operand's memory location. The displacement value is a signed integer, which allows the displacement to be plus or minus the program counter value. For example:

```
LD        R2,$-%02FA
```

This instruction tells the Z8000 to load register R2 into the memory location determined by subtracting 02FA from the current value of the program counter. What's important to know here is that the program counter is incremented as soon as the LD instruction is fetched, and that incremented value will determine the address. Any carry produced by adding the program counter and immediate value is ignored.

In the segmented Z8001, the value contained in the instruction is added to the program counter offset. The segment value remains unchanged.

Summary. This completes the description of the Z8000 addressing modes. Most are quite flexible. In the last portion of this chapter, each instruction is described, and the possible addressing modes listed.

Implied Addressing Modes

The addressing mode is implied only in string manipulation instructions and particularly in the Z8000's autoincrement and autodecrement functions. Implied addressing is a variation of the indirect register addressing mode. When an instruction uses a register or register pair to contain the operand's address, that register or register pair is automatically incremented or decremented at the completion of the instruction.

Let's use the OTIRB instruction as an example. The OTIRB instruction causes the byte contents of a memory location to be output to an I/O port. The register containing the address of the memory location is then incremented, and the action is repeated until another register used as a counter reaches zero. The assembly language syntax of the OTIR instruction looks like this:

```
OTIRB          @R5,@R2,R4
```

In this example, register R5 contains the address of the I/O port. Register R2 contains the starting address of the block of memory to be output to the port. Register R4 contains an integer indicating the number of data bytes to be sent to the port. Suppose, for example, the following values were loaded into the registers:

```
R5 = 235
R2 = 3000
R4 = 03
```

When the instruction began execution, the data contained at address 3000 would be sent to port 235. Then register R2 would be incremented to 3002, register R4 would be decremented to 02, and the next byte would be sent to port 235. This action repeats until R4 = 0.

Segmentation and Memory Management

By now you've seen the pattern that Z8001 instructions take--that is, a segment value and a memory offset value are always considered in any instruction. Without the assistance of a Memory Management Device, you must be very specific when addressing memory. The thing to keep in mind is that there can be up to 128 segments of memory, each containing 64K

offset values. The Z8001 can access any of these locations, as long as there is memory there to be accessed.

In the next part, we talk about interrupts and traps. One of the software traps involves segmentation errors. Read this text carefully for some insight into what the Z8001 expects.

INTERRUPTS AND TRAPS

In general, interrupts are events that occur outside the Z8000, but require the Z8000's attention. Traps are internal interrupts, generated when an attempt is made to execute an instruction in an environment that prohibits that instruction. In addition, you can inhibit some interrupts, but traps can never be inhibited.

In order to understand how interrupt and trap servicing routines are processed, we have to look closer at the New Program Status Area. Recall that the New Program Status Area (NPSA) is pointed to by the contents of the New Program Status Area Pointer within the Z8000. The New Program Status Area Pointer is changed with the LDCTL instruction, which is a system, or privileged instruction.

Figure 2-4 illustrates the Z8001 NPSA. As you can see, the NPSA is divided into eight areas: an unused area, an unimplemented area, privileged instruction trap, system call trap, segmentation trap, non-maskable interrupt, non-vectored interrupt, and vectored interrupt. Each area contains a repeat of the unimplemented area in Figure 2-4. That is, each area contains a reserved address, a flag and control word address, and program counter information. Depending on the interrupt or trap type, the Z8000 will expect the address of the proper trap or interrupt handling subroutine to be located within its designated area. The Z8000 calculates the proper location from the base address in the New Program Status Area Pointer (which, in this case, would be address 00).

The New Program Status Area for the Z8002 (shown in Figure 2-5) is similar to that of the Z8001, with the exception that the program counter location is only one address long.

Interrupts

The Z8000 supports three types of interrupts: non-maskable, vectored, and non-vectored. Both the vectored and non-vectored interrupts must be enabled by their associated control bits in the Flag and Control Word register, explained earlier in this Chapter.

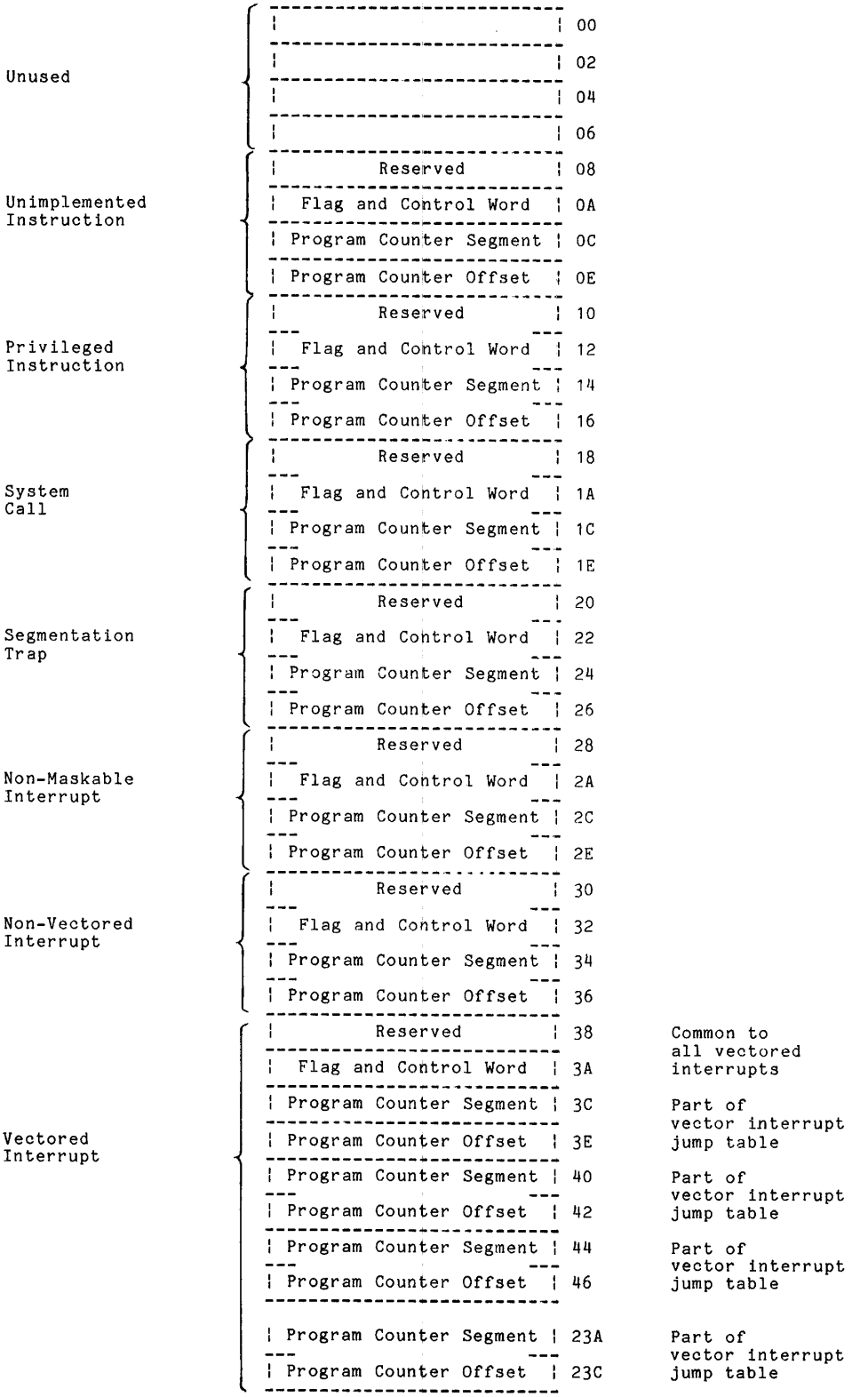


Figure 2-5 Z8001 new program status area.

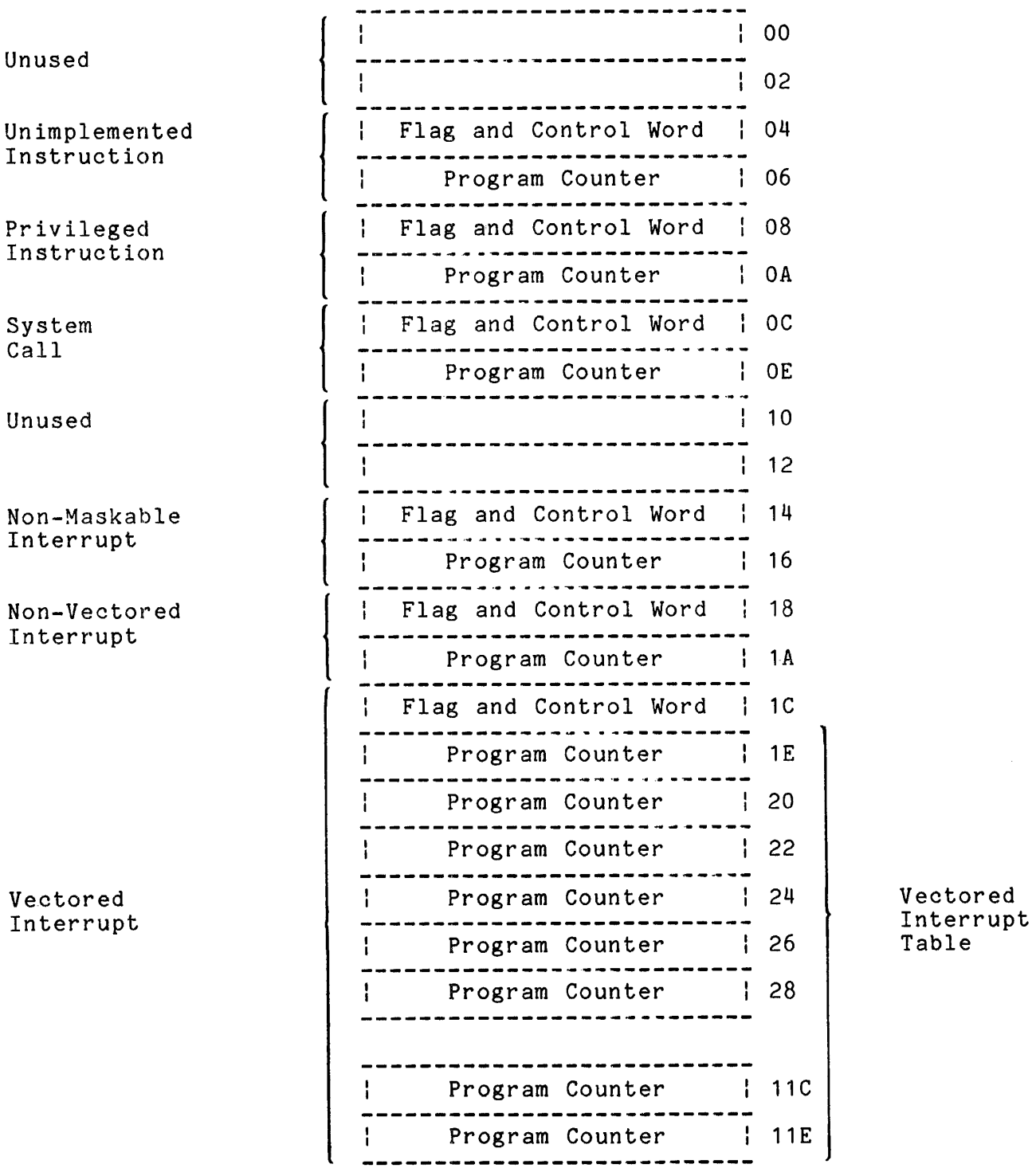


Figure 2-5A Z8002 new program status area.

Non-Maskable Interrupts. Non-maskable interrupts have the highest priority. No matter what the Z8000 is doing, if a non-maskable interrupt (NMI) occurs, the Z8000 will stop execution and service the interrupt.

Let's look at the procedure for handling an NMI.

1. An NMI occurs.

2. The Z8000 completes the current instruction, fetches and then discards the next instruction. The Program Counter is not incremented.

3. Status lines ST0-ST3 acknowledge the NMI.

4. The interrupting device reads the acknowledgment, then generates its 16-bit "identifier."

5. The Z8000 reads the 16-bits of the identifier. The upper 8-bits of the identifier indicate the interrupting device.

6. The Z8000 pushes the Program Counter, Flag and Control Word, and identifier onto the current stack.

7. The Z8000 reads the New Program Status Area Pointer for the location of the new processor status information. (In the organization shown in Figure 2-5, the Z8001 would look to locations 28 through 2E.)

8. The Z8000 reads the new processor status information, then jumps to the memory location pointed to and begins execution.

   There can actually be several NMI servicing routines. The beginning of the servicing routine can calculate the actual routine required by any number of methods. For example, the lower eight bits of the identifier may be used to determine the location of a servicing routine by multiplying the identifier value and the new Program Counter value, then jumping to the address thus calculated. Or the identifier may be used to look up the correct address of the servicing routine by pointing to a location in a look-up table.

9. When the interrupt servicing routine is completed (indicated by a Return From Interrupt instruction--IRET), the Z8000 pops the identifier, the old Flag and Control Word, and the old Program Counter value from the stack and continues with the original program.

Non-Vectored Interrupts. Although the lowest priority, non-vectored interrupts are handled in exactly the same way as

non-maskable interrupts. The only difference is that non-vectored interrupts can be disabled by resetting the NVIE bit within the Flag and Control Word.

Vectored Interrupts. Vectored Interrupts have the second-highest interrupt priority within the Z8000. Vectored interrupts do not require any calculation of the address of a servicing routine, but instead provide the address of the servicing routine automatically. Notice in Figure 2-5 that the NPSA for vectored interrupts contains an interrupt table. The table is made up of a series of program counter segments and offsets. When the vectored interrupt occurs, the proper program counter value is selected by the interrupting devices identifier. For example, if the interrupting device gave an identifier value of xx02, the correct program counter values would be located at addresses 44 and 46. The Z8001 would then jump to the location pointed to by addresses 44 and 46.

Traps

The Z8000 uses four instruction traps: system call, unimplemented instruction, privileged instruction in normal mode, and segmentation error. Whenever a trap occurs, the Z8000 jumps to a servicing routine in much the same way as if an interrupt had occured. Traps, however, cannot be masked. Traps also generate an identifier, except that the identifier is provided within the instruction that caused the trap. Refer again to Figure 2-5 while reading the following.

Unimplemented Instructions. Unimplemented instructions (those instructions that are not found within the Z8000 instruction set) cause a software trap to occur. When the Z8000 tries to execute an unimplemented instruction, it reacts in about the same way as if an NMI had occured. The current status information is pushed onto the stack, and the Z8000 fetches the new processor information from the NPSA and jumps to the correct servicing routine.

Unimplemented instructions can sometimes be put to good use in debugging a program. You can insert unused instructions, ZZ for example, in your program that will cause the Z8000 to jump to a servicing routine. Part of that routine might be to dump register contents out to a terminal or hard-copy unit. It's like setting breakpoints within the program.

Privileged Instructions. Privileged instructions are instructions that can only be executed when the Z8000 is in "system mode." If the Z8000 is in "normal" mode and encounters a privileged instruction, this trap will be generated. Again, the trap is handled in the same way as an NMI.

System Call Traps. These traps are generated by the SC instruction. The SC instruction is generally used to request some system operation. The SC instruction is sixteen bits long, and the lower eight bits are used to identify the call. Therefore, there can be up to 256 different system calls. In other words, you can request 256 different servicing routines. Typically, when an SC instruction occurs, the servicing routine will pop the identifier off the system stack to see what kind of service is required.

Segmentation Traps. Segmentation traps occur when the Z8001 is used with a Memory Management Unit (MMU). The MMU has discovered some error and informs the Z8001 by issuing a segmentation trap. The MMU also provides an identifier so that the servicing routine can determine the cause of the trap.

Summary

There you have interrupts and traps. The interrupt and trap structure of the Z8000 is quite flexible and provides a lot of good tools for the programmer.

More information on the use of interrupts and traps can be found in Chapter 3, Device Interface.

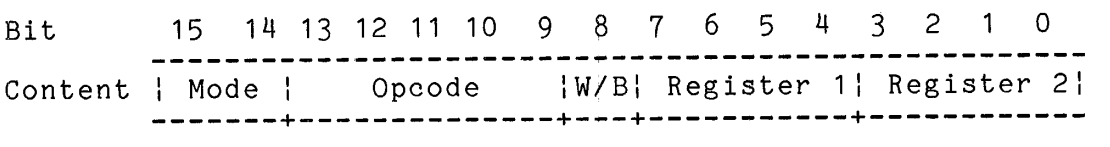
THE Z8000 INSTRUCTION SET

The remainder of this chapter is devoted to the Z8000 instruction set.

Instruction Format

The length of Z8000 instructions range from one to five 16-bit words long. The instruction opcode is always within the first word of the instruction.

Generally, a single-word Z8000 instruction is divided into these sub-parts:



The mode field, along with the opcode and register fields, determine the addressing mode used by the instruction.

The W/B field is used to specify whether the instruction uses a word or byte operand. The register 1 and register 2 fields usually designate general purpose registers. Of course, each instruction has its particular requirements as to organization of information within the instruction. The format, however, is generally the same.

Let's look a little closer at how the addressing mode is selected.

Bits 15, 14, 13, 12, and 7 through 4 are used to select the addressing mode. Table 2-2 lists the addressing modes and the states of relevant bits. An "x" in the table indicates a "don't care" state. That is, the state of the bit is not defined.

Also note from Table 2-2 that, for the IM, RA, and DA addressing modes, the register field must be all zeros. This is because register R0 is not interpreted as a general purpose register in these addressing modes. Notice that the IR, BA, X, and BX addressing modes will not allow register R0 to be selected as a general purpose register.

The exact bit pattern for each Z8000 register is given earlier in Table 2-1.

Table 2-2
Z8000 Address Mode Selection

Addressing Mode	Fields		
	Mode (15,14)	Opcode (13,12)	Register (7,6,5,4)
R	1 0	x x	x x x x
IM	0 0	any but 1 1	0 0 0 0
IR	0 0	same as IM	any but 0 0 0 0
RA	0 0	1 1	0 0 0 0
BA	0 0	1 1	same as IR
DA	0 1	x x	0 0 0 0
X	0 1	same as IM	same as IR
BX	0 1	1 1	same as IR

Condition Codes

Some instructions, particularly branch or jump instructions, require a 4-bit field within the instruction to specify flag settings to determine whether the instruction will be executed. In other words, instruction execution is conditional on the state of the flags.

As explained earlier in this chapter, the Z8000 uses six flags to indicate the result of an operation: half carry (H), decimal adjust (DA), parity/overflow (P/V), sign (S), zero (Z), and carry (C). The condition code indicates the required state of these flags. Table 2-3 lists the possible condition codes and their meanings.

Table 2-3
Condition Codes

Condition Code Field	Flag Affected	Meaning
1 1 1 0	Z=0	NZ -- Not Zero
0 1 1 0	Z=1	Z -- Zero
1 1 1 1	C=0	NC -- No Carry
0 1 1 1	C=1	C -- Carry
1 1 0 0	P/V=0	PO -- Parity Odd
0 1 0 0	P/V=1	PE -- Parity Even
1 1 0 1	S=0	PL -- Plus
0 1 0 1	S=1	MI -- Minus
1 1 1 0	Z=0	NE -- Not Equal
0 1 1 0	Z=1	EQ -- Equal
1 1 0 0	P/V=0	NOV -- Overflow is reset
0 1 0 0	P/V=1	OV -- Overflow is set
1 0 0 1	S XOR P/V=0	GE -- Greater than or equal
0 0 0 1	S XOR P/V=1	LT -- Less than
1 0 1 0	Z OR (S XOR P/V)=0	GT -- Greater than

Table 2-3 (Cont.)
Condition Codes

Condition Code Field	Flag Affected	Meaning
0 0 1 0	Z OR (S XOR P/V)=1	LE -- Less than or equal
1 1 1 1	C=0	LGE -- Logical greater than or equal
0 1 1 1	C=1	LLT -- Logical less than
1 0 1 1	C=0 AND Z=0	LGT -- Logical greater than
0 0 1 1	C=1 OR Z=1	LLE -- Logical less than or equal
1 0 0 0	UNCONDITIONAL	

The Instructions

The rest of this chapter describe the instruction set of the Z8000. The instructions are divided into operational groups and listed alphabetically within those groups.

Each instruction is described in a similar manner. Figure 2-6 illustrates how the information is presented.

1. The name of each instruction is given in bold type, followed by whether the instruction is a "system," "normal," or "extended" instruction.

2. The definition for each instruction mnemonic is given next.

3. The format of the instruction illustrates the bit pattern for that instruction. Note that the bit pattern for each addressing mode is given. **Note** that the DA and X addressing modes simply list address. The actual number of words used in the specific addressing mode depends on whether you're using the Non-Segmented (NS), Segmented--Short Offset (SSO), or Segmented--Long Offset (SLO) modes.

4. Where more than two addressing modes are possible, the addressing modes are listed here.

5. Each instruction's operation is given next. If an operation is not listed, that the operation is too complex to illustrate graphically, and is given in the description.

6. When flags are affected by an instruction, they are listed here.

7. Clock cycles are given next, along with applicable addressing modes.

8. The description of each instruction is given, followed by:

9. an example of how the instruction works. Zilog assembly language format is used throughout.

① **DBJNZ**
(Normal)

② **Definition:**
Decrement Byte Register and Jump if Not Zero

③ **Format:**

④ RA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Frxx 1	1	1	1	Register c				0	Displacement						

⑤ **Operation:**
Z8001/Z8002

```
Register c<0:7> <-- Register c<0:7> - 1
If Register c<0:7>≠0
Then PC <-- Updated PC - 2x displacement
Otherwise PC <-- Updated PC
```

⑥ **Flags Affected:** None

⑦ **Clock Cycles** = 11

⑧ **Description:**
The DBJNZ instruction is used to do a qualified jump to another address, using the Relative Addressing mode.

When the DBJNZ instruction is executed, byte Register 'c' is decremented by one. If the contents of Register 'c' do not equal 0, a relative address jump is performed.

When Register 'c' ≠ 0, the Program Counter is incremented by two, the 7-bit unsigned displacement value is shifted left one (its value is doubled), and the new displacement value is subtracted from the updated Program Counter contents to create a new Program Counter value.

If the byte-register contents do equal 0, the Program Counter is incremented by two, and execution continues.

⑨ **Example:**

```
DBJNZ    RH4,%20    !Decrement RH4.  If RH4 >=1 then
                    !increment PC by two, subtract 40
                    !new PC, and jump to new location.
                    !If RH4=0, increment PC by two,
                    !and continue execution.
```

Figure 2-6

CPU Control

The CPU Control group of instructions controls the operation of the Z8000. These instructions perform operations such as HALT, NOP, set and clear flags, set interrupt masks, load the flag and control word, set up the new program status area, set the stack pointers, and control multi-microprocessor functions. Table 2-4 lists the instructions within the CPU Control group.

Table 2-4
CPU Control Group Instruction Summary

Mnemonic	Function
DI	Disable Interrupts
EI	Enable Interrupts
HALT	Halt Program Execution
LDCTL	Load Control Into/From Register
LDCTLB	Load Flag Byte Into/From Register
MBIT	Multi-microprocessor Bit Test
MREQ	Multi-microprocessor Request
MRES	Multi-microprocessor Reset
MSET	Multi-microprocessor Set
NOP	No Operation
POP	Pop A Word From The Stack
POPL	Pop A Long Word From The Stack
PUSH	Push A Word Onto The Stack
PUSHL	Push A Long Word Onto The Stack
COMFLG	Complement Status Flags
RESFLG	Reset Status Flags
SETFLG	Set Status Flags

DI
(System)

Definition:
Disable Interrupts

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7B0x	0	1	1	1	1	1	0	0	0	0	0	0	0	0	VI	NVI

Operation:
Z8001/2

```
FCW<11> <-- 0 for NVI=0
FCW<11> <-- FCW<11> for NVI=1
FCW<12> <-- 0 for VI=0
FCW<12> <-- FCW<12> for VI=1
```

Flags Affected: None

Clock Cycles = 7

Description:
The DI instruction disables non-vectored interrupts (NVI), vectored interrupts (VI), or both.

```
Bit 0 = 0 -- Disable non-vectored interrupts
Bit 0 = 1 -- No Effect
Bit 1 = 0 -- Disable vectored interrupts
Bit 1 = 1 -- No Effect
```

Example:

```
DI      VI      !Disable vectored interrupts
DI      VI,NVI  !Disable vectored and non-vectored interrupts
```

EI
(System)

Definition:
Enable Interrupts

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7B0x	0	1	1	1	1	1	0	0	0	0	0	0	0	1	VI	NVI

Operation:
Z8001/2

FCW<11> <-- 1 for NVI=0
FCW<11> <-- FCW<11> for NVI=1
FCW<12> <-- 1 for VI=0
FCW<12> <-- FCW<12> for VI=1

Flags Affected: None

Clock Cycles = 7

Description:
The EI instruction enables non-vectored interrupts (NVI), vectored interrupts (VI), or both.

Bit 0 = 0 -- Enable non-vectored interrupts
Bit 0 = 1 -- No Effect
Bit 1 = 0 -- Enable vectored interrupts
Bit 1 = 1 -- No effect

Example:

```
EI      VI      !Enable vectored interrupts
EI      VI,NVI  !Enable vectored and non-vectored interrupts
```

HALT
(System)

Definition:
Halt Program Execution

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7A00	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0

Operation:
None

Flags Affected: None

Clock Cycles = 8 + 3n*
* Interrupts are recognized at the end of each 3 cycle period.

Description:
When the HALT instruction is executed, program execution stops. Register and status contents are saved, and the Z8000 refresh counter continues to run. Program execution is restarted by an external RESET signal or by an interrupt. Interrupt requests are serviced at the end of each 3-cycle period.

Example:

```
        HALT
```

LDCTL
(System)

Definition:
Load Designated Control Word Into/From a Register.

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8Crx	1	0	0	0	1	1	0	0	Register				1/0	CW		

Operation:
Z8001/2

(If bit 3=0) Register<0:15> <-- CW<0:15>
(If bit 3=1) CW<0:15> <-- Register<0:15>

Flags Affected:
Flags are affected only if the Flag and Control Word is selected as a destination. See below.

Clock Cycles = 7

Description:
The LDCTL instruction is used to manipulate the CW field of the instruction. If bit 3=0, the control word (CW) selected is moved to the designated word register. If bit 3=1, the contents of the designated word register are moved into the selected CW.

CW Field	Control Word
0 0 0	--
0 0 1	--
0 1 0	Flag and Control Word (FCW)
0 1 1	Refresh Register (bits 1 through 8)
1 0 0	NPSAP Segment
1 0 1	NPSAP Upper Offset
1 1 0	R14
1 1 1	R15

Example:

```
LDCTL   R2,FCW          !Move FCW to register R0.
LDCTL   FCW,R2          !Move contents of R2 to FCW.
LDCTL   R2,REFRESH      !Move contents of Refresh Register
                        !to R2.
```

LDCTLB
(Normal)

Definition:
Load Flag Byte Into/From Register

Format:

```
        15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
       ---------------------------------------------------------------
8Cr1   | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |    Register   | 0 | 0 | 0 | 1 |
       ---------------------------------------------------------------
```

Z8001/2

```
Register<0:7> <-- FCW<0:7>
          or
FCW<0:7> <-- Register<0:7>
```

Flags Affected:
All flags are affected if a register byte is moved into the flag portion of the Flag and Control Word (FCW).

Clock Cycles = 7

Description:
The flag byte of the Flag and Control Word is moved to or loaded from the general purpose 8-bit register designated in the instruction.

Example:

```
LDCTLB  RH2,FLAGS
LDCTLB  FLAGS,RH2
```

MBIT
(System)

Definition:
Multi-microprocessor Bit Test

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7B0A	0	1	1	1	1	0	1	1	0	0	0	0	1	0	1	0

Operation:
Z8001/2

S Flag <-- μI
Z Flag <-- 0

Flags Affected:
S: Set if μI asserted, else reset.
Z: Undefined.
All other flags unaffected.

Clock Cycles = 7

Description:
The MBIT instruction tests the state of the Z8000 μI input line. The μI line is used in multi-microprocessor environments. If μI is asserted low, the S flag is set. Otherwise, the S flag is reset. The Z flag state is undefined.

Example:

```
        MBIT
```

MREQ
(System)

Definition:
Multi-microprocessor Request

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7BrD	0	1	1	1	1	0	1	1	Register				1	1	0	1

Operation:

Flags Affected:
See Description

Clock Cycles = 12 + 7n
(n = number of times Register is decremented)
(n = 0 if initial state of μI = 1)

Description:
The MREQ instruction is used by the Z8000 in multi-microprocessor environments to request the use of a shared resource. The MREQ instruction tests the state of the μI line to determine if another microprocessor is using the resource. If the μI line is asserted, the MREQ instruction assumes the resource is being used, and the instruction terminates, setting the S and Z flags to 0.

If the μI line is not asserted, the Z8000 asserts the μO line, and begins to decrement the 16-bit general purpose register every 7 clock cycles. After the register contents reach zero, the Z8000 again tests the μI line. If the resource request is not approved, the μI line is remains in its original state, the MREQ instruction is terminated, the Z flag is set, and the S flag is reset.

If, when the register reached zero and the μI line was asserted, the Z8000 assumes the request was approved. The μO line remains asserted, and the Z and S flags are set.

The states of the Z and S flags indicate the outcome of the MREQ instruction as follows:

Flag		Description
Z	S	
0	0	μI line was asserted at beginning of instruction. Instruction terminated.
1	0	μI line was not asserted after register=0, but request was denied. Instruction terminated
1	1	Resource request approved.

Example:

```
        MREQ    R1
```

MRES
(System)

Definition:
Multi-microprocessor Reset

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7B09	0	1	1	1	1	0	1	1	0	0	0	0	1	0	0	1

Operation:
Z8001/2

μO <-- 0

Flags Affected: None

Clock Cycles = 5

Description:
The MRES instruction is used to indicate that a shared resource is not required. When the MRES instruction is executed, the μO line is reset (held high).

Example:

```
    MRES
```

MSET
(System)

Definition:
Multi-microprocessor Set

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7B08	0	1	1	1	1	0	1	1	0	0	0	0	1	0	0	0

Operation:
Z8001/2

μO <-- 1

Flags Affected: None

Clock Cycles = 5

Description:
The MSET instruction is used to force the state of the μO line. When MSET is executed, the μO line is forced low (asserted), regardless of the state of the μI line.

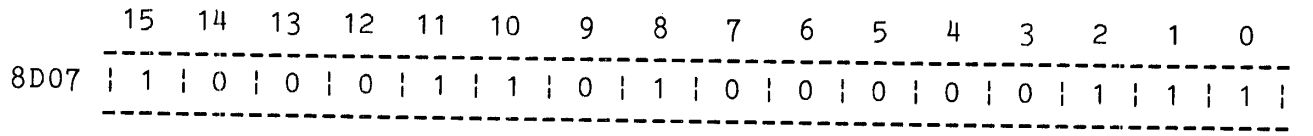
Example:

```
    MSET
```

NOP
(Normal)

Definition:
No Operation

Format:



Operation:

Flags Affected: None

Clock Cycles = 7

Description:
When the NOP instruction is executed, no operation takes place, however the Program Counter is incremented by two.

Example:

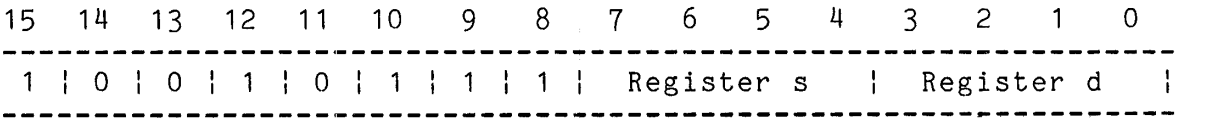
```
NOP
```

POP
(Normal)

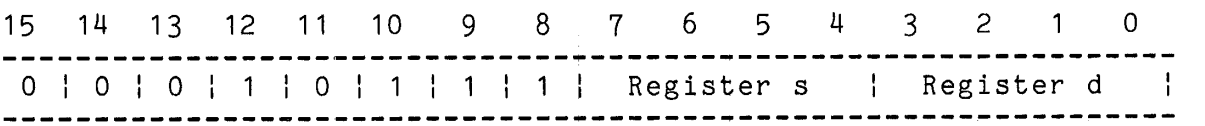
Definition:
Pop Word Off Designated Stack

Format:

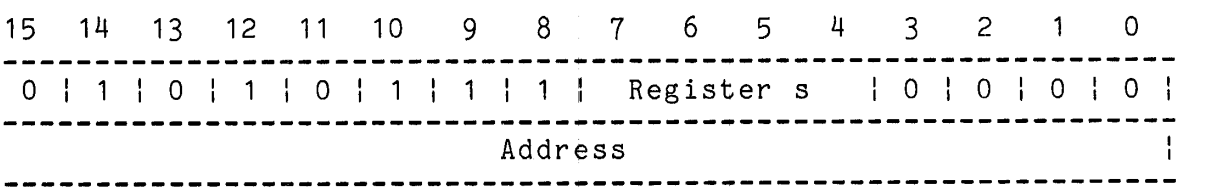
R
97rr



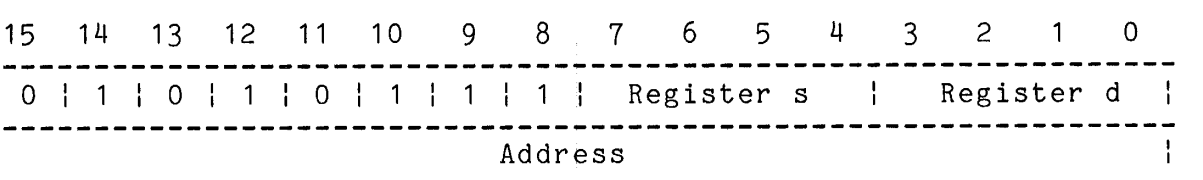
IR
17rr



DA
57r0



X
57rr



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	8
IR - S,NS	12
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:

When POP is executed, the memory word pointed to by Register 's' is loaded into destination Register 'd' or into the appropriate destination address, depending on the addressing mode. (The registers indicated can be any general purpose register except R0.) Register 's' acts as the stack pointer.

Once the data is transferred, Register 's' is incremented by two.

Example:

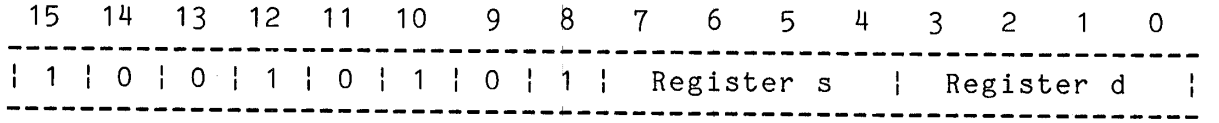
```
    POP     R4,R12      !The contents of the memory location
                        !pointed to by R12 (the stack
                        !pointer) are popped into register
                        !R4.  (Register Mode)
```

POPL
(Normal)

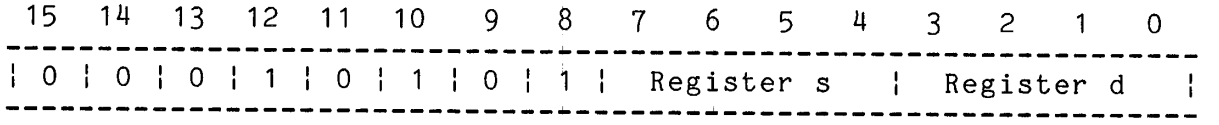
Definition:
Pop Long Word Off Designated Stack

Format:

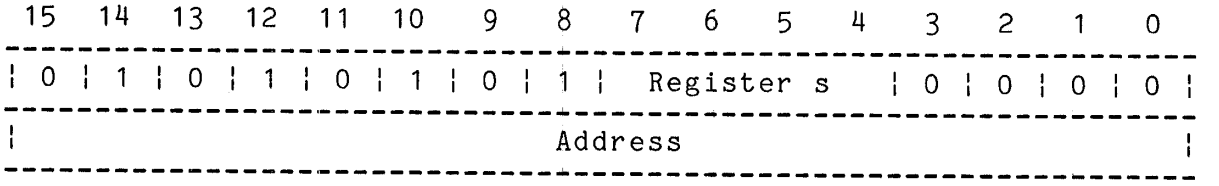
R
95rr



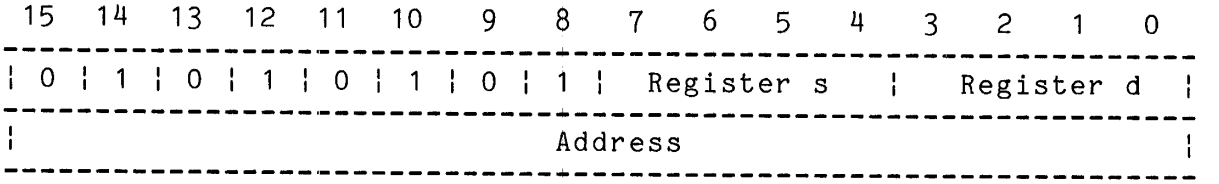
IR
15rr



DA
55r0



X
55rr



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	12
IR - S,NS	19
DA - NS	22
DA - SSO	23
DA - SLO	25
X - NS	23
X - SSO	23
X - SLO	26

Description:
When POPL is executed, the long memory word pointed to by Register 's' is loaded into destination Register 'd' or into the appropriate destination address, depending on the addressing mode. (The registers indicated can be any general purpose register except RR0.) Register 's' acts as the stack pointer.

Once the data is transferred, Register 's' is incremented by four.

Example:

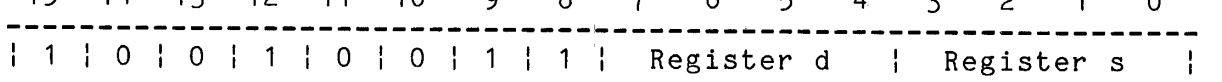
```
    POPL    RR4,RR12    !The contents of the memory location
                        !pointed to by RR12 (the stack
                        !pointer) are popped into register
                        !RR4.  (Register Mode)
```

PUSH
(Normal)

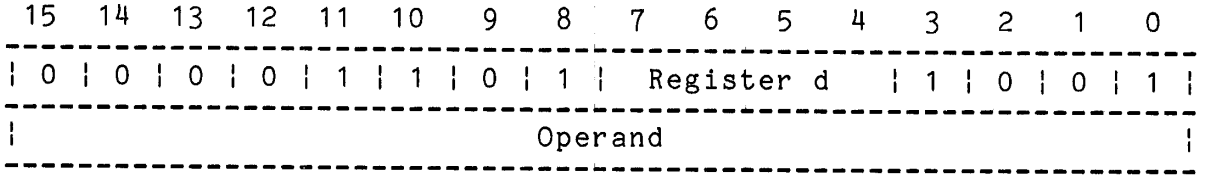
Definition:
Push Word Onto Designated Stack

Format:

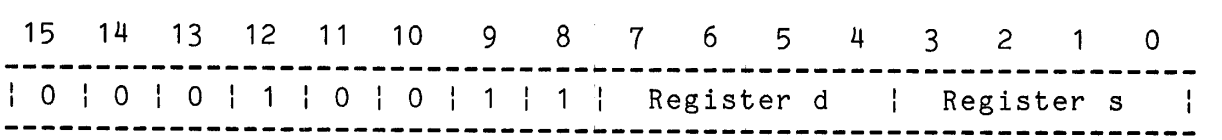
R
93rr



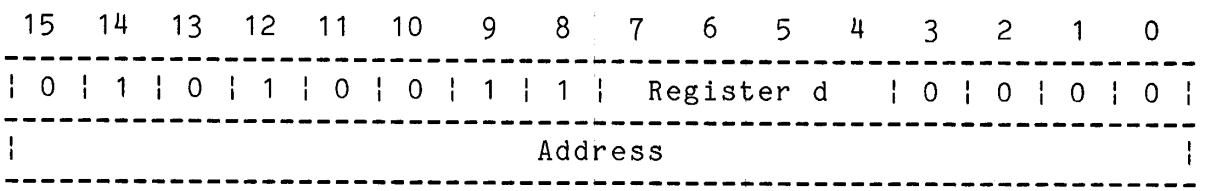
IM
0Dr9



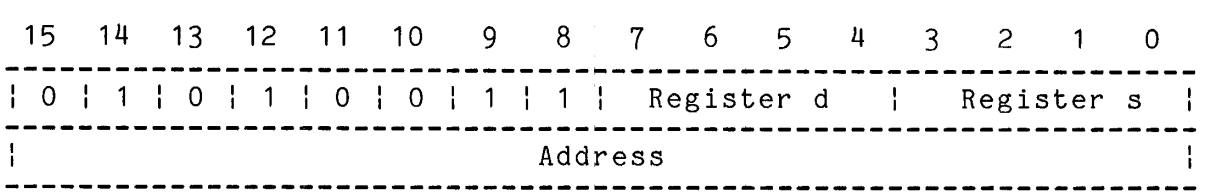
IR
13rr



DA
53r0



X
53r0



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	9
IM - S,NS	12
IR - S,NS	13
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

Description:
When a PUSH instruction is executed, the following occurs:

1. The contents of Register 'd' are decremented by two.
2. The source word operand (Register 's', an immediate operand, or a source address) is loaded into the location now pointed to by Register 'd'.

Any general purpose register excluding R0 can be specified as the stack pointer (Register 's'). The source operand is determined by the addressing mode used.

Example:

```
PUSH    R2,3055                 !PUSH 3055 into stack pointer
                                !R2. (Immediate Mode)

PUSH    R2,<<%1F>>%30FF(R4)     !PUSH the contents of segment
                                !1F, location 30FF plus the
                                !contents of register R4, into
                                !register R2 (stack pointer).
                                !(Indexed Mode - Long Offset)
```

PUSHL
(Normal)

Definition:
Push Long Word Onto Designated Stack

Format:

R
91rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	1	Register d				Register s			

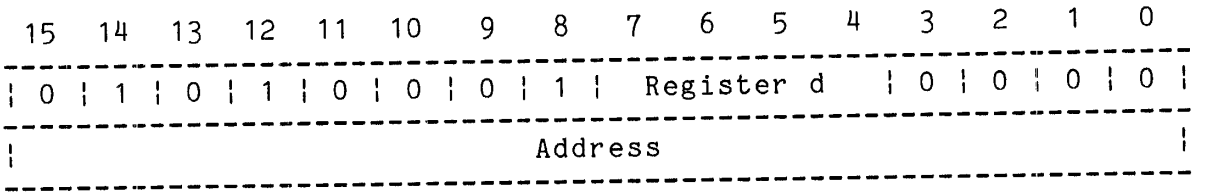
IM
0DrB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	Register d				1	0	1	1
Operand															
Operand															

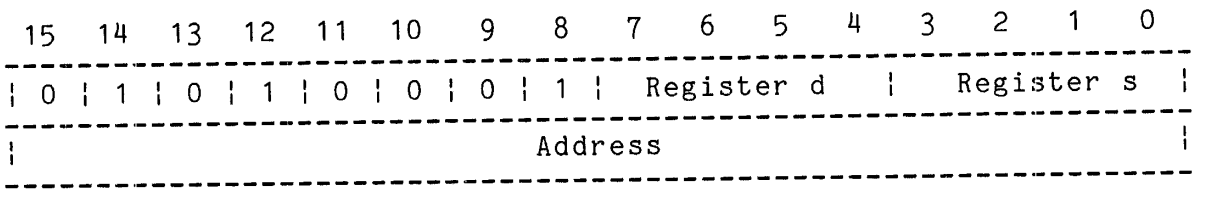
IR
11rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	Register d				Register s			

DA 51r0



X 51r0



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	12
IM - S,NS	19
IR - S,NS	20
DA - NS	20
DA - SSO	21
DA - SLO	23
X - NS	21
X - SSO	21
X - SLO	24

Description:
When a PUSHL instruction is executed, the following occurs:

1. The contents of Register 'd' are decremented by four.
2. The source long-word operand (Register 's', an immediate operand, or a source address) is loaded into the location now pointed to by Register 'd'.

Any general purpose register excluding RR0 can be specified as the stack pointer (Register 's'). The source operand is determined by the addressing mode used.

Example:

```
    PUSHL   RR2,3055A877            !PUSH 3055A877 into stack
                                    !pointer RR2. (Immediate Mode)

    PUSHL   RR2,<<%1F>>%30FF(RR4)   !PUSH the contents of segment
                                    !1F, location 30FF plus the
                                    !contents of register RR4, into
                                    !register RR2 (stack pointer).
```

COMFLG
(Normal)

Definition:
Complement Flags

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8Dx5	1	0	0	0	1	1	0	1	C	Z	S	PV	0	1	0	1

Operation:
See Description

Flags Affected: See below

Clock Cycles = 7

Description:
The COMFLG instruction allows the C, Z, S, or PV flags to be complemented. If the appropriate bit in the instruction is set to 1, the flag designated is complemented. If the bit is reset to 0, the designated flag is not complemented.
The DA and H flags remain unaffected.

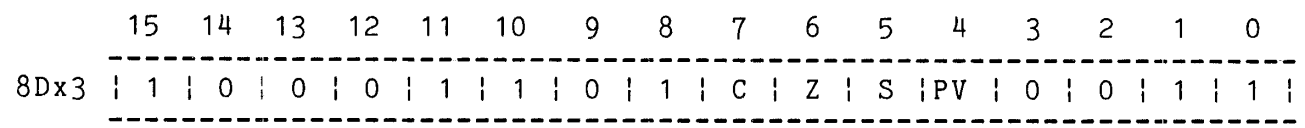
Example:

```
    COMFLG  Z,S          !Complement flags Z and S

    COMFLG  C,Z,S,PV     !Complement flags C, Z, S, and PV
```

RESFLG
(Normal)

Definition:
Reset Flags

Format:



Operation:
See Description

Flags Affected: See below

Clock Cycles = 7

Description:
The RESFLG instruction allows the C, Z, S, or PV flags to be reset. If the appropriate bit in the instruction is set to 1, the flag designated is reset. If the bit is 0, the designated flag is not reset.

The DA and H flags remain unaffected.

Example:

```
    RESFLG  Z,S          !Reset flags Z and S

    RESFLG  C,Z,S,PV     !Reset flags C, Z, S, and PV
```

SETFLG
(Normal)

Definition:
Set Flags

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8Dx1	1	0	0	0	1	1	0	1	C	Z	S	PV	0	0	0	1

Operation:

Flags Affected: See below

Clock Cycles = 7

Description:
The SETFLG instruction allows the C, Z, S, or PV flags to be set. If the appropriate bit in the instruction is set to 1, the flag designated is set. If the bit is 0, the designated flag is not set.
The DA and H flags remain unaffected.

Example:

```
SETFLG  Z,S          !Reset flags Z and S

SETFLG  C,Z,S,PV     !Reset flags C, Z, S, and PV
```

Program Control

The Program Control group of Z8000 instructions is used to control flow of execution within a program. Table 2-5 lists the instructions within this group.

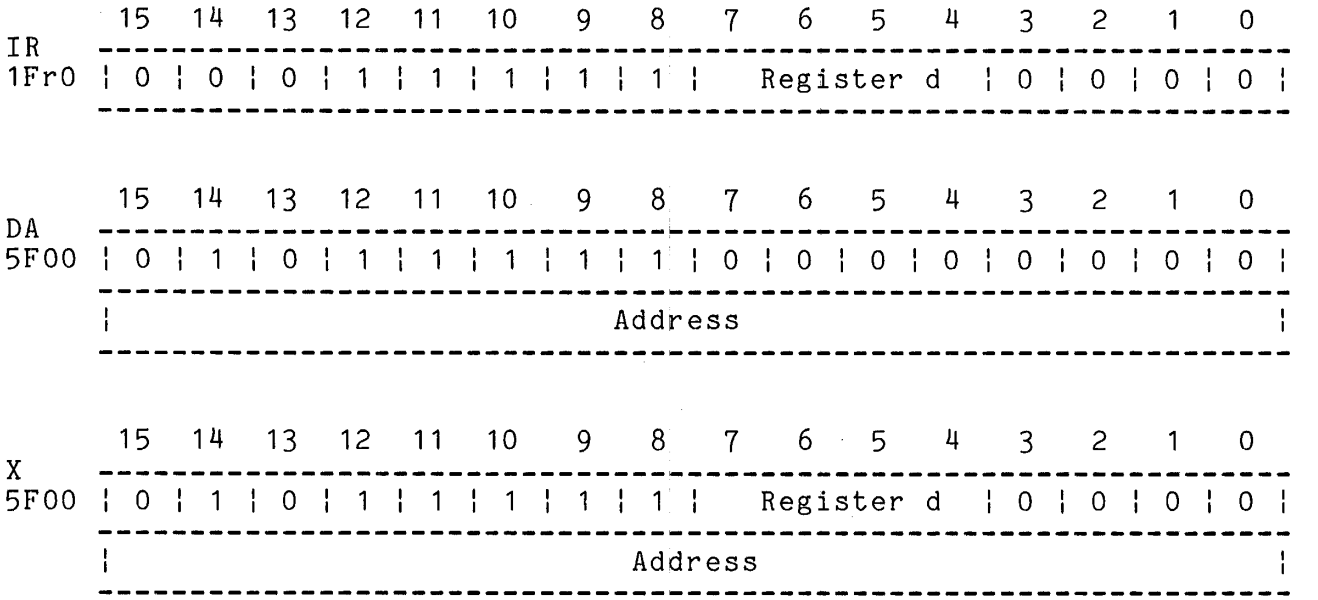
Table 2-5
Program Control Group

Mnemonic	Function
CALL	Call Subroutine
CALR	Call Subroutine Using Program Relative Addressing
DBJNZ	Decrement Byte Register and Jump if Not Zero
DJNZ	Decrement Word Register and Jump if Not Zero
IRET	Return From Interrupt
JP	Jump Conditional
JR	Jump Conditional Relative
RET	Return From Subroutine
SC	System Call

CALL
(Normal)

Definition:
Call Subroutine

Format:



Addressing Modes:
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:

Segmented Z8001

```
R15<0:15> <-- R15<0:15>-2
(RR14<0:22>) <-- Updated PC Offset
R15<0:15> <-- R15<0:15>-2
(RR14<0:22>) <-- PC Segment
PC Segment <-- destination<24:30>
PC Offset <-- destination<0:15>
```

Non-Segmented Z8001/Z8002

```
R15<0:15> <--R15<0:15>-2
(R15<0:15>) <-- Updated PC
PC <-- destination<0:15>
```

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	15,10
DA - NS	12
DA - SSO	18
DA - SLO	20
X - NS	13
X - SSO	18
X - SLO	21

Description:

The CALL instruction is used to transfer program execution to a subroutine. The CALL instruction expects the stack pointer to be R15 (non-segmented) or RR14 (segmented). That is, R15 and RR14 are implied stack pointers.

When the CALL instruction is executed, the Program Counter is pushed onto the stack pointed to R15 or RR14 **minus** 2. The new Program Counter value is then loaded into the Program Counter. The location of the new program counter value depends on the addressing mode used.

The last instruction in the subroutine must always be a RET (Return From Subroutine).

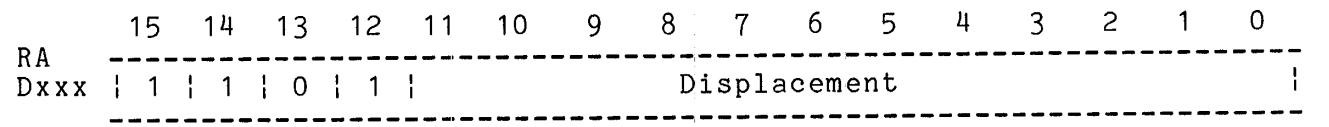
Example:

```
    CALL    R6              !Call the subroutine pointed to by
                            !word register R6.
                            !(Indirect Register Mode)

    CALL    <<%7A>>%13F0    !Call the subroutine located in
                            !segment 7A, location 13F0.
                            !(Direct Address Mode - Long Offset)

    CALL    13F0,(R4)       !Call the subroutine at the address
                            !located by adding 13F0 to the
                            !contents of register R4.
                            !(Indexed Mode - Non-Segmented)
```

CALR
(Normal)

Definition:
Call Subroutine Relative

Format:



Operation:

Segmented Z8001

```
R15<0:15> <-- R15<0:15> -2
(RR14<0:22>) <-- Updated PC Offset
R15<0:15> <-- R15<0:15> -2
(RR14<0:22>) <-- Pc Segment
PC Offset <-- Updated PC Offset
              - 2x displacement
```

Non-Segmented Z8001/Z8002

```
R15<0:15> <-- r15<0:15> -2
(R15<0:15>) <-- Updated PC
PC Offset <-- Updated PC
              - 2x displacement
```

Flags Affected: None

Clock Cycles = 15,10

Description:
The CALR instruction is used to call a subroutine using program relative addressing. When the CALR instruction is executed, the Program Counter is incremented to the next instruction address and then popped onto the stack. The stack pointer is implied within the instruction to be R15 (non-segmented) or RR14 (segmented).

The displacement is a signed 12-bit integer. The new Program Counter value is calculated by shifting the displacement value one bit to the left. In effect, this doubles the value of the displacement. This new displacement value is then subtracted from the old Program Counter value plus 2 to create the new program counter value (PC <-- Updated PC - 2x displacement).

This instruction does not affect the Program Counter segment value when executed on the Z8001 in segmented mode. In other words, the CALR instruction cannot be used to call a subroutine outside the current segment.

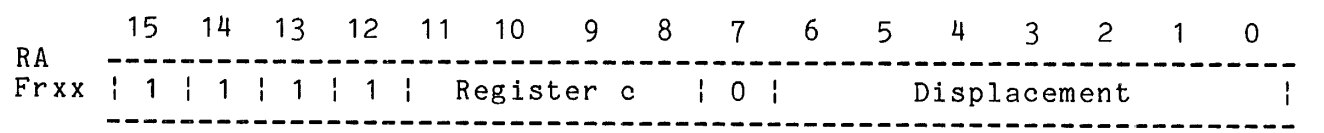
Example:

```
CALR    $-%200      !Jump to the subroutine located at
                    !the current Program Counter value
                    !plus 2, minus 2x 200 (PC+2-400).
```

DBJNZ
(Normal)

Definition:
Decrement Byte Register and Jump if Not Zero

Format:



Operation:
Z8001/Z8002

```
Register c<0:7> <-- Register c<0:7> - 1
If Register c<0:7>≠0
Then PC <-- Updated PC - 2x displacement
Otherwise PC <-- Updated PC
```

Flags Affected: None

Clock Cycles = 11

Description:
The DBJNZ instruction is used to do a qualified jump to another address, using the Relative Addressing mode.

When the DBJNZ instruction is executed, byte Register 'c' is decremented by one. If the contents of Register 'c' do not equal 0, a relative address jump is performed.

When Register 'c' ≠ 0, the Program Counter is incremented by two, the 7-bit unsigned displacement value is shifted left one (its value is doubled), and the new displacement value is subtracted from the updated Program Counter contents to create

a new Program Counter value.
If the byte-register contents do equal 0, the Program Counter is incremented by two, and execution continues.

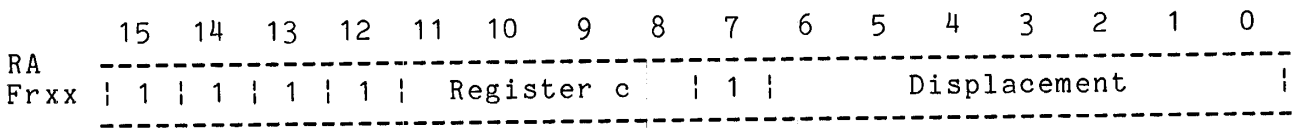
Example:

```
DBJNZ   RH4,%20   !Decrement RH4.  If RH4 >=1 then
                  !increment PC by two, subtract 40
                  !new PC, and jump to new location.
                  !If RH4=0, increment PC by two,
                  !and continue execution.
```

DJNZ
(Normal)

Definition:
Decrement Word Register and Jump if Not Zero

Format:



Operation:
Z8001/Z8002

```
Register c<0:15> <-- Register c<0:15> - 1
If Register c<0:15>≠0
Then PC <-- Updated PC - 2x displacement
Otherwise PC <-- Updated PC
```

Flags Affected: None

Clock Cycles = 11

Description:
The DJNZ instruction is used to do a qualified jump to another address, using the Relative Addressing mode.

When the DJNZ instruction is executed, word Register 'c' is decremented by one. If the contents of Register 'c' do not equal 0, a relative address jump is performed.

When Register 'c' ≠ 0, the Program Counter is incremented by two, the 7-bit unsigned displacement value is shifted left one (its value is doubled), and the new displacement value is subtracted from the updated Program Counter contents to create a new Program Counter value.

If the word-register contents do equal 0, the Program Counter is incremented by two, and execution continues.

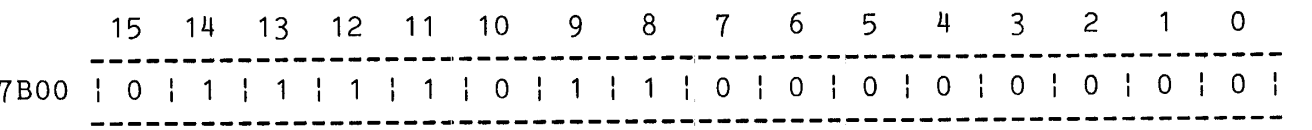
Example:

```
DJNZ    R4,%20      !Decrement R4.  If R4 >=1 then
                    !increment PC by two, subtract 40
                    !from new PC, and jump to new
                    !location.  If R4=0, increment
                    !PC by two, and continue execution.
```

IRET
(System)

Definition:
Return From Interrupt

Format:



Operation:

Segmented Z8001	Non-Segmented Z8001/Z8002
R15<0:15> <-- R15<0:15> + 2	R15<0:15> <-- R15<0:15> +2
FCW <-- (RR14<0:22>)	FCW <-- (R15<0:15>)
R15<0:15> <-- R15<0:15> + 2	R15<0:15> <-- R15<0:15> +2
PC Segment <-- (RR14<0:22>)	PC <-- (R15<0:15>)
R15<0:15> <-- R15<0:15> + 2	R15<0:15> <-- R15<0:15> +2
PC Offset <-- (RR14<0:22>)	
R15<0:15> <-- R15<0:15> + 2	

Flags Affected:
All flags are restored to pre-interrupt status.

Clock Cycles = 16,13

Description:
The IRET instruction is used to return control to a main program following the execution of an interrupt servicing routine.

When the IRET instruction is executed, the old program status information is popped off the system stack in the following order:

1. Identifier Word - discarded.
2. Status Word - returned to FCW.
3. Program Counter Segment - Z8001 only.
4. Program Counter Offset (or PC)

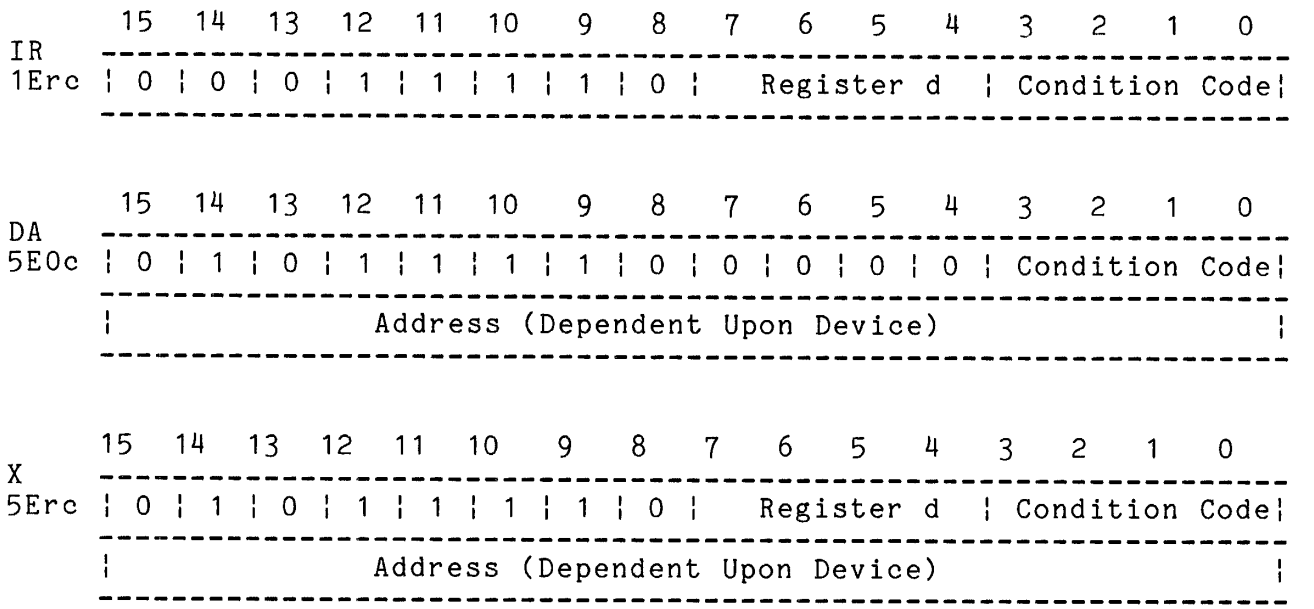
Example:

```
IRET
```

JP
(Normal)

Definition:
Jump Conditional

Format:



Addressing Modes:
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/Z8002

PC<0:15> <-- Register d<0:15>
(if condition is met)
PC<0:15> <-- Updated PC
(if condition is not met)

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock* Cycles
IR - S,NS	15,10/7,7
DA - NS	7/7
DA - SSO	8/8
DA - SLO	10/10
X - NS	8/8
X - SSO	8/8
X - SLO	11/11

*(CC true/CC false)

Description:

The JP instruction allows conditional jumps to other addresses. When the condition code selected within the instruction is true, program execution begins at the location selected by the appropriate addressing mode. If the condition code is not true when tested, program execution continues at the next instruction location.

The Z8001/2 condition codes are listed earlier in this chapter in Table 2-3.

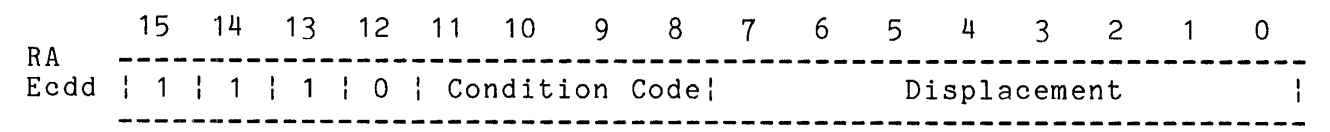
Example:

```
    JP    NZ,R6                 !If the Z flag is not zero (NZ)
                                !jump to the address pointed to
                                !by word register R6.
                                !(Indirect Register mode)

    JP    NZ,<<%7A>>%13F0       !If the Z flag is not zero, jump
                                !to segment 7A, address 13F0.
                                !(Direct Address - Segmented,
                                !Long Offset)

    JP    NZ,<<%7A>>%13F0(R6)   !If the Z flag is not zero, jump
                                !to segment 7A, address 13F0 plus
                                !the contents of R6.
                                !(Indexed - Segmented, Long Offset)
```

JR
(Normal)

Definition:
Jump Conditional Relative

Format:



Operation:
Z8001/Z8002

PC <-- Updated PC + 2x Displacement
if condition met, otherwise
PC <-- Updated PC

Flags Affected: None

Clock Cycles = 6

Description:
The JR instruction causes a relative jump to a new address if the condition code specified is true.

When the condition code is true, the signed 8-bit displacement value is shifted left one position and added to the current Program Counter value plus 2. The displacement value has a range of -128 to +127 words. Keep in mind that the shift left, in effect, doubles the displacement value.

Example:

```
JR      NZ,+40  !If Z is not zero, jump to the
                !current Program Counter value
                !plus 2, plus 80.
```

RET
(Normal)

Definition:
Return Conditional From Subroutine

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
9E0c	1	0	0	1	1	1	1	0	0	0	0	0	Condition Code			

Operation:

Non-Segmented Z8001/Z8002	Segmented Z8001
If CC True	If CC True
PC <-- (R15<0:15>)	PC Segment <-- (RR14<0:22>)
R15<0:15> <-- R15<0:15>+2	R15<0:15> <-- R15<0:15>+2
Else	PC Offset <-- (RR14<0:22>)
PC <-- PC +2	R15<0:15> <-- R15<0:15>+2
	Else
	PC Offset <-- PC Offset +2

Flags Affected: None

Clock Cycles =
CC True/CC False
10,13/7,7

Description:
The RET instruction provides a conditional return from subroutine. If the condition code is met, the Flag and Control Word and Program Counter values are popped off the stack, and program execution is continued where left off before the subroutine call.

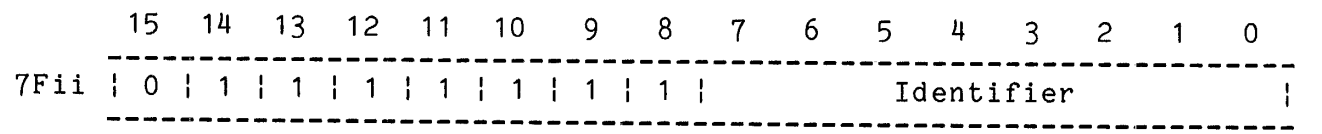
Example:

```
RET     NZ      !Return from subroutine if
                !Z flag is not zero.
```

SC
(Normal)

Definition:
System Call

Format:



Operation:

Non-Segmented Z8001/Z8002

```
R15<0:15> <-- R16<0:15>-2
(R15<0:15>) <-- PC<0:15>+2
R15<0:15> <-- R15<0:15>-2
(R15<0:15>) <-- FCW
R15<0:15> <-- R15<0:15>-2
(R15<0:15>) <-- Identifier
FCW <-- (NPSAP<0:15>+12
PC <-- (NPSAP<0:15>+14
```

Segmented Z8001

```
R15<0:15> <-- R15<0:15>-2
(RR14<0:22>) <-- PC Offset +2
R15<0:15> <-- R15<0:15>-2
(RR14<0:22>) <-- PC Segment
R15<0:15> <-- R15<0:15>-2
(RR14<0:22>) <-- FCW
R15<0:15> <-- R15<0:15>-2
(RR14<0:22>) <-- Identifier
FCW <-- (NPSAP<0:22>+24
PC Segment <-- (NPSAP<0:22>+26
PC Offset <-- (NPSAP<0:22>+ 28
```

Flags Affected:
Specified by the new FCW

Clock Cycles = 39,33

Description:
The SC instruction generates a system call trap (described earlier in this chapter).

When the SC instruction is executed, the current program status is pushed onto the system stack, and new program status is loaded from the location pointed to by the New Program Status Area Pointer (NPSAP). The 8-bit identifier is user definable.

Example:

```
SC      3F      !Generate a system call, using
                !identifier 3F.
```

Load and Exchange

The Load and Exchange group of Z8000 instructions is used to move information between registers and memory. Table 2-6 lists the instructions within this group.

Table 2-6
Load and Exchange Group

Mnemonic	Function
EX	Exchange Words
EXB	Exchange Bytes
LD	Move a Data Word into Memory
LDB	Move a Data Byte into Memory
LDA	Load Address
LDAR	Load Program Relative Address
LDD	Move Word and Decrement
LDDB	Move Byte and Decrement
LDDR	Move Word, Decrement, and Repeat
LDDRB	Move Byte, Decrement, and Repeat
LDI	Move Word and Increment
LDIB	Move Byte and Increment
LDIR	Move Word, Increment, and Repeat
LDIRB	Move Byte, Increment, and Repeat
LDK	Load Constant
LDL	Load Long Word
LDM	Load Multiple Words
LDPS	Load Program Status and Jump
LDR	Load Word Using Program Relative Addressing
LDRB	Load Byte Using Program Relative Addressing
LDRL	Load Long Word Using Program Relative Addressing

EX
(Normal)

Definition:
Exchange Source Word with Destination Word

Format:

R
Adsd

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	Register s				Register d			

IR
2Dsd

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	1	Register s				Register d			

DA
6D0d

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	1	0	0	0	0	Register d			
Address															

X
6Dsd

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	1	Register s				Register d			
Address															

Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SSL
X - NS,SSO,SSL

Operation:
Z8001/2

Register s<0:15> <--> Register d<0:15>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	6
IR - S,NS	12
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The EX instruction is used to exchange the contents of two word registers or the contents of a word register and a memory location.

Example:

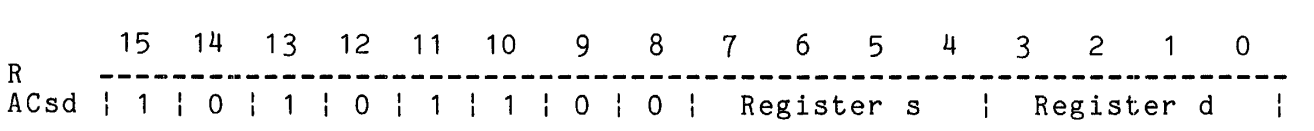
```
    EX      R4,R8           !Exchange the contents of
                            !registers R4 and R8.
                            !(Register Addressing mode)

    EX      R4,<<7B>>2A     !Exchange the contents of
                            !register R4 with the
                            !information contained in
                            !segment 7B, location 002A.
                            !(DA Addressing Mode - Short
                            !Offset)
```

EXB
(Normal)

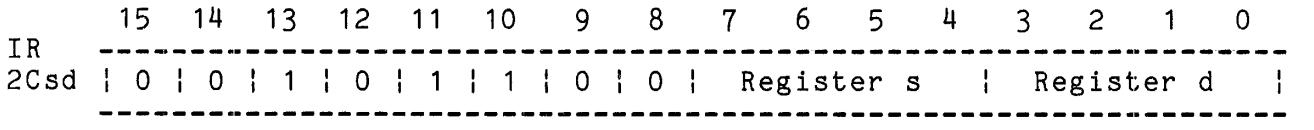
Definition:
Exchange Source Byte with Desination Byte

Format:

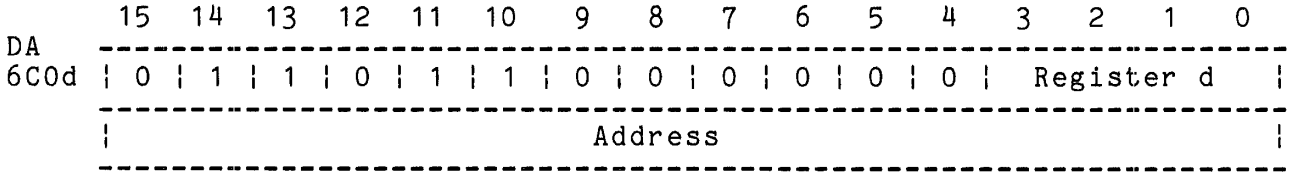
R



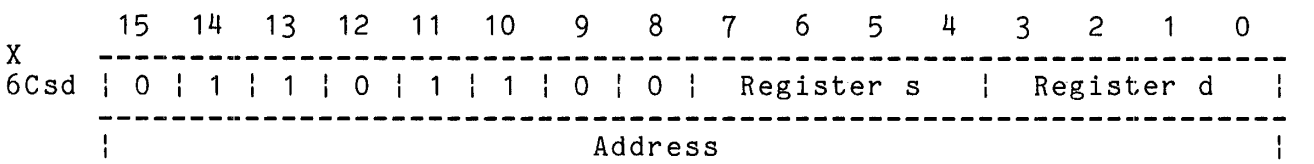
IR



DA



X



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SSL
X - NS,SSO,SSL

Operation:
Z8001/2

Register s<0:7> <--> Register d<0:7>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	6
IR - S,NS	12
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The EX instruction is used to exchange the contents of two byte registers or the contents of a byte register and a memory location.

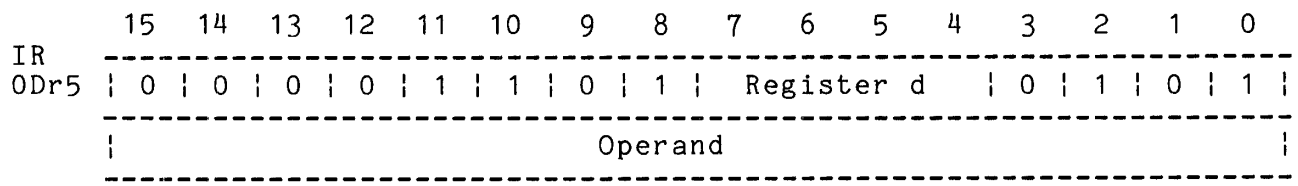
Example:

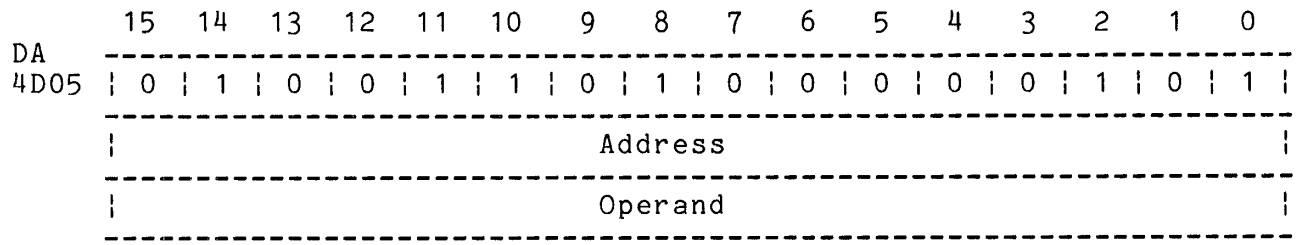
```
    EXB     RH4,RL5          !Exchange the contents of
                             !registers RH4 and RL5.
                             !(Register Addressing mode)

    EXB     RH4,<<7B>>2A     !Exchange the contents of
                             !register RH4 with the
                             !information contained in
                             !segment 7B, location 002A.
                             !(DA Addressing Mode - Short
                             !Offset)
```

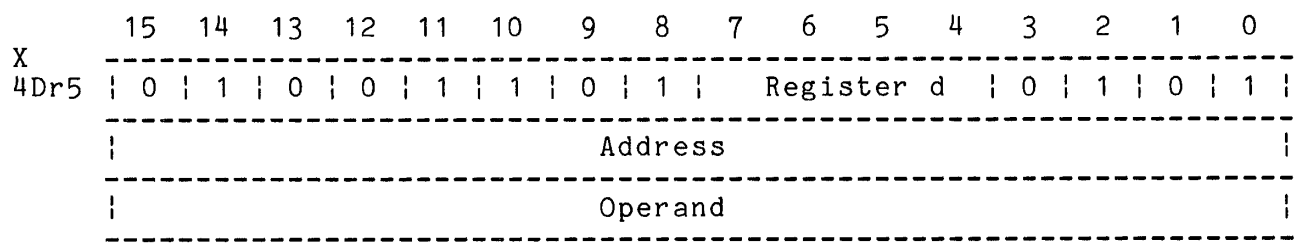
LD
(Normal)

Definition:
Move a Data Word Into/From Memory Immediate

Format:







Addressing Modes:
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Register d<0:15> <-- Operand<0:15>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	11
DA - NS	14
DA - SSO	15
DA - SLO	17
X - NS	15
X - SSO	15
X - SLO	18

Description:
The operand immediately following the LD instruction is loaded into a word register or memory, depending upon the addressing mode used. Note that in the Indexed mode (X), register R0 cannot be used as the index register.

Example:

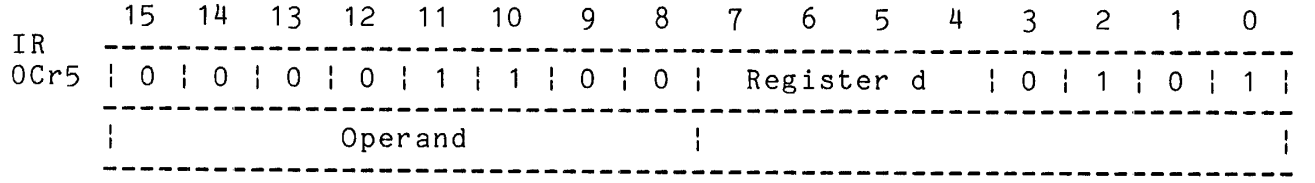
```
LD      @R4,#FFFF       !Load the value FFFF into
                        !the memory location pointed
                        !to by register R4.
                        !(IR mode)

LD      %3000(R2),R6    !Load the value contained
                        !in R6 into memory location
                        !3000 plus the value
                        !stored in register R2.
                        !(X Mode)

LD      R6,%6000        !Load R6 from memory address 6000.
```

LDB
(Normal)

Definition:
Move a Data Byte Into/From Memory Immediate

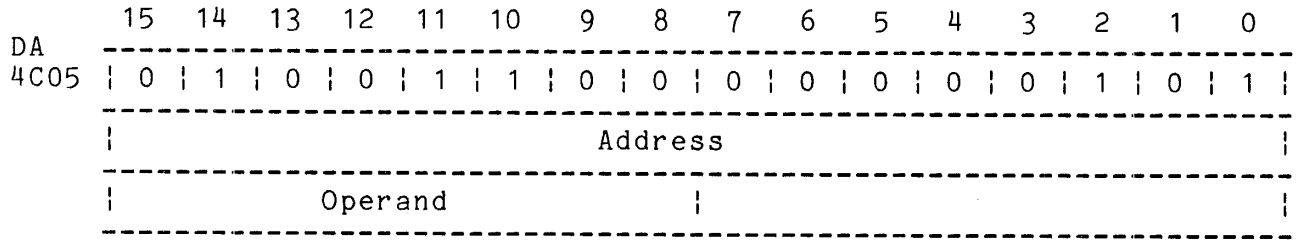
Format:

IR



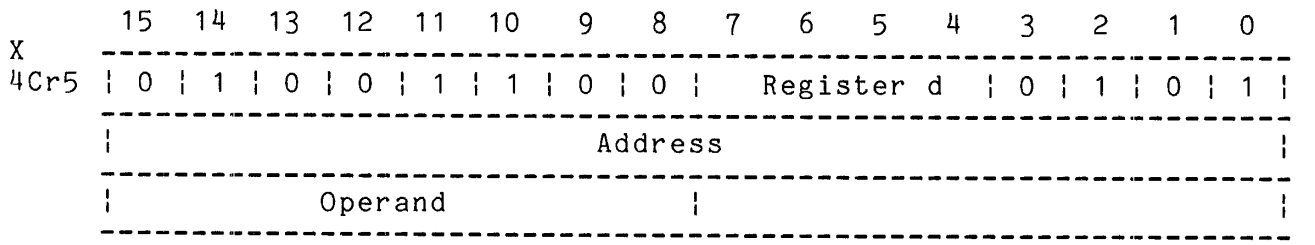
OCr5

DA



4C05

X



4Cr5

Addressing Modes:
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Register d<0:7> <-- Operand<0:7>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	11
DA - NS	14
DA - SSO	15
DA - SLO	17
X - NS	15
X - SSO	15
X - SLO	18

Description:
The operand immediately following the LDB instruction is loaded into a byte register or memory, depending upon the addressing mode used. Note that in the Indexed mode (X), register RH0 cannot be used as the index register.

Example:

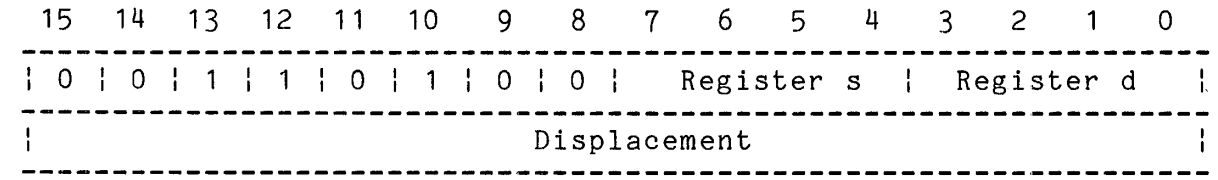
```
    LDB     @RL4,#FF          !Load the value FF into the
                              !memory location pointed to
                              !by register RL4.
                              !(IR mode)

    LDB     %3000(RH2),RL6    !Load the value contained in
                              !RL6 into memory location
                              !(3000 plus RH2)
                              !(X Mode)

    LDB     RH6,%6000         !Load RH6 from memory address 6000.
```

LDA
(Normal)

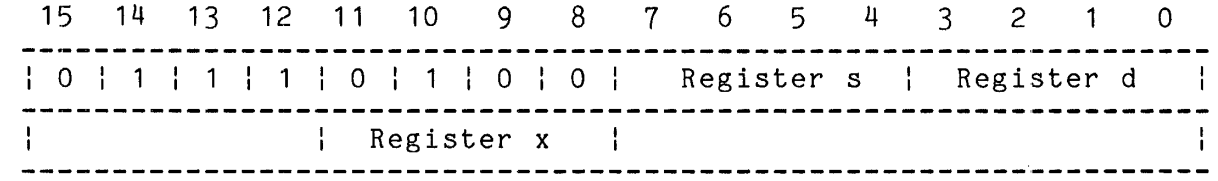
Definition:
Load Address

Format:

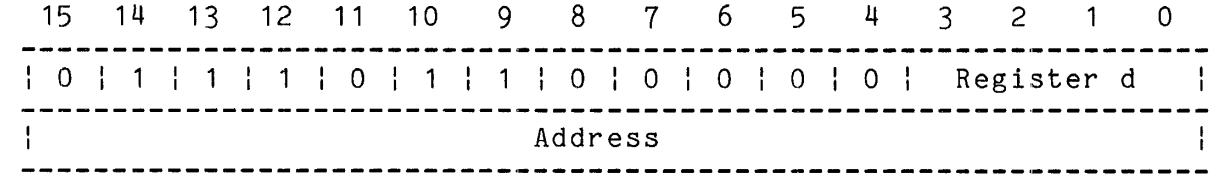
BA 34rr



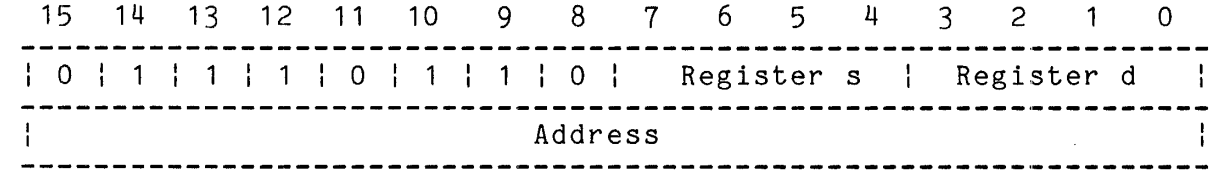
BX 34rr



DA 760r



X 76rr



Addressing Modes:
BA - S,NS
BX - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
BA - S,NS	15
BX - S,NS	15
DA - NS	12
DA - SSO	13
DA - SLO	15
X - NS	13
X - SSO	13
X - SLO	16

Description:
The LDA instruction loads a memory address, as opposed to the contents of a memory location, into a specified register. The actual address loaded is determined by the addressing mode selected.

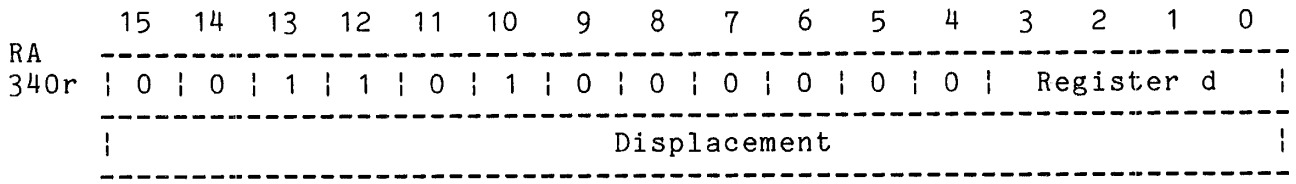
Example:

```
    LDA     R2,RR4(R8)      !Load into register R2 the calculated
                            !address by adding the contents of
                            !R8 to the long offset value of the
                            !Program Counter stored in RR4.
                            !(BX mode)
```

LDAR
(Normal)

Definition:
Load Address Into Register Relative

Format:



Operation:
See Description.

Flags Affected: None

Clock Cycles = 15

Description:
The LDAR instruction loads an address, as opposed to the contents of a memory location, into the designated register. The LDAR instruction uses the Relative Addressing mode. Therefore, any carry generated will be ignored. The segment value remains unchanged.

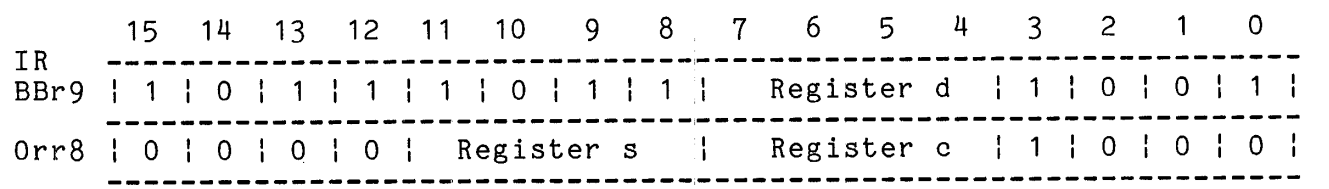
Example:

```
LDAR    R2,$-%02FA      !Load into register R2 the
                        !calculated address by subtracting
                        !02FA from the current Program
                        !Counter value. (RA mode)
```

LDD
(Normal)

Definition:
Move Word From Source Location to Destination Location
and Decrement Both Addresses

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> -2
Register d<0:15> <-- Register d<0:15> -2
Register c<0:15> <-- Register c<0:15> -1

Flags Affected:
P/V set to 1 if Register c =0.
P/V reset otherwise.

Clock Cycles = 20

Description:
The LDD instruction moves the contents of the memory location pointed to by Register 's' to the location pointed to by Register 'd'. The contents of Register 'c' are decremented by one. The LDDR instruction repeats this action until Register 'c' = 0.

Example:

```
    LDD     @R4,@R8,R12     !Move the contents of the location
                            !pointed to by R12 into the location
                            !pointed to by R8.  If R4 = 0, set
                            !P/V flag, else reset P/V flag.
```

LDDB
(Normal)

Definition:
Move Byte From Source Location to Destination Location and Decrement Both Addresses

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR																
Bar9	1	0	1	1	1	0	1	0	Register d				1	0	0	1
Orr8	0	0	0	0	Register s				Register c				1	0	0	0

Operation:
Z8001/2

```
Register d<0:7> <-- Register s<0:7>
Register s<0:7> <-- Register s<0:7> -1
Register d<0:7> <-- Register d<0:7> -1
Register c<0:7> <-- Register c<0:7> -1
```

Flags Affected:
P/V set to 1 if Register c =0.
P/V reset otherwise.

Clock Cycles = 20

Description:
The LDDB instruction moves the contents of the memory location pointed to by Register 's' to the location pointed to by Register 'd'. The contents of Register 'c' are decremented by one.

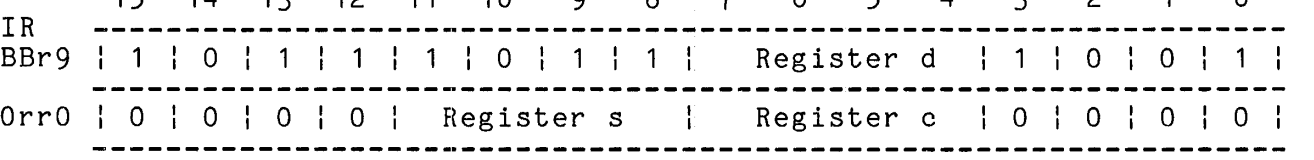
Example:

```
    LDDB    @R4,@R8,R12     !Move the contents of the location
                            !pointed to by R12 into the location
                            !pointed to by R8.  If R4 = 0, set
                            !P/V flag, else reset P/V flag.
```

LDDR
(Normal)

Definition:
Move Word From Source Location to Destination Location, Decrement Both Addresses, and Repeat

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> -2
Register d<0:15> <-- Register d<0:15> -2
Register c<0:15> <-- Register c<0:15> -1
Repeat until Register c = 0

Flags Affected:
P/V set to 1 when Register c =0.

Clock Cycles = 11 + 9n*
* n = number of iterations

Description:
The LDDR instruction moves the contents of the memory location pointed to by Register 's' to the location pointed to by Register 'd'. The contents of Register 'c' are decremented by one. The LDDR instruction repeats this action until Register 'c' = 0.

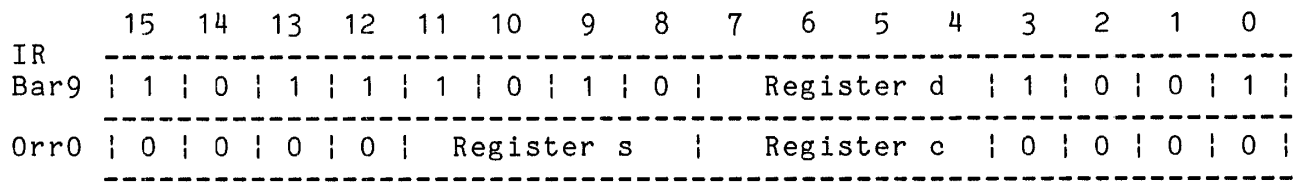
Example:

```
    LDDR    @R4,@R8,R12     !Move the contents of the location
                            !pointed to by R12 into the location
                            !pointed to by R8.  Repeat until
                            !R4 = 0.
```

LDDRB
(Normal)

Definition:
Move Byte From Source Location to Destination Location, Decrement Both Addresses, and Repeat

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:7> <-- Register s<0:7> -1
Register d<0:7> <-- Register d<0:7> -1
Register c<0:7> <-- Register c<0:7> -1
Repeat until Register c = 0

Flags Affected:
P/V set to 1 if Register c =0.

Clock Cycles = 11 + 9n*
* n = number of iterations

Description:
The LDDRB instruction moves the contents of the memory location pointed to by Register 's' to the location pointed to by Register 'd'. The contents of Register 'c' are decremented by one. The LDDRB instruction repeats this action until Register 'c' = 0.

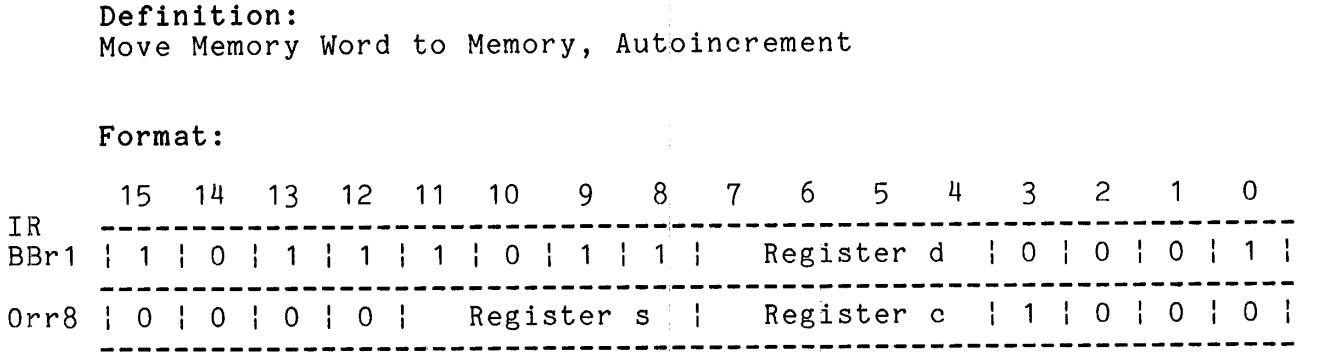
Example:

```
    LDDRB   @R4,@R8,R12     !Move the contents of the location
                            !pointed to by R12 into the location
                            !pointed to by R8.  Repeat until
                            !R4 = 0.
```

LDI
(Normal)

Definition:
Move Memory Word to Memory, Autoincrement

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> +2
Register d<0:15> <-- Register d<0:15> +2
Register c<0:15> <-- Register c<0:15> -1
```

Flags Affected:
P/V set if Register c = 0
P/V reset otherwise

Clock Cycles = 20

Description:
The LDI instruction moves the contents of the memory location pointed to by Register 's' into the location pointed to by Register 'd'. Register 'c' is decremented by 1, and Registers 'd' and 's' are incremented by 2. If Register 'c' = 0, the P/V flag is set. If Register 'c' ≠ 0, the P/V flag is reset.

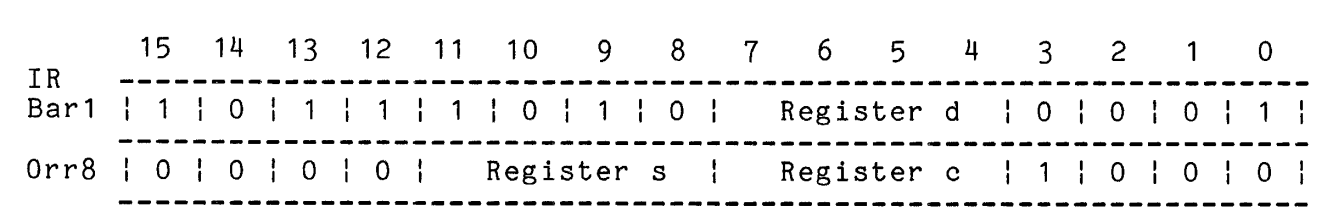
Example:

```
    LDI     @RR2,@RR6,R10   !Move the contents of
                            !memory location RR6
                            !to memory location RR2.
                            !Increment RR2 and RR6 by 2.
                            !Decrement R10 by 1
```

LDIB
(Normal)

Definition:
Move Memory Byte to Memory, Autoincrement

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> +1
Register d<0:15> <-- Register d<0:15> +1
Register c<0:15> <-- Register c<0:15> -1

Flags Affected:
P/V set if Register c = 0
P/V reset otherwise

Clock Cycles = 20

Description
The LDIB instruction moves the byte contents of the memory location pointed to by Register 's' into the location pointed to by Register 'd'. Register 'c' is decremented by 1, and Registers 'd' and 's' are incremented by 1. If Register 'c' = 0, the P/V flag is set. If Register 'c' ≠ 0, the P/V flag is reset.

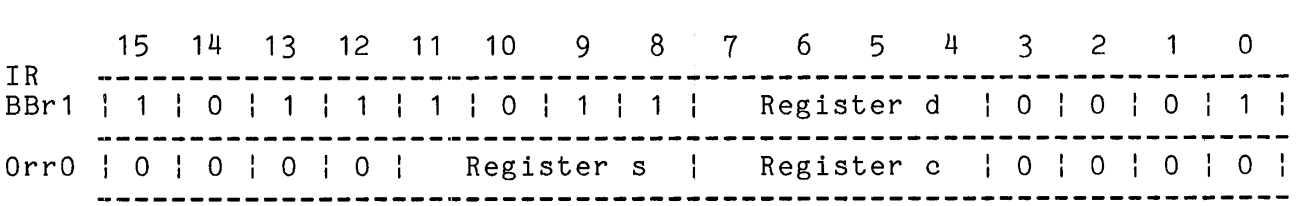
Example:

```
    LDIB    @R2,@R6,R10    !Move the contents of memory
                           !location R6 to memory
                           !location R2.  Increment
                           !R2 and R6 by 1.  Decrement
                           !R10 by 1.
```

LDIR
(Normal)

Definition:
Move Memory Word to Memory, Autoincrement and Repeat

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> +2
Register d<0:15> <-- Register d<0:15> +2
Register c<0:15> <-- Register c<0:15> -1
Repeat until Register c = 0

Flags Affected:
P/V set when Register c = 0

Clock Cycles = 11 + 9*
* n = number of iterations

Description
The LDIR instruction moves the word contents of the memory location pointed to by Register 's' into the location pointed to by Register 'd'. Register 'c' is decremented by 1, and Registers 'd' and 's' are incremented by 2. This action repeats until Register c = 0.

Interrupts may be serviced while this instruction is executing.

Example:

```
    LDIR    @RR2,@RR6,R10   !Move the contents of memory
                            !location RR6 to memory
                            !location RR2.  Increment RR2
                            !and RR6 by 2.  Decrement R10
                            !by 1.  Repeat until R10 = 0.
```

LDIRB
(Normal)

Definition:
Move Memory Byte to Memory, Autoincrement, Repeat

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR																
Bar1	1	0	1	1	1	0	1	0	Register d				0	0	0	1
Orr0	0	0	0	0	Register s				Register c				0	0	0	0

Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> +1
Register d<0:15> <-- Register d<0:15> +1
Register c<0:15> <-- Register c<0:15> -1
Repeat until Register c = 0

Flags Affected:
P/V set when Register c = 0

Clock Cycles = 11 + 9*
* n = number of iterations

Description
The LDIR instruction moves the byte contents of the memory location pointed to by Register 's' into the location pointed to by Register 'd'. Register 'c' is decremented by 1, and Registers 'd' and 's' are incremented by 1. This action repeats until Register c = 0.

Interrupts may be serviced while this instruction is executing.

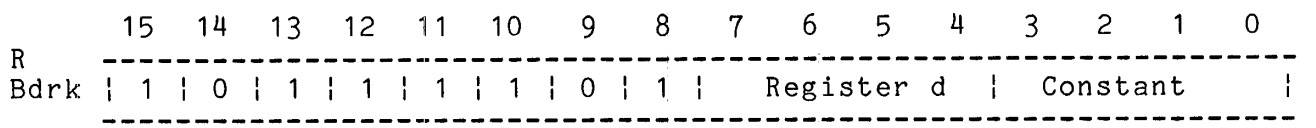
Example:

```
    LDIRB   @RR2,@RR6,R10   !Move the contents of memory
                            !location RR6 to memory
                            !location RR2.  Increment
                            !RR2 and RR6 by 1.  Decrement R10
                            !by 1.  Repeat until R10 = 0.
```

LDK
(Normal)

Definition:
Load Constant into a Register

Format:



Operation:
Z8001/2

```
-------
Register d<0:3> <-- Constant<0:3>
Register d<4:15> <-- 0
```

Flags Affected: None

Clock Cycles = 5

Description:
The LDK instruction is used to immediately load a 4-bit nybble into the least significant position of a general purpose word register. When the nybble is loaded, the remaining bits of the register are cleared.

Example:

```
    LDK     R4,#%8          !Load the lower four bits of R4
                            !with the value 0008, and fill
                            !the remaining bits with 0.
```

LDL
(Normal)

Definition:
Load Long (32-bit) Word into Word Register

Format:

R
94rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0	Register s				Register d			

IM
140r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	0	0	0	Register d			
Operand (High-Order Word)															
Operand (Low-Order Word															

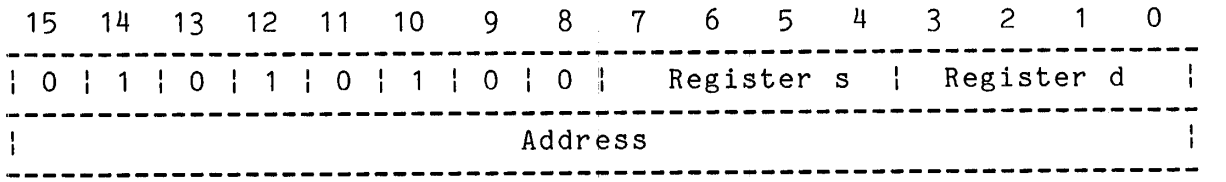
IR
14rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	Register s				Register d			

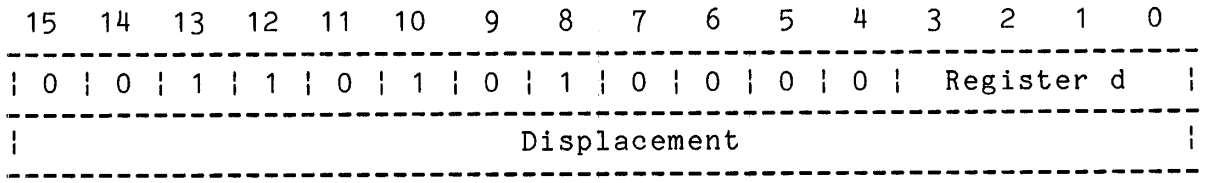
DA
540r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	0	0	0	0	Register d			
Address															

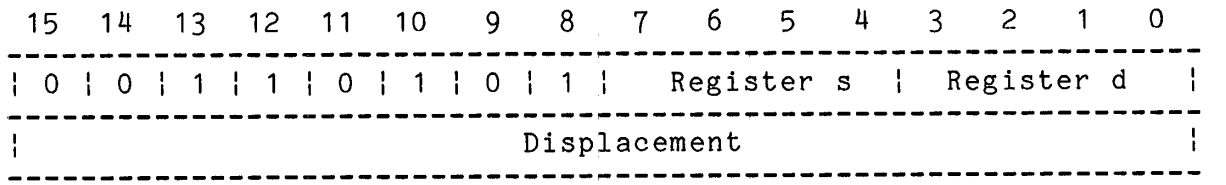
X
54rr



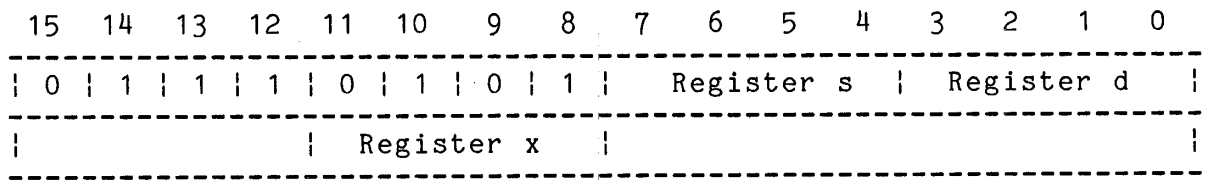
RA
350r



BA
35rr



BX
75rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO
RA - S,NS
BA - S,NS
BX - S,NS

Operation:
Z8001/2

Register d<0:31> <-- Register s<0:31>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	5
IM - S,NS	11
IR - S,NS	11
DA - NS	12
DA - SSO	13
DA - SLO	15
X - NS	13
X - SSO	13
X - SLO	16
RA - S,NS	17
BA - S,NS	17
BX - S,NS	17

Description:

The LDL instruction moves a long word into one of the 32-bit long word registers designated as Register 'd'. The source of the long word can be an immediate value, another register, or a memory location, depending on the addressing mode used.

Example:

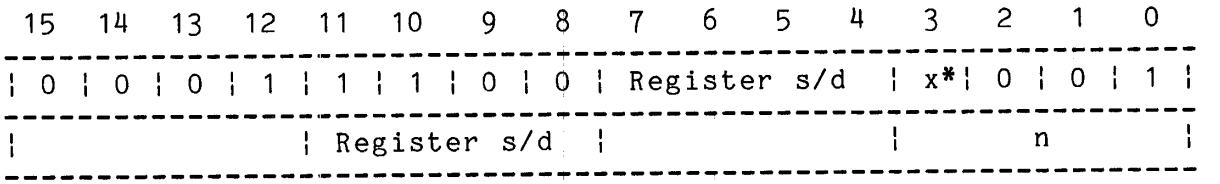
```
    LDL     RR4,3BEE0277    !Move the value 3BEE0277 to
                            !long word register RR4.
                            !(Immediate mode)
```

LDM
(Normal)

Definition:
Load Multiple Register Words to/from Memory

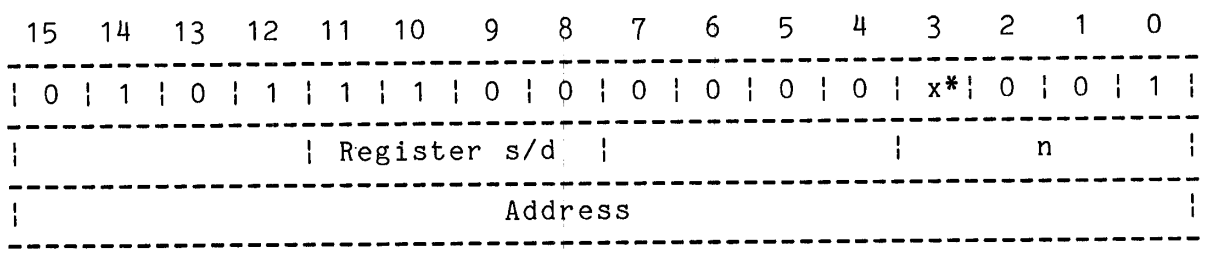
Format:

IR 1Crx



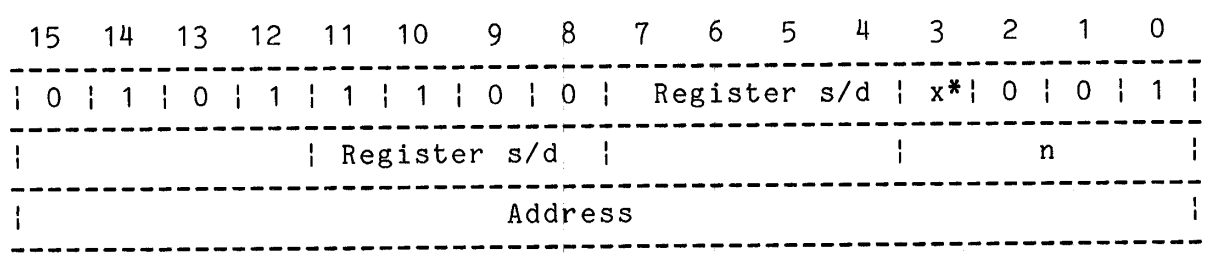
* If bit 3=1, bits 4-7 = destination register
* If bit 3=0, bits 4-7 = source register

DA 5C0x



* If bit 3=1, bits 8-11 of second word = source register
* If bit 3=0, bits 8-11 of second word = destination register

X 5C0x



* If bit 3=1, bits 8-11 of second word = source register
* If bit 3=0, bits 8-11 of second word = destination register

Addressing Modes:
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock * Cycles
IR - S,NS	11 + 3n
DA - NS	14 + 3n
DA - SSO	15 + 3n
DA - SLO	17 + 3n
X - NS	15 + 3n
X - SSO	15 + 3n
X - SLO	18 + 3n

* n = number of registers

Description:
The LDM instruction is used to either move the contents of "n" 16-bit registers to memory or to load "n" 16-bit registers from memory.

When the LDM instruction begins execution, the first register specified in the instruction as Register 's' is moved to the destination, followed in ascending order by the remaining registers until "n" = 0. Movement can be from register to memory or from memory to register. In either direction, memory is accessed in ascending order. The number of registers to be moved or loaded is specified in the 'n' field of the instruction. The LDM instruction wraps around the register set. That is, if 8 registers are specified in the 'n' field, and the first register to be moved or loaded is specified as R12, the instruction will move the following registers in the following order:

R12, R13, R14, R15, R0, R1, R2, R3

Note that if 'n' = 0, one register is moved or loaded. If 'n' =1, two registers are moved or loaded, etc.
Note that bit 3 of the first instruction word indicates the destination or source:

- If bit 3 = 0, data is moved from memory to register, and the first destination register is always specified in bits 8-11 of the second instruction word.
- If bit 3 = 1, data is moved from register to memory, and the first source register is always specified in bits 8-11 of the second instruction word.

The LDM instruction cannot be interrupted.

Example:

```
LDM     30FF,R4,#4      !Starting with register R4,
                        !move 4 registers to memory,
                        !beginning at location 30FF.
                        !(Non-segmented, DA mode)

LDM     R4,30FF,#4      !Starting with register R4,
                        !load 4 registers from memory,
                        !beginning at location 30FF.
                        !(Non-segmented, DA mode)
```

LDPS
(System)

Definition:
Load Program Status (FCW)

Format:

IR 30r0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1	Register s				0	0	0	0

DA 7900

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0
Address															

X 79r0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	1	Register s				0	0	0	0
Address															

Addressing Modes:
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected: All
See Description

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	12,16
DA - NS	16
DA - SSO	20
DA - SLO	22
X - NS	17
X - SSO	20
X - SLO	23

Description:
The LDPS instruction is used to load processor status information from consecutive memory locations, accessed in ascending order. The starting address of the status information is determined by the addressing mode used. Once the new status information is loaded, execution begins at the new Program Counter location.

Z8001 status is contained in four consecutive memory words, and Z8002 status is contained in two consecutive memory words. **Note** that the Program Counter segment value is not affected in non-segmented operation.

Example:

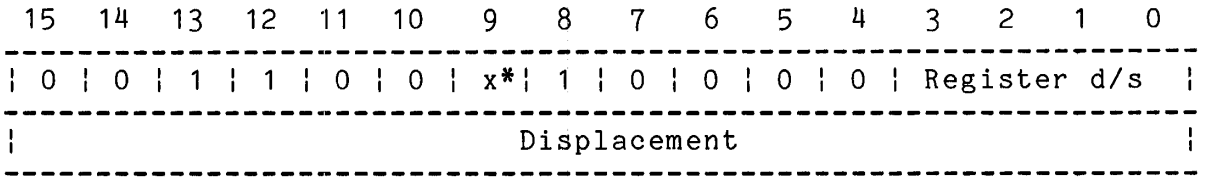
```
LDPS    R12             !Load processor status
                        !information starting
                        !at the address pointed to
                        !by register R12.
                        !(Indirect Register mode)

LDPS    <<2B>>%30FF     !Load processor status
                        !information starting
                        !at segment 2B, location 30FF.
                        !(DA mode, segmented, long offset)
```

LDR
(Normal)

Definition:
Load Word Using Relative Memory Addressing

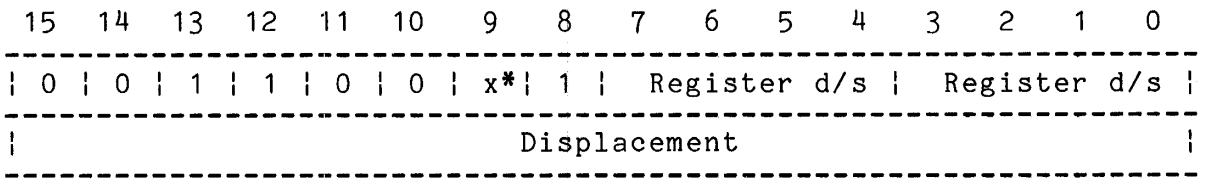
Format:

RA 3x0r



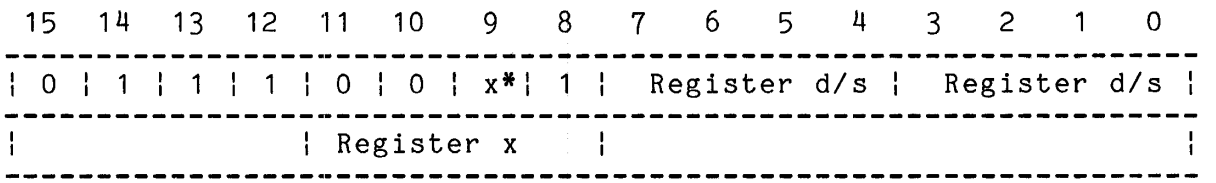
* If bit 9=0, bits 0-3 are Register d
If bit 9=1, bits 0-3 are Register s

BA 3xrr



* If bit 9=0, bits 0-3 = Register d, and bits 4-7 = Register s
If bit 9=1, bits 0-3 = Register s, and bits 4-7 = Register d

BX 7xrr



* If bit 9=0, bits 0-3 = Register d, and bits 4-7 = Register s
If bit 9=1, bits 0-3 = Register s, and bits 4-7 = Register d

Addressing Modes:
RA - S,NS
BA - S,NS
BX - S,NS

Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15> +/- Displacement

Flags Affected:
None

Clock Cycles = 14
(for all modes)

Description:
The LDR instruction loads a register from memory or moves register contents to memory, using program relative memory addressing.

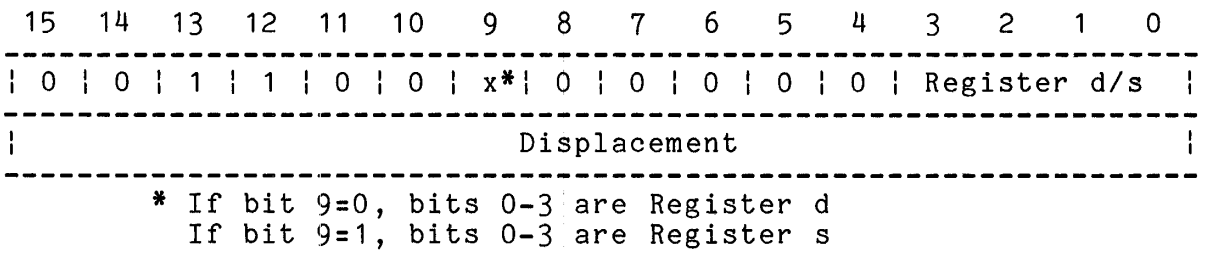
Example:

```
    LDR     R2,$+%100       !Load register R2 from memory
                            !location (PC + 100).

    LDR     $+%100,R2       !Move the contents of R2
                            !to memory location (PC +100)
```

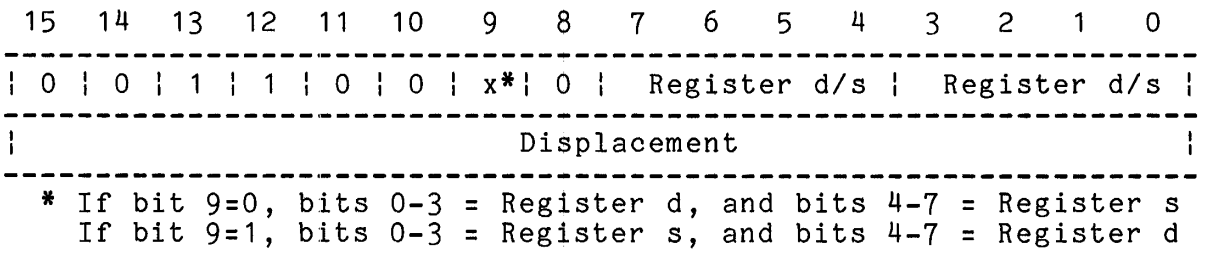
LDRB
(Normal)

Definition:
Load Byte Using Relative Memory Addressing

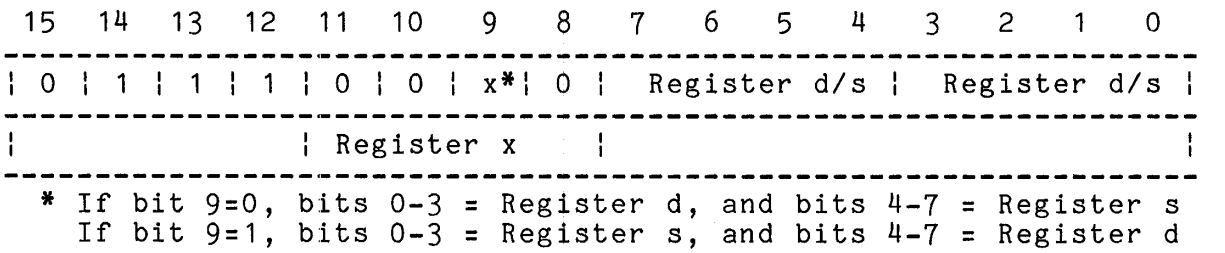
Format:



* If bit 9=0, bits 0-3 are Register d
If bit 9=1, bits 0-3 are Register s



* If bit 9=0, bits 0-3 = Register d, and bits 4-7 = Register s
If bit 9=1, bits 0-3 = Register s, and bits 4-7 = Register d



* If bit 9=0, bits 0-3 = Register d, and bits 4-7 = Register s
If bit 9=1, bits 0-3 = Register s, and bits 4-7 = Register d

Addressing Modes:
RA - S,NS
BA - S,NS
BX - S,NS

Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15> +/- Displacement

Flags Affected: None

Clock Cycles = 14
(for all modes)

Description:
The LDRB instruction loads a byte register from memory or moves byte register contents to memory, using program relative memory addressing.

Example:

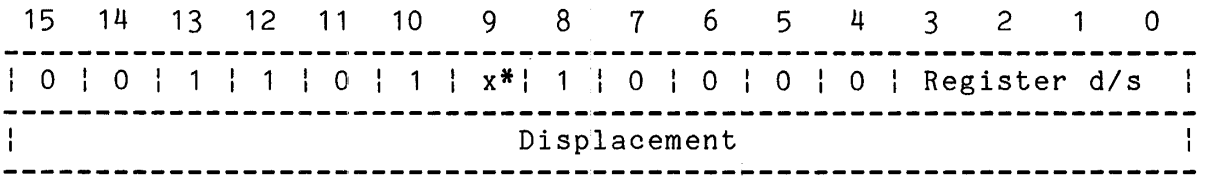
```
    LDRB    R2,$+%100       !Load register R2 from memory
                            !location (PC + 100).

    LDRB    $+%100,R2       !Move the contents of R2
                            !to memory location (PC +100)
```

LDRL
(Normal)

Definition:
Load Long Word Using Relative Memory Addressing

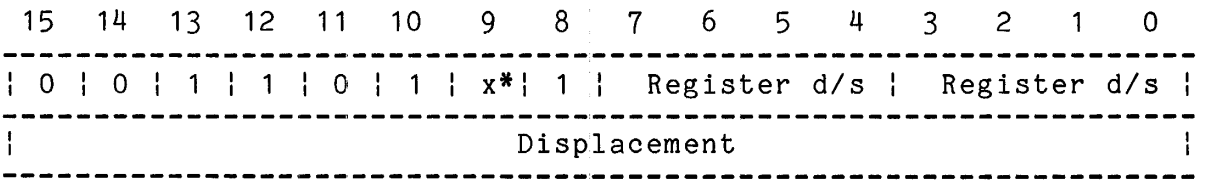
Format:

RA
3x0r



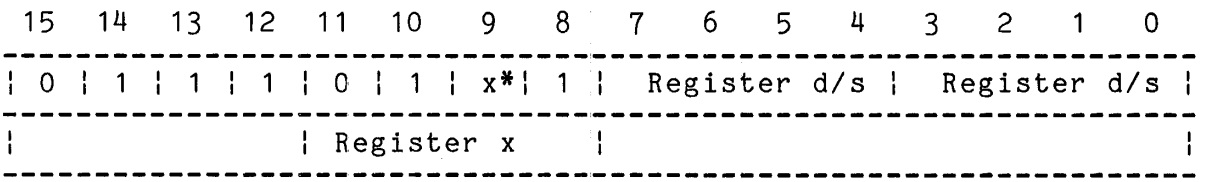
* If bit 9=0, bits 0-3 are Register d
If bit 9=1, bits 0-3 are Register s

BA
3xrr



* If bit 9=0, bits 0-3 = Register d, and bits 4-7 = Register s
If bit 9=1, bits 0-3 = Register s, and bits 4-7 = Register d

BX
7xrr



* If bit 9=0, bits 0-3 = Register d, and bits 4-7 = Register s
If bit 9=1, bits 0-3 = Register s, and bits 4-7 = Register d

Addressing Modes:
RA - S,NS
BA - S,NS
BX - S,NS

Operation:
Z8001/2

Register d<0:31> <-- Register s<0:31> +/- Displacement

Flags Affected: None

Clock Cycles = 17
(for all modes)

Description:
The LDRL instruction loads a long word register from memory or moves long word register contents to memory, using program relative memory addressing.

Example:

```
    LDRL    RR2,$+%100      !Load register RR2 from memory
                            !location (PC + 100).

    LDRB    $+%100,RR2      !Move the contents of RR2
                            !to memory location (PC +100)
```

The Arithmetic Group

The arithmetic group is made up of the following instructions:

Table 2-7
The Arithmetic Group

Mnemonic	Function
ADC	Add Register Words With Carry
ADCB	Add Register Bytes With Carry
ADD	Add Words
ADDB	Add Bytes
ADDL	Add Long Words
CLR	Clear a Word
CLRB	Clear a Byte
CLRL	Clear a Long Word
COM	Complement a Word
COMB	Complement a Byte
DAB	Decimal Adjust
DEC	Decrement a Word
DECB	Decrement a Byte
DIV	Divide Words
DIVL	Divide Long Words
EXTS	Extend Word Sign Bit
EXTSB	Extend Byte Sign Bit
EXTSL	Extend Long Word Sign Bit
INC	Increment Word
INCB	Increment Byte
MULT	Multiply Words
MULTL	Multiply Long Words
NEG	Two's Complement a Word
NEGB	Two's Complement a Byte
RES	Reset a Bit in a Word
RESB	Reset a Bit in a Byte
RL	Rotate Word Left and into Carry
RLB	Rotate Byte Left and into Carry
RLC	Rotate Word Left Through Carry
RLCB	Rotate Byte Left Through Carry
RLDB	Rotate Left BCD

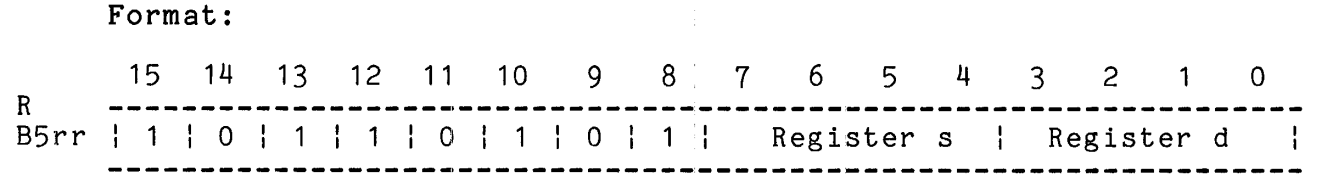
Table 2-7 (Cont.)
The Arithmetic Group

Mnemonic	Function
RR	Rotate Word Right and to Carry
RRB	Rotate Byte Right and to Carry
RRC	Rotate Word Right Through Carry
RRCB	Rotate Byte Right Through Carry
RRDB	Rotate Right BCD
SBC	Subtract Register Words with Borrow
SBCB	Subtract Register Bytes with Borrow
SUB	Subtract Word from Register
SUBB	Subtract Byte from Register
SUBL	Subtact Long Word from Register
SDA	Arithmetic Shift Word
SDAB	Arithmetic Shift Byte
SDAL	Arithmetic Shift Long Word
SET	Set a Bit in a Word
SETB	Set a Bit in a Byte
SLA	Arithmetic Left Shift Word
SLAB	Arithmetic Left Shift Byte
SLAL	Arithmetic Left Shift Long Word
SRA	Arithmetic Right Shift Word
SRAB	Arithmetic Right Shift Byte
SRAL	Arithmetic Right Shift Long Word

ADC
(Normal)

Definition:
Add Register Words With Carry

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15> + Register d<0:15> + C
```

Flags Affected:
C set to 1 if carry, reset otherwise
Z set to 1 if result = 0, reset otherwise
S set to 1 if result is negative, reset otherwise
P/V set to 1 on arithmetic overflow, reset otherwise

Clock Cycles = 5

Description:
The ADC instruction adds the contents of Register 's', Register 'd', and the Carry flag, and places the sum into Register 'd'. If there is a carry from the most significant position of the word, the Carry (C) flag is set.

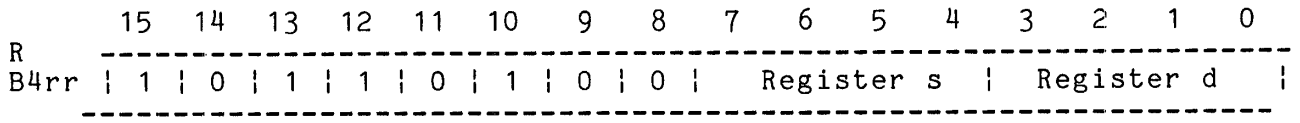
Example:

```
    ADC     R4,R6   !Add the contents of R4, R6, and the
                    !carry flag.  Place the sum in R4 and
                    !set or reset the carry flag accordingly.
```

ADCB
(System)

Definition:
Add Register Bytes with Carry

Format:



Operation:
Z8001/2

```
Register d<0:7> <-- Register s<0:7> + Register d<0:7> + C
```

Flags Affected:
C set to 1 if carry, reset otherwise
Z set to 1 if result = 0, reset otherwise
S set to 1 if result is negative, reset otherwise
P/V set to 1 on arithmetic overflow, reset otherwise

Clock Cycles = 5

Description:
The ADCB instruction adds the contents of byte Register 's', byte Register 'd', and the Carry flag, and places the sum into Register 'd'. If there is a carry from the most significant position of the word, the Carry (C) flag is set.

Example:

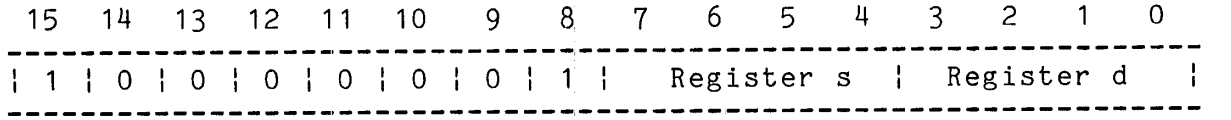
```
ADCB    RL4,RH6    !Add the contents of RL4, RH6, and
                   !the carry flag.  Place the sum in
                   !RL4 and set or reset the carry flag
                   !accordingly.
```

ADD
(Normal)

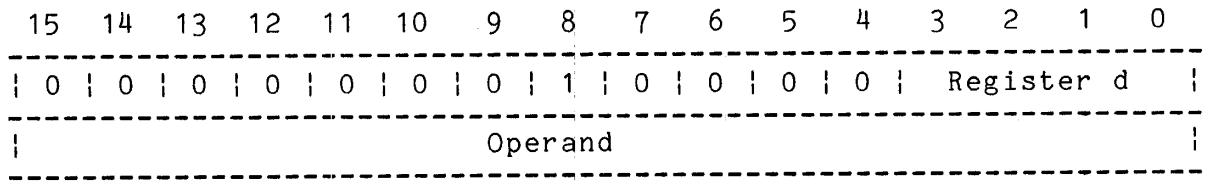
Definition:
Add Words

Format:

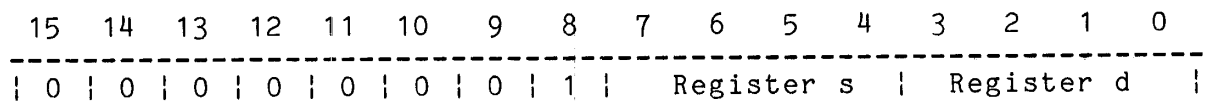
R
81rr



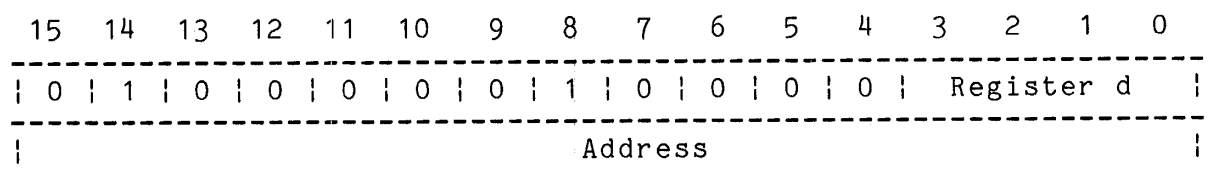
IM
010r



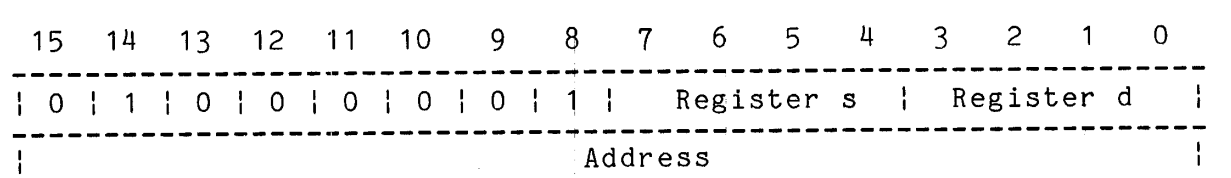
IR
01rr



DA
410r



X
41rr



Addressing Modes:

R - S,NS
IM - S,NS
IR - S,NS
DA - NS
DA - SSO
DA - SLO
X - NS
X - SSO
X - SLO

Operation:

Z8001/2

```
Register d<0:15> <-- Register s<0:15> + Register d<0:15>
```

Flags Affected:

C set to 1 if carry, reset otherwise
Z set to 1 if result = 0, reset otherwise
S set to 1 if result is negative, reset otherwise
P/V set to 1 on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:

The ADD instruction adds the source and destination words and places the sum in the destination register. If a carry results, the Carry flag is set to 1. Otherwise, the Carry flag is reset. The source word is determined by the particular addressing mode used.

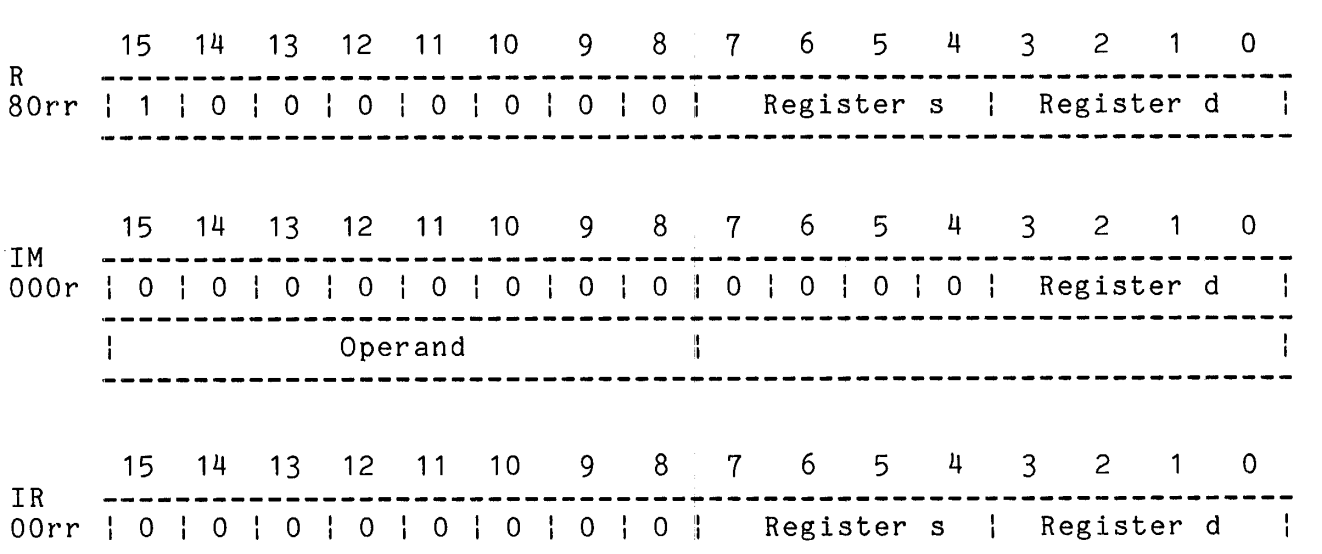
Example:

```
    ADD     R2,R4             !Add the contents of register
                              !R4 to the contents of register
                              !R2, and place the sum in register
                              !R2, changing the carry flag as
                              !appropriate.  (Register Mode)

    ADD     R2,<<2F>>%3000    !Add the contents of segment 2F,
                              !location 3000 to the contents of R2.
                              !Place the sum in R2, changing the
                              !carry flag as appropriate.
                              !(DA Mode, segmented - long offset)
```

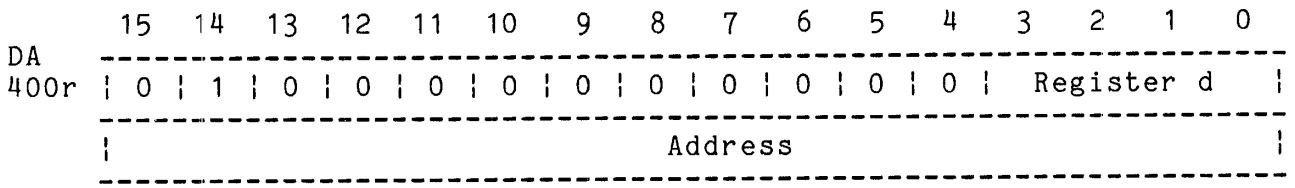
ADDB
(Normal)

Definition:
Add Bytes

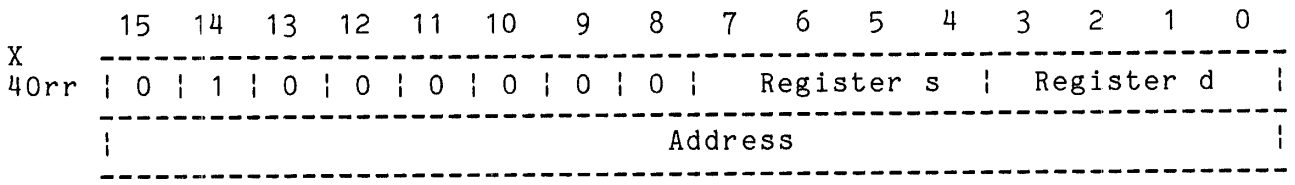
Format:



DA



X



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS
DA - SSO
DA - SLO
X - NS
X - SSO
X - SLO

Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7> + Register d<0:7>

Flags Affected:
C set to 1 if carry, reset otherwise
Z set to 1 if result = 0, reset otherwise
S set to 1 if result is negative, reset otherwise
P/V set to 1 on arithmetic overflow, reset otherwise
DA always reset
H set to 1 if carry from least significant digits

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The ADDB instruction adds the source and destination bytes and places the sum in the destination register. If a carry results, the Carry flag is set to 1. Otherwise, the Carry flag is reset. The source byte is determined by the particular addressing mode used.

Example:

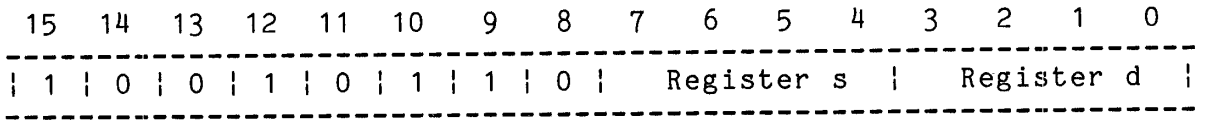
```
    ADDB    RL2,RL4          !Add the contents of register
                             !RL4 to the contents of register
                             !RL2, and place the sum in
                             !register RL2, changing the carry
                             !flag as appropriate.
                             !(Register Mode)

    ADD     R12,<<2F>>%3000  !Add the contents of segment 2F,
                             !location 3000 to the contents of
                             !R12.  Place the sum in R12, changing
                             !the carry flag as appropriate.
                             !(DA Mode, segmented - long offset)
```

ADDL
(Normal)

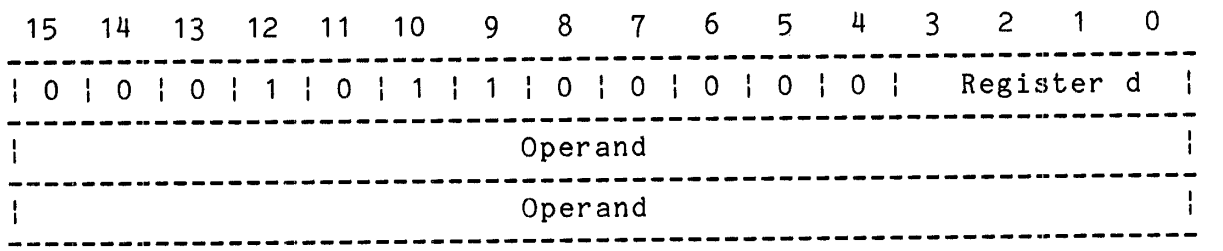
Definition:
Add Long Words

Format:

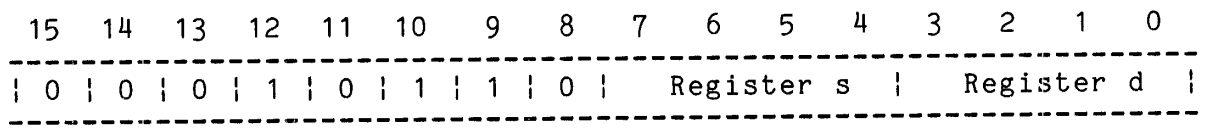
R
96rr



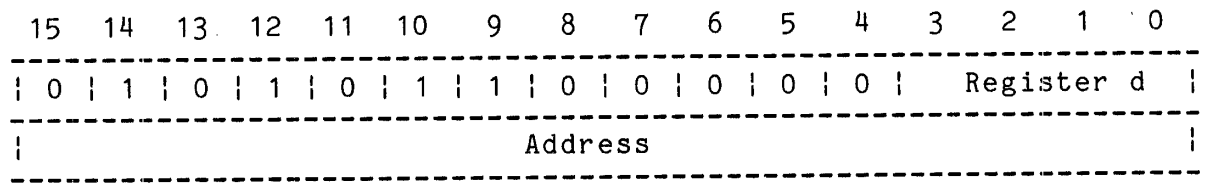
IM
160r



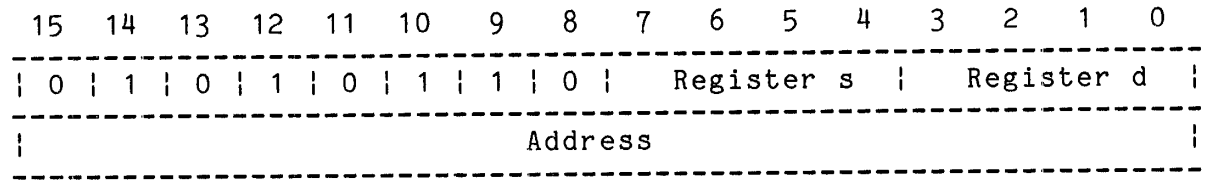
IR
16rr



DA
560r



X
56rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Register d<0:31> <-- Register s<0:31> + Register d<0:31>

Flags Affected:
C set if carry from MSB of long word
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	8
IM - S,NS	14
IR - S,NS	14
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The ADDL instruction adds a 32-bit source value to a 32-bit destination value and places the sum in the destination register pair. The source depends on the addressing mode used. The destination is always a long word register. If a carry occurs, the carry flag is set.

Example:

```
ADDL    RR4,RR8             !Add the contents of RR8 to
                            !the contents of RR4, and
                            !place the sum in RR4.  Set
                            !the carry flag if applicable.
                            !(Register mode)

ADDL    RR4,<<%1A>>%30FF    !Add the contents of segment 1A,
                            !location 30FF to the contents
                            !of register RR4.  Place the sum
                            !in RR4, and set the carry flag
                            !if a carry resulted.
                            !(DA mode, Segmented - Long Offset)
```

CLR
(Normal)

Definition:
Clear Word

Format:

R
8Dr8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	1	Register d				1	0	0	0

IR
0Dr8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	Register d				1	0	0	0

DA
4D08

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	1	0	0	0	0	1	0	0	0
Address															

```
X
4Dr8  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
     ---------------------------------------------------------------
     | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |   Register d  | 1 | 0 | 0 | 0 |
     ---------------------------------------------------------------
     |                          Address                              |
     ---------------------------------------------------------------
```

Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Register d<0:15> <-- 0
or
Destination<0:15> <-- 0

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	7
IR - S,NS	8
DA - NS	11
DA - SSO	12
DA - SLO	14
X - NS	12
X - SSO	12
X - SLO	15

Description:
The CLR instruction sets all bits of the specified destination to zero. The destination is determined by the addressing mode used.

Example:

```
CLR     R4              !Clear R4.
                        !(R mode)

CLR     <<%1F>>%2A      !Clear segment 1F, location
                        !002A.
                        !(DA mode, Segmented - Short Offset)
```

CLRB
(Normal)

Definition:
Clear Byte

Format:

R
8Cr8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0	Register d				1	0	0	0

IR
0Cr8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	Register d				1	0	0	0

DA
4C08

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0
Address															

X
4Cr8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	Register d				1	0	0	0
Address															

Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Register d<0:7> <-- 0
or
Destination<0:7> <-- 0

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	7
IR - S,NS	8
DA - NS	11
DA - SSO	12
DA - SLO	14
X - NS	12
X - SSO	12
X - SLO	15

Description:
The CLRB instruction sets all bits of the specified destination to zero. The destination is determined by the addressing mode used.

Example:

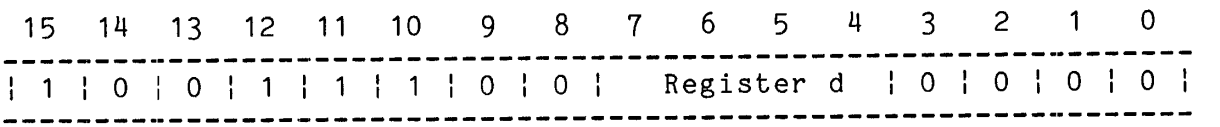
```
    CLRB    RL4             !Clear RL4.
                            !(R mode)

    CLRB    <<%1F>>%2A      !Clear segment 1F, location
                            !002A.
                            !(DA mode, Segmented - Short Offset)
```

CLRL
(Normal)

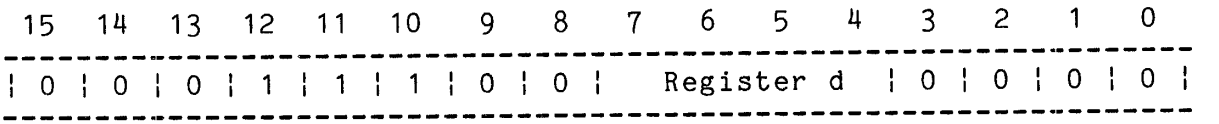
Definition:
Clear Long Word

Format:

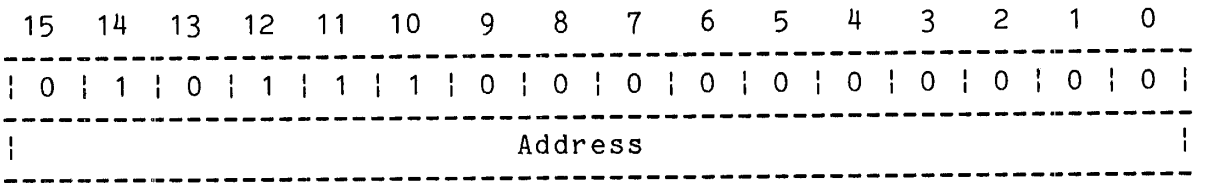
R
9Cr0



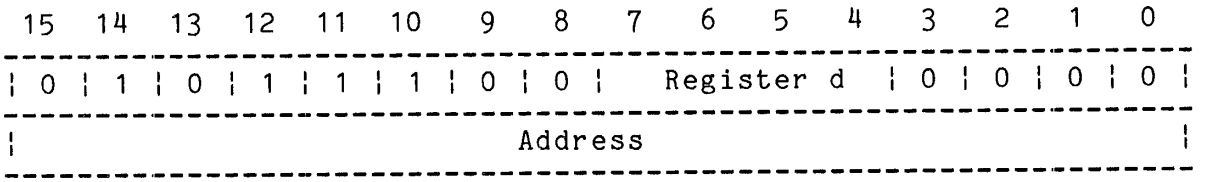
IR
1Cr0



DA
5C00



X
5Cr0



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Register d<0:31> <-- 0
or
Destination<0:31> <-- 0

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	5
IR - S,NS	11
DA - NS	14
DA - SSO	15
DA - SLO	17
X - NS	15
X - SSO	15
X - SLO	18

Description:
The CLRL instruction sets all bits of the specified destination to zero. The destination is determined by the addressing mode used.

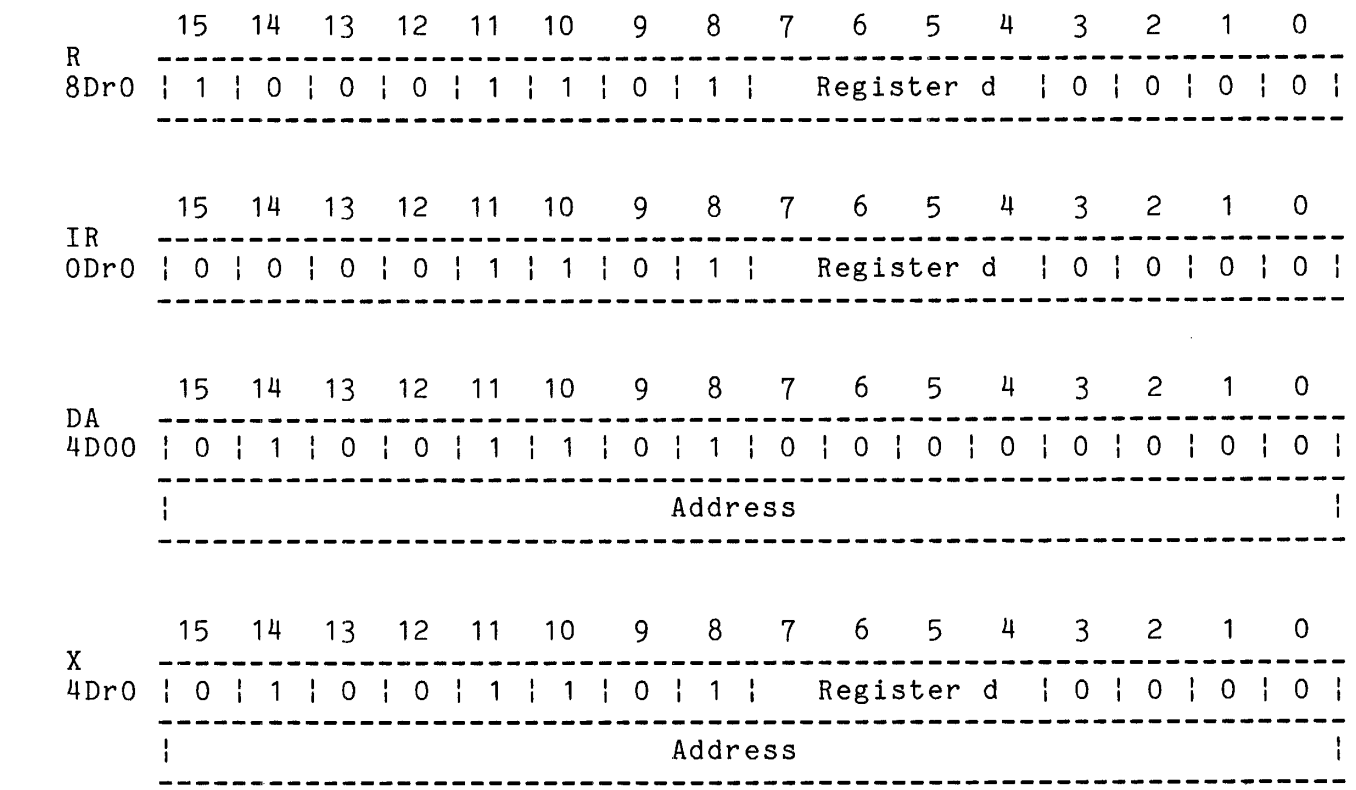
Example:

```
    CLRL    RR4             !Clear RR4.
                            !(R mode)

    CLRL    <<%1F>>%2A      !Clear segment 1F, long word
                            !starting at 002A.
                                !(DA mode, Segmented - Short Offset)
```

COM
(Normal)

Definition:
Complement Word

Format:



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Register d<0:15> <-- Register d<0:15>
or
Destination<0:31> <-- Destination<0:15>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	7
IR - S,NS	12
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The COM instruction complements the contents of the destination word. The destination is determined by the specific addressing mode used.

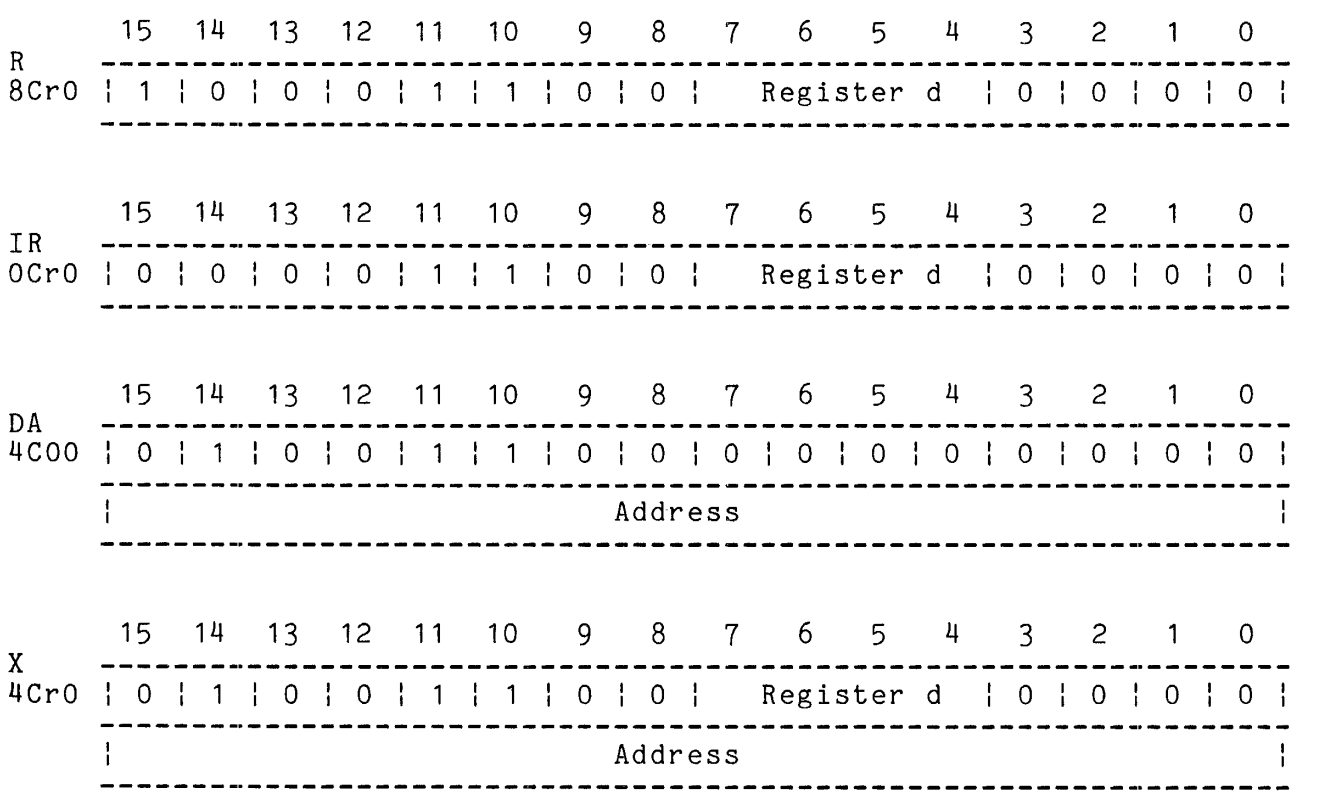
Example:

```
    COM     R2              !Complement the contents of
                            !register R2.
                            !(R mode)

    COM     <<%1F>>%3F00    !Complement the contents of
                            !segment 1F, location 3F00.
                            !(DA mode, Segmented - Long Offset)
```

COMB
(Normal)

Definition:
Complement Byte

Format:



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Register d<0:7> <-- Register d<0:7>
or
Destination<0:7> <-- Destination<0:7>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if parity even, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	7
IR - S,NS	12
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The COMB instruction complements the contents of the destination byte. The destination is determined by the specific addressing mode used.

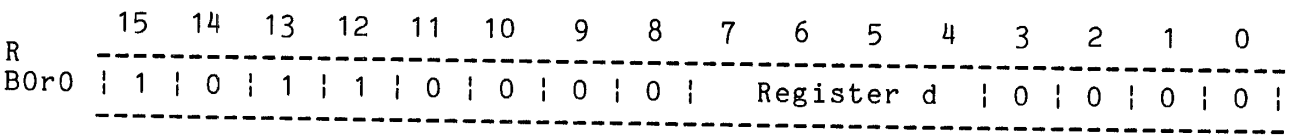
Example:

```
    COMB    RL2             !Complement the contents of
                            !register RL2.
                            !(R mode)

    COMB    <<%1F>>%3F00    !Complement the contents of
                            !segment 1F, location 3F00.
                            !(DA mode, Segmented - Long Offset)
```

DAB
(Normal)

Definition:
Decimal Adjust Byte

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register d<0:7> + BCD<0:7>

Flags Affected:
C set or reset according to table below
Z set if result = 0, reset otherwise
S set if the MSB of the result is set, reset otherwise

Clock Cycles = 5

Description:
The DAB instruction performs a decimal adjust on the byte contained in the register specified in the instruction.

The DAB operation involves adding a binary-coded decimal (BCD) value to the byte register contents. This converts the byte to a two digit BCD representation. The DAB instruction is used following the ADDB, ADCB, SUBB, or SBCB instructions to generate a valid BCD answer.

The exact value added to the byte register depends upon the previous instruction executed, and the states of the DA, C, and H flags which are set by those instructions, and the contents of the byte register. The following table lists the BCD values added to the register contents.

Table 2-8
DAB Operations

Preceding Instruction	DA Flag	C Flag Before DAB	Bits 4-7	H Flag Before DAB	Bits 0-3	BCD Value	C Flag After DAB
ADDB or ADCB	0	0	0-9	0	0-9	00	0
		0	0-8	0	A-F	06	0
		0	0-9	1	0-3	06	0
		0	A-F	0	0-9	60	1
		0	9-F	0	A-F	66	1
		0	A-F	1	0-3	66	1
		1	0-2	0	0-9	60	1
		1	0-2	0	A-F	66	1
		1	0-3	1	0-3	66	1
SUBB or SBCB	1	0	0-9	0	0-9	00	0
		0	0-8	1	6-F	FA	0
		1	7-F	0	0-9	A0	1
		1	6-F	1	6-F	9A	1

Example:
As an example, let's assume an ADDB instruction was executed, the sum contained in register RH4 was 4B, and the C and H flags = 0.

```
DAB     RH4
```

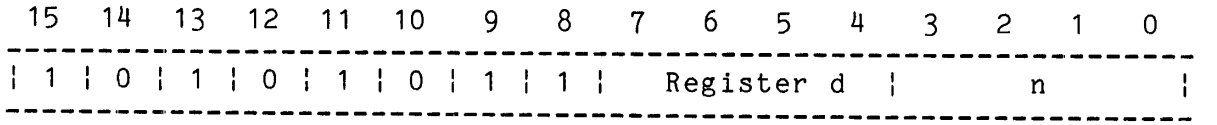
Using the table, you can see that, if C = 0, the high-order nybble falls between 0-9, H = 0, and the low-order nybble falls between A-F, the BCD value added to 4B will be 06. Therefore, the BCD adjusted sum will be 51.

DEC
(Normal)

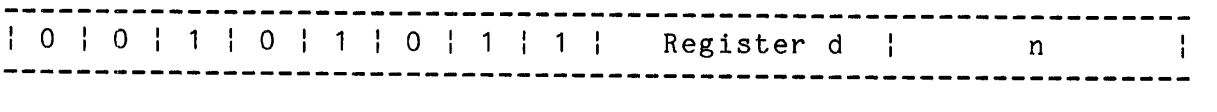
Definition:
Decrement a Word

Format:

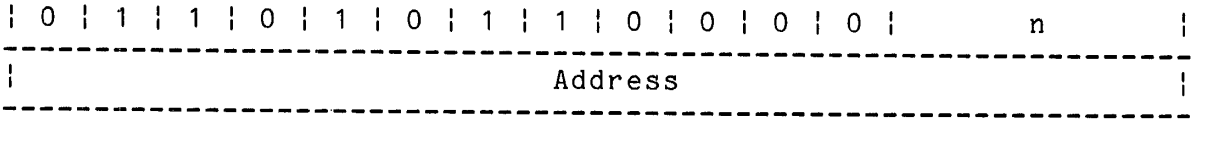
R
ABrn



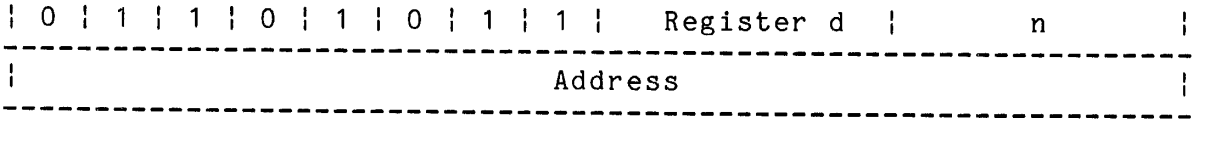
IR
2Brn



DA
6B0n



X
6Brn



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:15> <-- Destination<0:15> - n - 1

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

Description:
The DEC instruction decrements the selected word register or location contents by "n". The DEC instruction somewhat resembles a SUB instruction, except that the value subtracted (n) can only range from 1 to 16.
If "n" = 0, the destination operand is decremented by 1. If "n" = 1111, the destination operand is decremented by 16.

Example:

```
DEC     R2      !Decrement the contents of register R2
                !by 1.  If no "n" value is given, "n"
                !defaults to 0.

DEC     R2,#8   !Decrement the contents of register R2
                !by 8.
```

DECB
(Normal)

Definition:
Decrement Byte

Format:

R
AArn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	0	Register d					n		

IR
2Arn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	Register d					n		

DA
6A0n

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	0	0	0		n		
Address															

X
6Arn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	Register d					n		
Address															

Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:7> <-- Destination<0:7> - n - 1

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

Description:
The DEC instruction decrements the selected byte register or location contents by "n". The DEC instruction somewhat resembles a SUB instruction, except that the value subtracted (n) can only range from 1 to 16.

If "n" = 0, the destination operand is decremented by 1. If "n" = 1111, the destination operand is decremented by 16. If no value for "n" is given in the instruction, it defaults to zero.

Example:

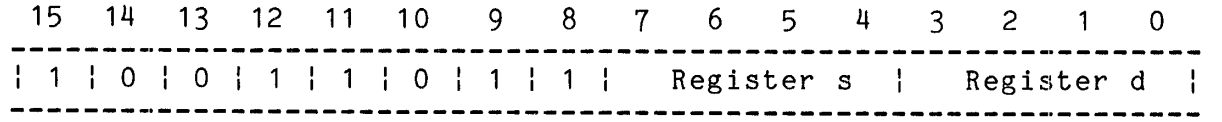
```
    DEC     RH2     !Decrement the contents of register RH2
                    !by 1.  If no "n" value is given, "n"
                    !defaults to 0.

    DEC     RH2,#8  !Decrement the contents of register RH2
                    !by 8.
```

DIV
(Normal)

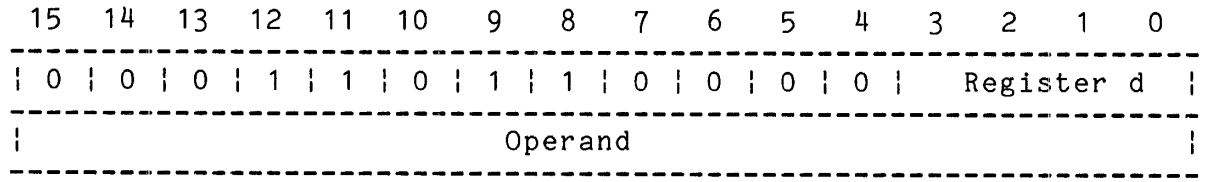
Definition:
Divide Register Pair by Source Word

Format:

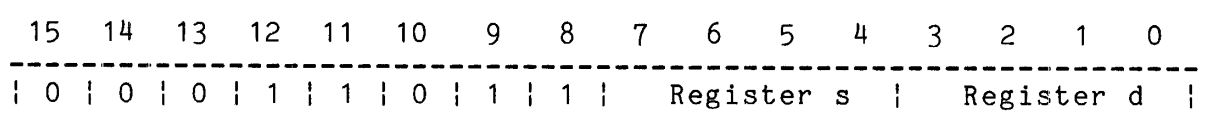
R 9Brr



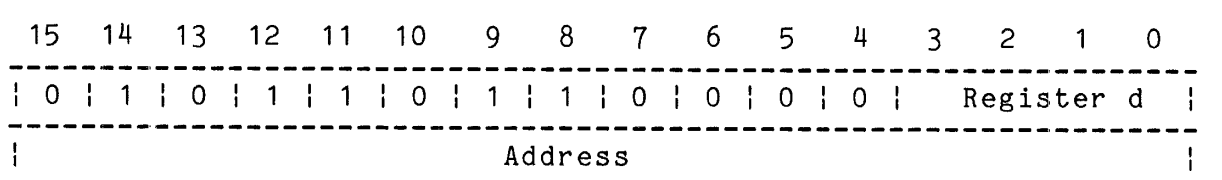
IM 1B0r



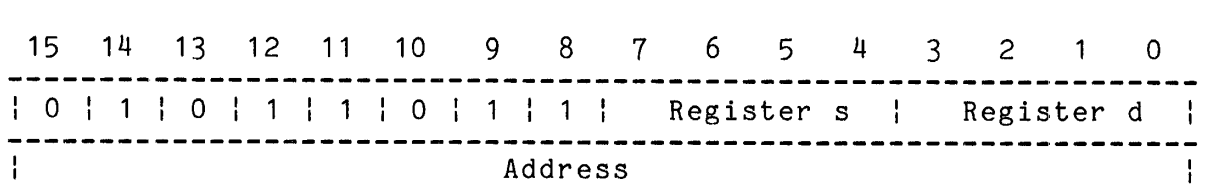
IR 1Brr



DA 5B0r



X 5Brr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:15> <-- Destination<0:31>/Source<0:15>
Destination<16:31> <-- Remainder

Flags Affected:
C set if quotient is <-2^15 or >=+2^15-1, reset otherwise
Z set if either quotient or divisor = 0, reset otherwise
S set if quotient is negative, reset otherwise
P/V set if division is aborted, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	95
IM - S,NS	95
IR - S,NS	95
DA - NS	96
DA - SSO	97
DA - SLO	99
X - NS	97
X - SSO	97
X - SLO	100
Aborted	>30

Description:
The DIV instruction divides a 16-bit signed integer source operand into a 32-bit signed integer destination register pair (Register d). The source operand depends on the addressing mode. The DIV instruction produces a 16-bit quotient and a 16-bit remainder. The quotient is loaded into the low-order register of the destination register pair. The remainder is loaded into the high-order register of the destination register pair. For example, if the destination register pair were RR4, the quotient would be loaded into R5, while the remainder would be loaded into R4.

The original contents of the destination register pair are lost unless the operation is aborted. Abortion occurs if the divisor = 0, or if the magnitude of the divisor is less

than or equal to the magnitude of the high-order half of the dividend.

Example:

```
DIV     RR8,#%30FF      !Divide the contents of register
                        !pair RR8 by 30FF.  Place the
                        !quotient and remainder in RR8.
```

DIVL
(Normal)

Definition:
Divide Register Quad by Source Long Word

Format:

R 9Arr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	0	Register s				Register d			

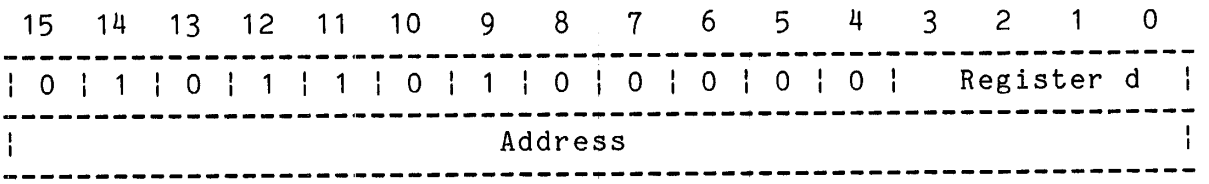
IM 1AOr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	0	0	0	0	0	Register d			
31						Operand									16
15						Operand									0

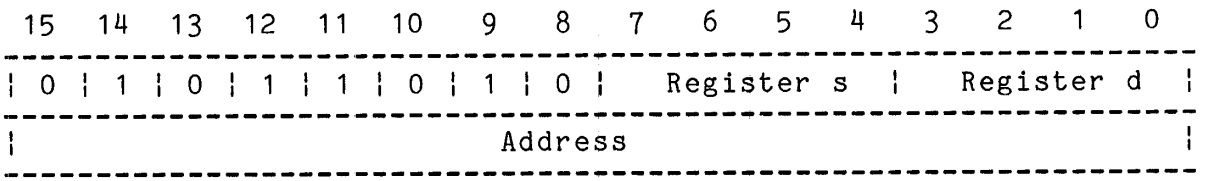
IR 1Arr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	0	Register s				Register d			

DA 5A0r



X 5Arr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:31> <-- Destination<0:63>/Source<0:31>
Destination<32:63> <-- Remainder

Flags Affected:
C set if quotient is <-2^31 or >=+2^31-1, reset otherwise
Z set if either quotient or divisor = 0, reset otherwise
S set if quotient is negative, reset otherwise
P/V set if division is aborted, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	723
IM - S,NS	723
IR - S,NS	723
DA - NS	724
DA - SSO	725
DA - SLO	727
X - NS	725
X - SSO	725
X - SLO	728
Aborted	>60

Description:
The DIVL instruction divides a 32-bit signed integer source operand into a 64-bit signed integer destination register quad (Register d). The source operand depends on the addressing mode. The DIVL instruction produces a 32-bit quotient and a 32-bit remainder. The quotient is loaded into the low-order register pair of the destination register quad. The remainder is loaded into the high-order register pair of the destination register quad. For example, if the destination register quad were RQ4, the quotient would be loaded into RR6, while the remainder would be loaded into RR4.

The original contents of the destination register quad are lost unless the operation is aborted. Abortion occurs if the divisor = 0, or if the magnitude of the divisor is less than or equal to the magnitude of the high-order half of the dividend.

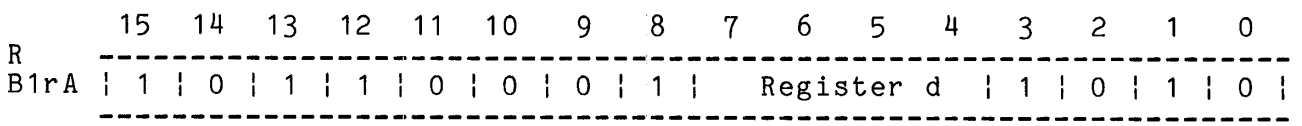
Example:

```
    DIVL    RQ4,#%30FF0020  !Divide the contents of register
                            !quad RQ4 by 30FF0020.  Place the
                            !quotient and remainder in RQ4.
```

EXTS
(Normal)

Definition:
Extend Sign of a Word

Format:



Operation:
Z8001/2

```
If Register d<0:15> is negative,
Register d<16:31> <-- 1, otherwise
Register d<16:31> <-- 0
```

Flags Affected: None

Clock Cycles = 11

Description:
The EXTS instruction extends the sign of a 16-bit integer into a 32-bit integer, while retaining the original value. For example, if the low-order register of register pair RR4 contains a positive integer, then the EXTS instruction will fill the high-order register with 0's.

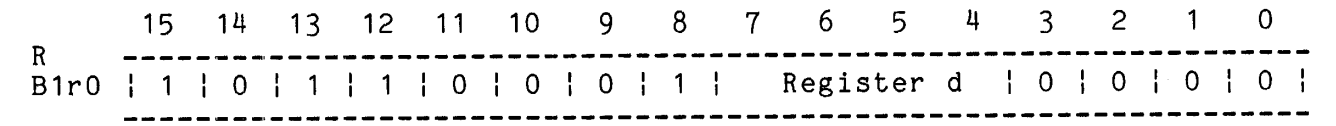
Example:

```
    EXTS    RR4     !While maintaining the value of the low-order
                    !register (R5), fill the high-order register
                    !(R4) with the sign of R5.
```

EXTSB
(Normal)

Definition:
Extend Sign of a Byte

Format:



Operation:
Z8001/2

If Register d<0:7> is negative,
Register d<8:15> <-- 1, otherwise
Register d<8:15> <-- 0

Flags Affected: None

Clock Cycles = 11

Description:
The EXTSB instruction extends the sign of a 8-bit integer into a 16-bit integer, while retaining the original value. For example, if the low-order byte register of register R4 contains a positive integer, then the EXTSB instruction will fill the high-order byte register with 0's.

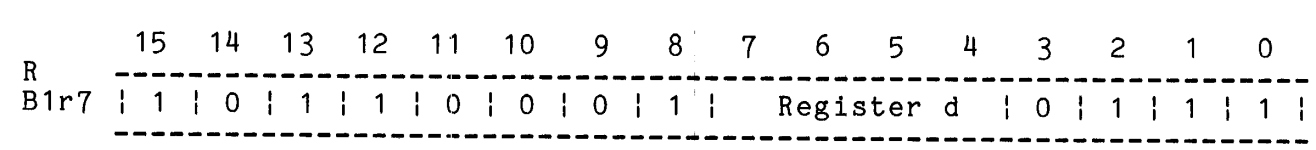
Example:

```
    EXTSB   R4      !While maintaining the value of the low-order
                    !register (RH5), fill the high-order register
                    !(RH4) with the sign of RH5.
```

EXTSL
(Normal)

Definition:
Extend Sign of a Long Word

Format:



Operation:
Z8001/2

If Register d<0:31> is negative,
Register d<32:63> <-- 1, otherwise
Register d<32:63> <-- 0

Flags Affected: None

Clock Cycles = 11

Description:
The EXTSL instruction extends the sign of a 32-bit integer into a 64-bit integer, while retaining the original value. For example, if the low-order register pair of RQ4 contains a positive integer, then the EXTSL instruction will fill the high-order register pair with 0's.

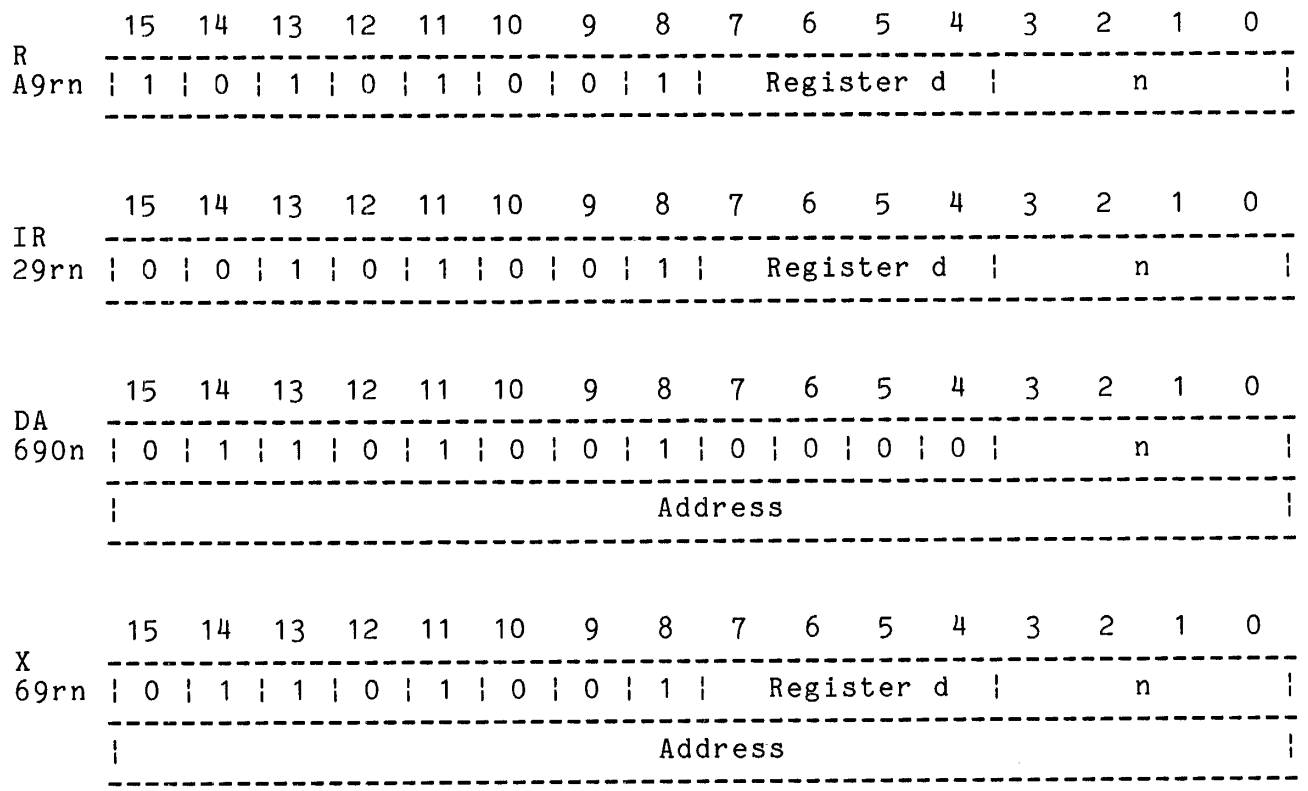
Example:

```
    EXTSL   RQ4     !While maintaining the value of the low-order
                    !register pair (RR5), fill the high-order register
                    !(RR4) with the sign of RR5.
```

INC
(Normal)

Definition:
Increment a Word

Format:



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:15> <-- Destination<0:15> + n + 1

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

Description:
The INC instruction increments the selected word register or location contents by "n". The INC instruction somewhat resembles an ADD instruction, except that the value added (n) can only range from 1 to 16.

If "n" = 0, the destination operand is incremented by 1. If "n" = 1111, the destination operand is incremented by 16.

Example:

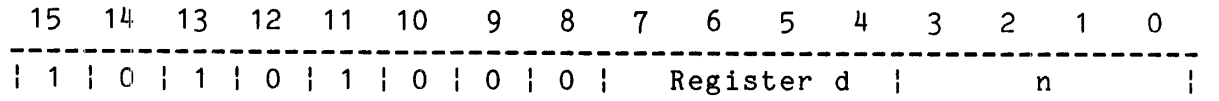
```
    INC     R2      !Increment the contents of register R2
                    !by 1.  If no "n" value is given, "n"
                    !defaults to 0.

    INC     R2,#8   !Increment the contents of register R2
                    !by 8.
```

INCB
(Normal)

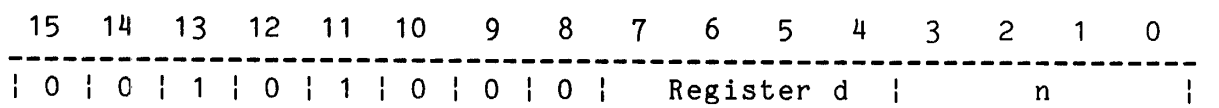
Definition:
Increment Byte

Format:

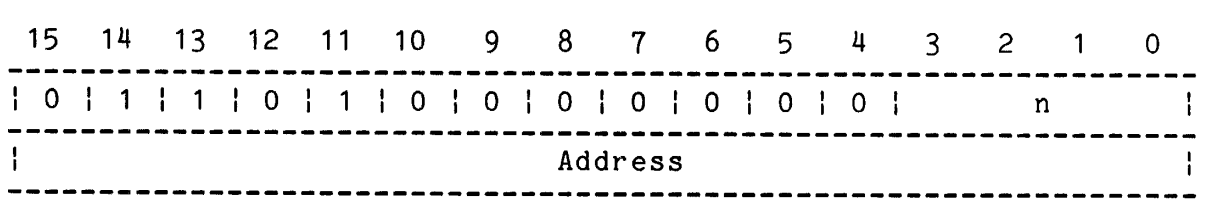
R
A8rn



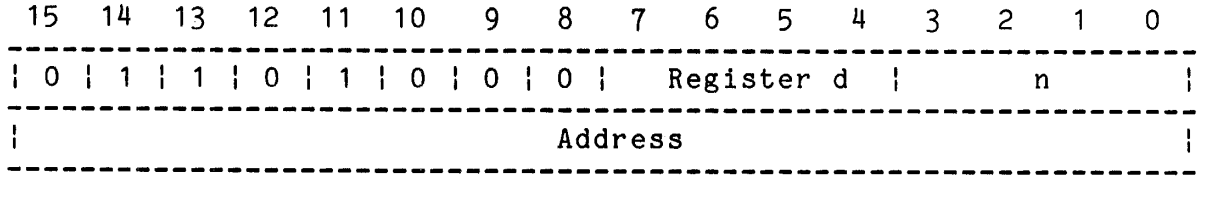
IR
28rn



DA
680n



X
68rn



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:7> <-- Destination<0:7> + n + 1

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

Description:
The INCB instruction increments the selected byte register or location contents by "n". The INCB instruction somewhat resembles an ADDB instruction, except that the value added (n) can only range from 1 to 16.

If "n" = 0, the destination operand is incremented by 1. If "n" = 1111, the destination operand is incremented by 16. If no value for "n" is given in the instruction, it defaults to zero.

Example:

```
INCB    RH2      !Increment the contents of register RH2
                 !by 1.  If no "n" value is given, "n"
                 !defaults to 0.

INCB    RH2,#8   !Increment the contents of register RH2
                 !by 8.
```

MULT
(Normal)

Definition:
Multiply Register with Word

Format

R
99rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	1	Register s				Register d			

IM
190r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	1	0	0	0	0	Register d			

IR
19rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	1	Register s				Register d			

DA
590r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	0	0	0	0	Register d			
Address															

X
59rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	Register s				Register d			
Address															

Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

```
-------
Destination<0:31> <-- Destination<0:15> x Source<0:15>
```

Flags Affected:
C set if product <-2^15 or >=+2^15-1, reset otherwise
Z set if product = 0, reset otherwise
S set if product is negative, reset otherwise
P/V reset

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	70
IM - S,NS	70
IR - S,NS	70
DA - NS	71
DA - SSO	72
DA - SLO	74
X - NS	72
X - SSO	72
X - SLO	75

Description:
The MULT instruction multiplies the low-order word of the 32-bit destination register pair (multiplicand) with the 16-bit source word (multiplier). The high-order word of the destination register pair is ignored and overwritten. The result of the operation is loaded into the destination register pair. The multiplicand and multiplier are signed two's complement 16-bit integers.

Example:

```
    MULT    RR2,#%30FF      !Multiply the low-order contents of RR2
                            !by 30FF, and place the product in RR2.
```

MULTL
(Normal)

Definition:
Multiply a Register by a Long Word

Format:

R
98rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	Register s				Register d			

IM
180r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	0	0	0	0	Register d			

IR
18rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	8	Register s				Register d			

DA
580r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	0	0	0	0	0	Register d			
Address															

X
58rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	0	Register s				Register d			
Address															

Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

```
Destination<0:63> <-- Destination<0:31> x Source<0:31>
```

Flags Affected:
C set if product <-2^31 or >=+2^31-1, reset otherwise
Z set if product = 0, reset otherwise
S set if product is negative, reset otherwise
P/V reset

Clock Cycles =

Addressing Mode	Clock Cycles *
R - S,NS	282 + 7n
IM - S,NS	282 + 7n
IR - S,NS	282 + 7n
DA - NS	283 + 7n
DA - SSO	284 + 7n
DA - SLO	286 + 7n
X - NS	284 + 7n
X - SSO	284 + 7n
X - SLO	287 + 7n

* n = number of bits equal to 1 in the absolute value of the low-order half of the destination.

Description:
The MULTL instruction multiplies the low-order pair of the 64-bit destination register quad (multiplicand) with the 32-bit source word (multiplier). The high-order pair of the destination register quad is ignored and overwritten. The result of the operation is loaded into the destination register quad. The multiplicand and multiplier are signed two's complement 32-bit integers.

Example:

```
    MULTL   RQ4,#%30FF002A  !Multiply the low-order contents
                            !of RQ4 by 30FF002A, and place the
                            !product in RQ4.
```

NEG
(Normal)

Definition:
Negate (2's Complement) a Word

Format:

R
8Dr2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	1	Register d				0	0	1	0

IR
0Dr2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	Register d				0	0	1	0

DA
4D02

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	1	0	0	0	0	0	0	1	0
Address															

X
4Dr2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	1	Register d				0	0	1	0
Address															

Addressing Modes
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:15> <-- Destination<0:15> + 1

Flags Affected:
C reset on carry from destination, set otherwise
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if operand = 8000, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	7
IR - S,NS	12
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The NEG instruction replaces the contents of the destination word with its two's complement. The location of the destination depends on the addressing mode used.

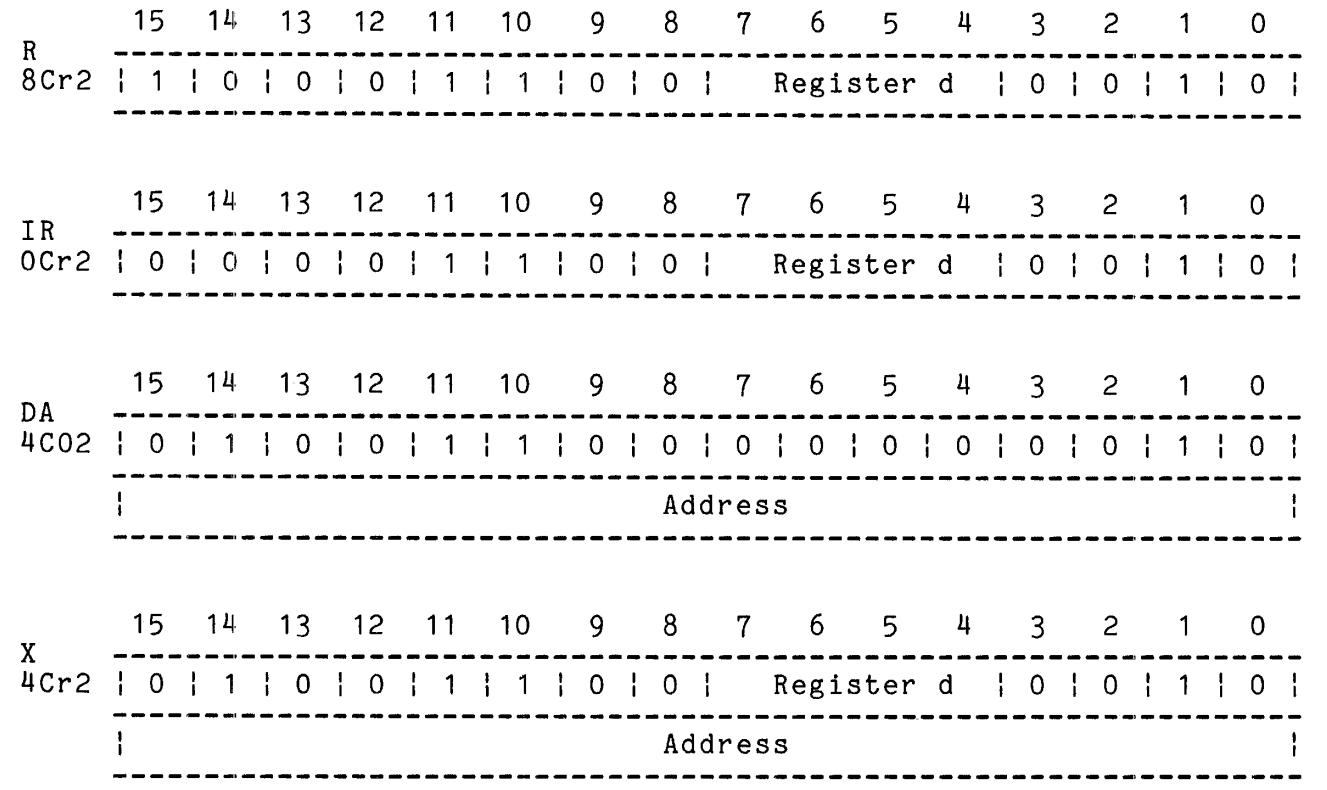
Example:

```
    NEG     R4      !Negate the contents of word register
                    !R4.
```

NEGB
(Normal)

Definition:
Negate (Two's Complement) a Byte

Format:



Addressing Modes
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:7> <-- Destination<0:7> + 1

Flags Affected:
C reset on carry from destination, set otherwise
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if operand = 8000, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	7
IR - S,NS	12
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The NEGB instruction replaces the contents of the destination byte with its two's complement. The location of the destination depends on the addressing mode used.

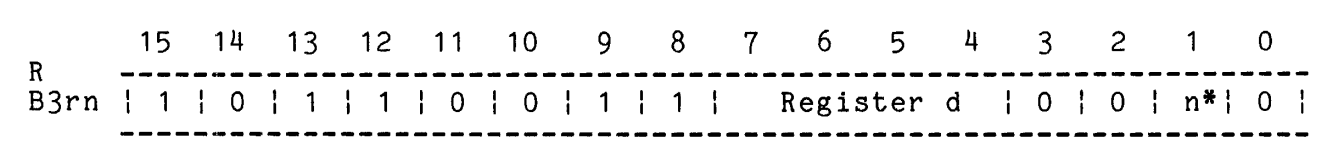
Example:

```
    NEGB    RL4     !Negate the contents of byte register
                    !RL4.
```

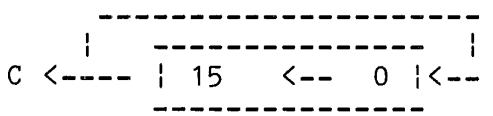
RL
(Normal)

Definition:
Rotate Word Left and into Carry

Format:



Operation:



* If n = 0, word is rotated 1 place.
  If n = 1, word is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RL instruction rotates a word either one or two places left, depending on the state of bit 1 of the instruction. If bit 1 = 0, the word is rotated one place. If bit 1 = 1, the word is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

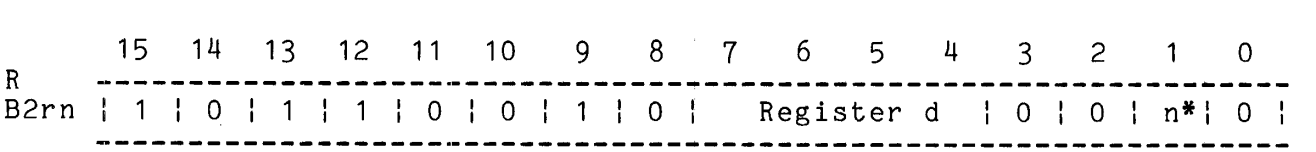
Example:

```
    RL      R4,#1           !Rotate the contents of
                            !register R4 one place to
                            !the left and into carry.
```

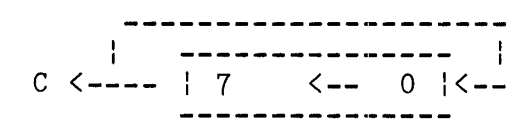
RLB
(Normal)

Definition:
Rotate Byte Left and into Carry

Format:



Operation:



* If n = 0, byte is rotated 1 place.
If n = 1, byte is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RLB instruction rotates a byte either one or two places left, depending on the state of bit 1 of the instruction. If bit 1 = 0, the byte is rotated one place. If bit 1 = 1, the byte is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

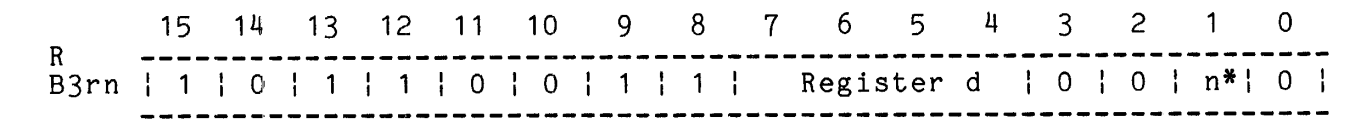
Example:

```
    RLB     RL4,#1          !Rotate the contents of
                            !register RL4 one place to
                            !the left and into carry.
```

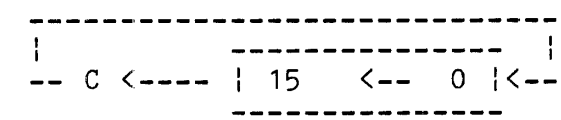
RLC
(Normal)

Definition:
Rotate Word Left and Through Carry

Format:



Operation:



* If n = 0, word is rotated 1 place.
  If n = 1, word is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RLC instruction rotates a word either one or two places left and through carry, depending on the state of bit 1 of the instruction. If bit 1 = 0, the word is rotated one place. If bit 1 = 1, the word is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

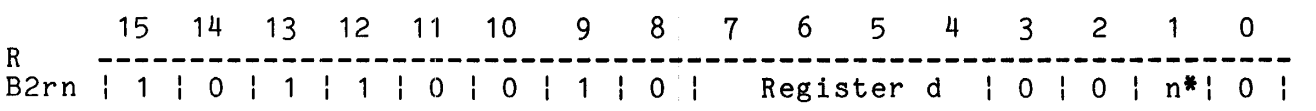
Example:

```
    RLC     R4,#1           !Rotate the contents of
                            !register R4 one place to
                            !the left and through carry.
```

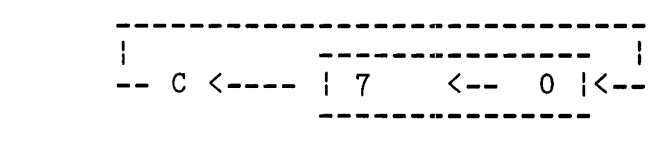
RLCB
(Normal)

Definition:
Rotate Byte Left and Through Carry

Format:



Operation:



* If n = 0, byte is rotated 1 place.
  If n = 1, byte is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RLCB instruction rotates a byte either one or two places left and through carry, depending on the state of bit 1 of the instruction. If bit 1 = 0, the byte is rotated one place. If bit 1 = 1, the byte is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

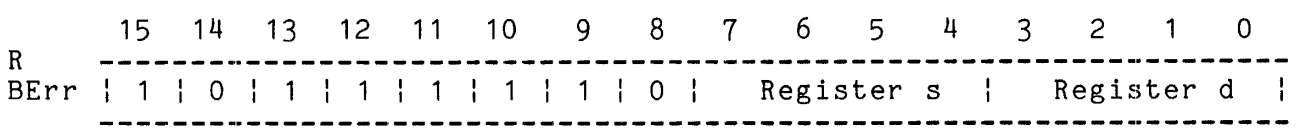
Example:

```
        RLCB    RL4,#1          !Rotate the contents of
                                !register RL4 one place to
                                !the left and through carry.
```

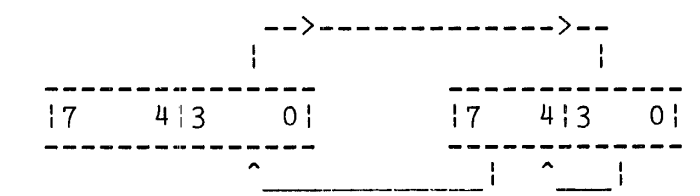
RLDB
(Normal)

Definition:
Rotate Digit Left

Format:



Operation:



Flags Affected:
Z set if destination result = 0, reset otherwise
S set if MSB of destination result = 1, reset otherwise

Clock Cycles = 9

Description:
The RLDB instruction rotates the contents of Register 's' one byte to the left, and into the low-order byte of Register 'd'. The high-order byte of Register 'd' remains unchanged. This instruction is normally used to shift-left BCD data.

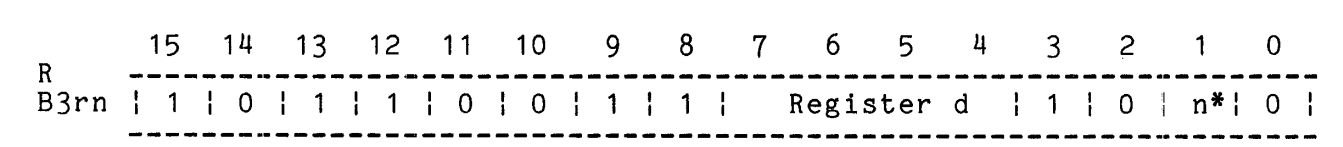
Example:

```
    RLDB     RH2,RL2          !Shift the high-order nybble
                              !of RL2 into the low-order
                              !position of RH2.  Shift the
                              !low-order nybble of RL2 into
                              !the high-order position.  Shift
                              !the old low-order nybble of RH2
                              !into the low-order nybble of
                              !RL2.
```

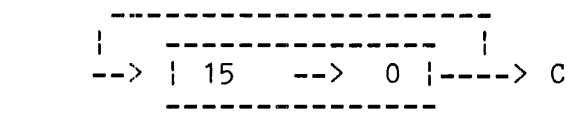
RR
(Normal)

Definition:
Rotate Word Right and into Carry

Format:



Operation:



* If n = 0, word is rotated 1 place.
  If n = 1, word is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RR instruction rotates a word either one or two places right, depending on the state of bit 1 of the instruction. If bit 1 = 0, the word is rotated one place. If bit 1 = 1, the word is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

Example:

```
        RR      R4,#1           !Rotate the contents of
                                !register R4 one place to
                                !the right and into carry.
```

RRB
(Normal)

Definition:
Rotate Byte Right and into Carry

Format:

```
          15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
R         ---------------------------------------------------------------
B2rn      | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |   Register d  | 0 | 1 | n*| 0 |
          ---------------------------------------------------------------
```

Operation:

```
      ---------------------
     |   ---------------   |
     --> | 7    -->  0 |-----> C
         ---------------
```

* If n = 0, byte is rotated 1 place.
If n = 1, byte is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RLB instruction rotates a byte either one or two places right, depending on the state of bit 1 of the instruction. If bit 1 = 0, the byte is rotated one place. If bit 1 = 1, the byte is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

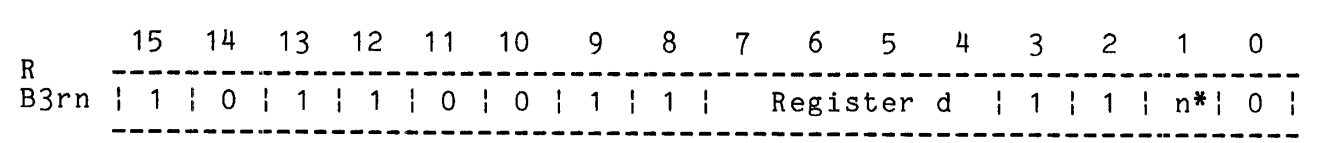
Example:

```
    RLB     RL4,#1          !Rotate the contents of
                            !register RL4 one place to
                            !the right and into carry.
```

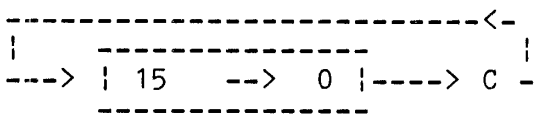
RRC
(Normal)

Definition:
Rotate Word Right and Through Carry

Format:



Operation:



* If n = 0, word is rotated 1 place.
  If n = 1, word is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RRC instruction rotates a word either one or two places right and through carry, depending on the state of bit 1 of the instruction. If bit 1 = 0, the word is rotated one place. If bit 1 = 1, the word is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

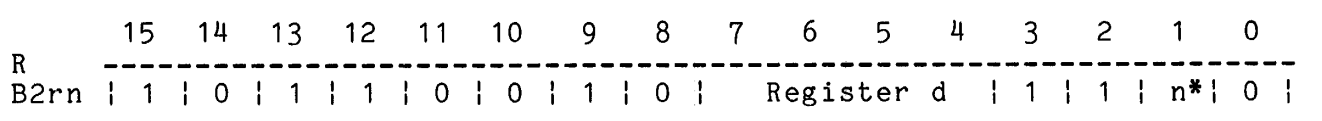
Example:

```
     RRC      R4,#1            !Rotate the contents of
                               !register R4 one place to
                               !the right and through carry.
```

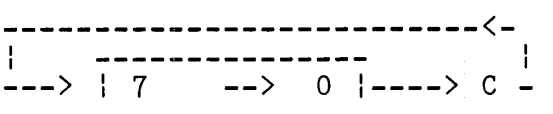
RRCB
(Normal)

Definition:
Rotate Byte Right and Through Carry

Format:



Operation:



* If n = 0, byte is rotated 1 place.
  If n = 1, byte is rotated 2 places.

Flags Affected:
C loaded from last bit rotated out of Register 'd'
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if sign of Register 'd' changed, reset otherwise

Clock Cycles =
6 if 1 place
7 if 2 places

Description:
The RRCB instruction rotates a byte either one or two places right and through carry, depending on the state of bit 1 of the instruction. If bit 1 = 0, the byte is rotated one place. If bit 1 = 1, the byte is rotated two places. If the state of bit 1 is undefined in the assembler syntax, bit 1 defaults to 0.

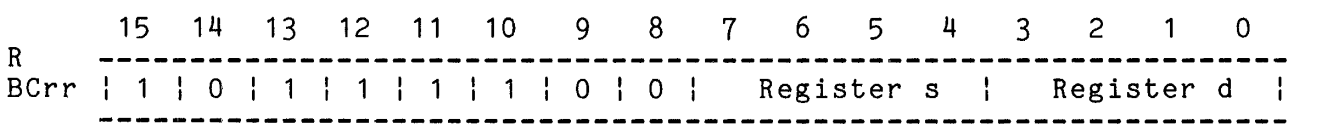
Example:

```
        RRCB    RL4,#1          !Rotate the contents of
                                !register RL4 one place to
                                !the right and through carry.
```

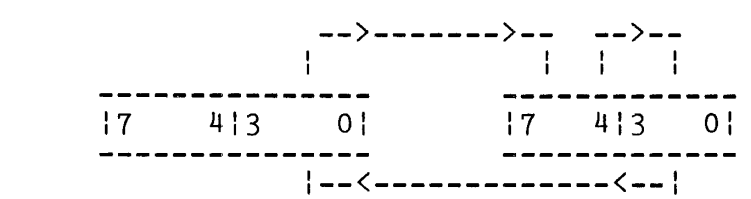
RRDB
(Normal)

Definition:
Rotate Digit Right

Format:



Operation:



Flags Affected:
Z set if destination result = 0, reset otherwise
S set if MSB of destination result = 1, reset otherwise

Clock Cycles = 9

Description:
The RRDB instruction rotates the contents of Register 's' one byte to the right, and into the low-order byte of Register 'd'. The high-order byte of Register 'd' remains unchanged. This instruction is normally used to shift-right BCD data.

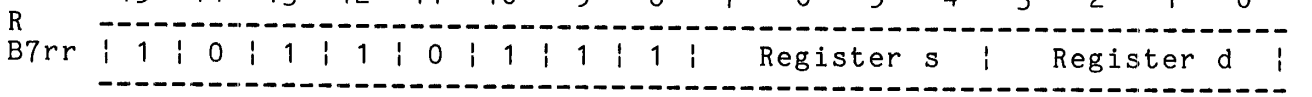
Example:

```
        RRDB    RH2,RL2         !Shift the low-order nybble
                                !of RL2 into the low-order
                                !position of RH2.  Shift the
                                !high-order nybble of RL2 into
                                !the low-order position.  Shift
                                !the old low-order nybble of RH2
                                !into the high-order nybble of
                                !RL2.
```

SBC
(Normal)

Definition:
Subtract Register Words With Carry

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register d<0:15> - Register s<0:15> - C

Flags Affected:
C reset on carry from most significant bit of result,
reset otherwise
Z set if result = 0, reset otherwise
S set if result negative, reset otherwise
P/V set if arithmetic overflow, reset otherwise

Clock Cycles = 5

Description:
The SBC instruction subtracts the contents of Register 's' from Register 'd', along with the contents of the carry flag (C). The result is loaded into Register 'd'.

The subtraction operation is carried out by adding the two's complement of Register 's' to the contents of Register 'd'.

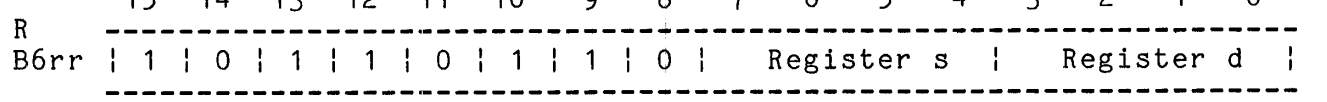
Example:

```
SBC     R2,R4   !Subtract the contents of R4,along with
                !the carry flag, from the contents
                !of R2
```

SBCB
(Normal)

Definition:
Subtract Register Bytes With Carry

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register d<0:7> - Register s<0:7> - C

Flags Affected:
C reset on carry from most significant bit of result,
reset otherwise
Z set if result = 0, reset otherwise
S set if result negative, reset otherwise
P/V set if arithmetic overflow, reset otherwise

Clock Cycles = 5

Description:
The SBCB instruction subtracts the contents of Register 's' from Register 'd', along with the contents of the carry flag (C). The result is loaded into Register 'd'.

The subtraction operation is carried out by adding the two's complement of Register 's' to the contents of Register 'd'.

Example:

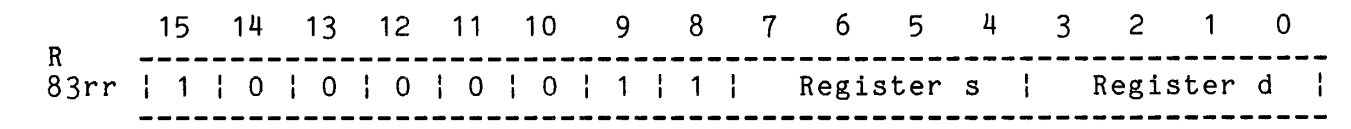
```
SBCB    RH2,RH4    !Subtract the contents of RH4,along with
                   !the carry flag, from the contents
                   !of RH2
```

SUB
(Normal)

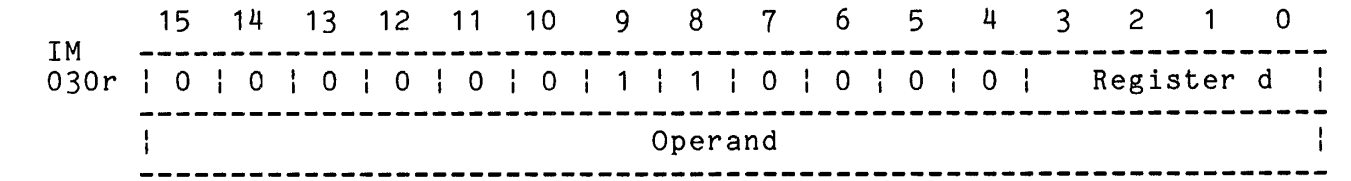
Definition: Subtract Word From Register

Format:

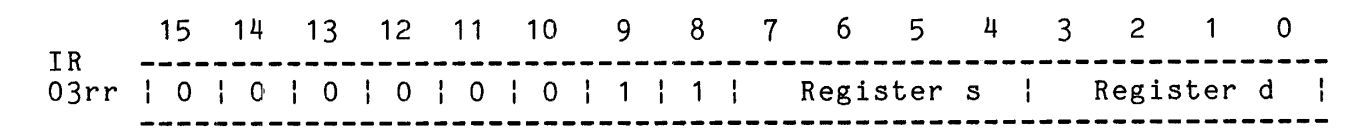
R



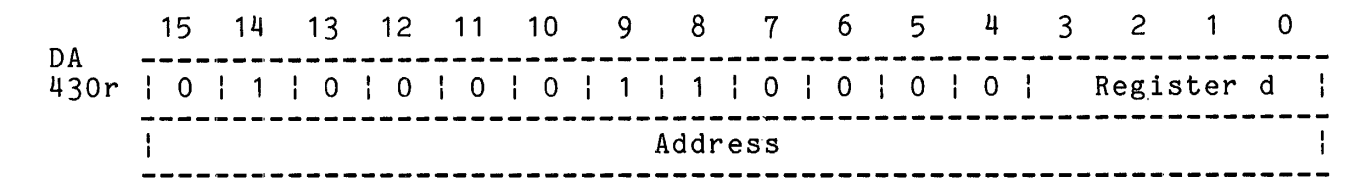
IM



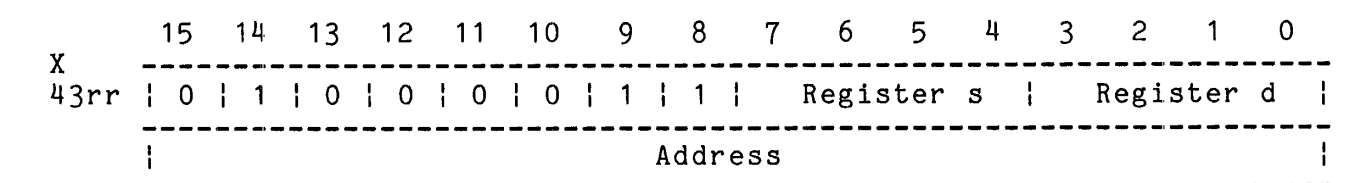
IR



DA



X



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

```
-------
Destination<0:15> <-- Destination<0:15> - Source<0:15>
```

Flags Affected:
C reset on carry from most significant bit of result,
  set otherwise
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The SUB instruction subtracts the source word (depending on the addressing mode) from the contents of the destination register (Register 'd'). The result of the operation is loaded into Register 'd'.

Example:

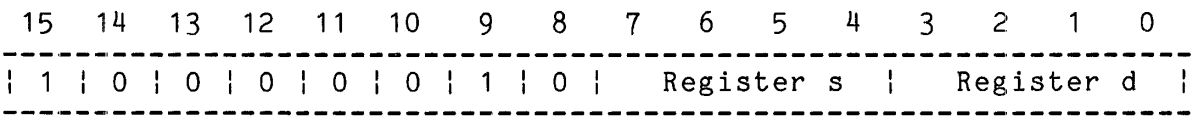
```
    SUB     R4,#100         !Subtract decimal 100 from the
                            !contents of register R4.
                            !(IM mode)
```

SUBB
(Normal)

Definition: Subtract Byte From Register

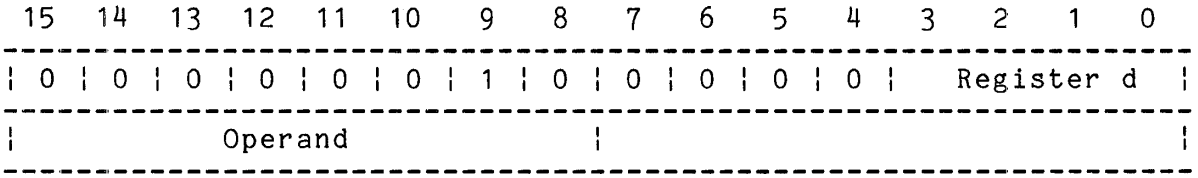
Format:

R



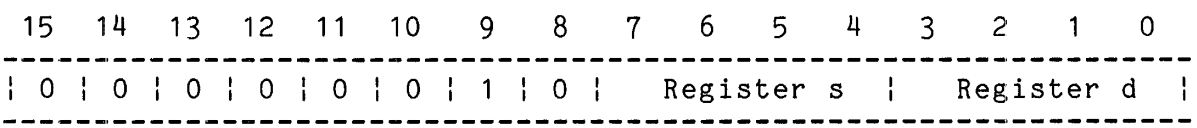
82rr

IM



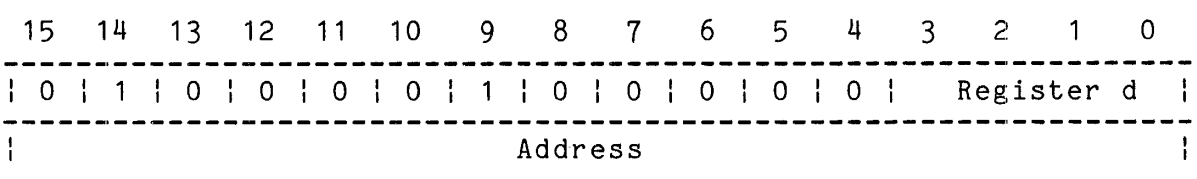
020r

IR



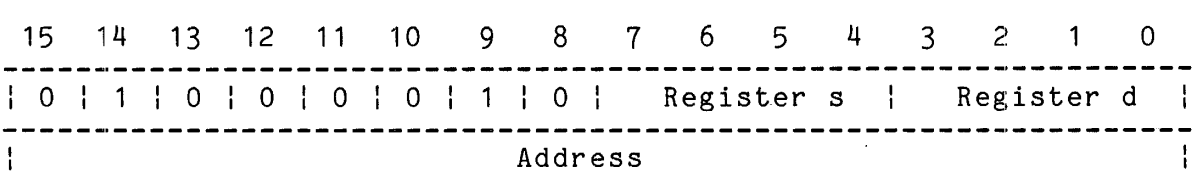
02rr

DA



420r

X



42rr

Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:7> <-- Destination<0:7> - Source<0:7>

Flags Affected:
C reset on carry from most significant bit of result, set otherwise
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The SUBB instruction subtracts the source byte (depending on the addressing mode) from the contents of the destination register (Register 'd'). The result of the operation is loaded into Register 'd'.

Example:

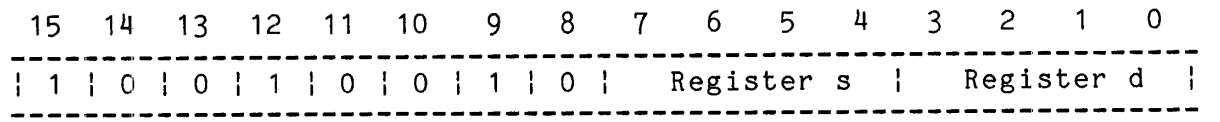
```
    SUBB    RH4,#100        !Subtract decimal 100 from the
                            !contents of register RH4.
                            !(IM mode)
```

SUBL
(Normal)

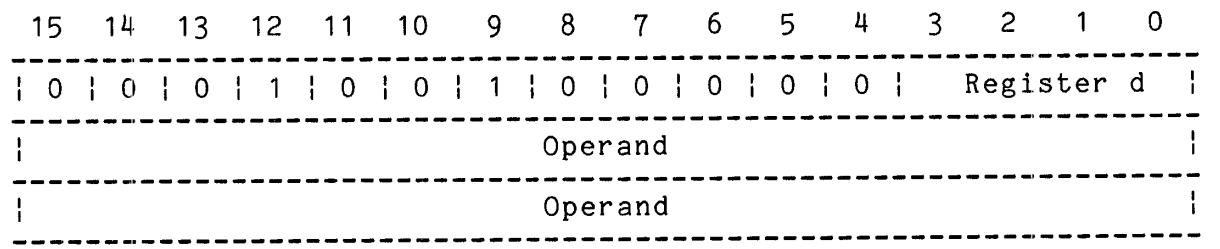
Definition: Subtract Long Word From Register

Format:

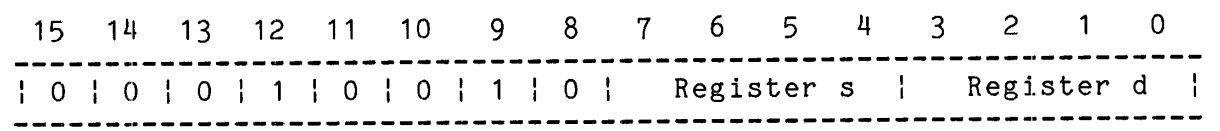
R
92rr



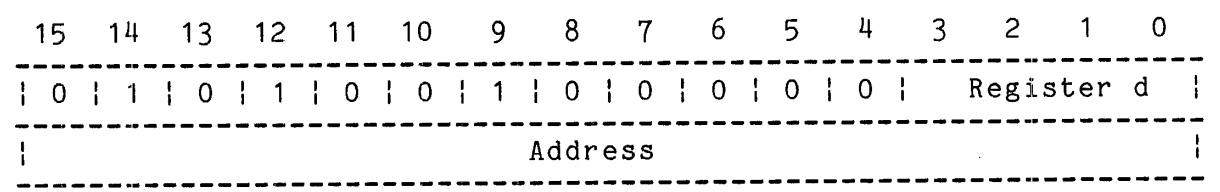
IM
920r



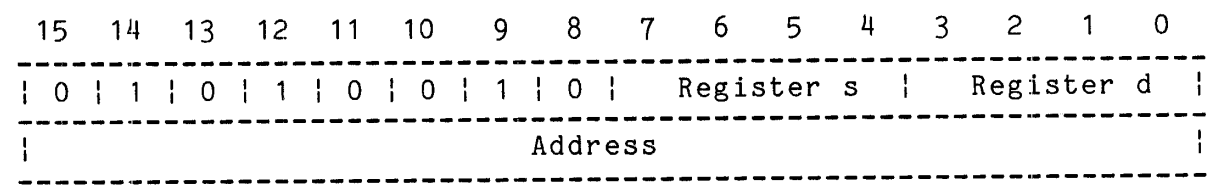
IR
92rr



DA
920r



X
92rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Destination<0:31> <-- Destination<0:31> - Source<0:31>

Flags Affected:
C reset on carry from most significant bit of result, set otherwise
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	8
IM - S,NS	14
IR - S,NS	14
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The SUBL instruction subtracts the source long word (depending on the addressing mode) from the contents of the destination register quad (Register 'd'). The result of the operation is loaded into Register 'd'.

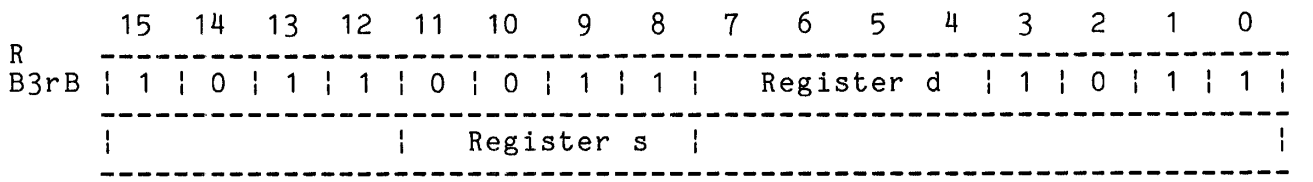
Example:

```
    SUBL    RQ4,#100        !Subtract decimal 100 from the
                            !contents of register quad RQ4.
                            !(IM mode)
```

SDA
(Normal)

Definition:
Arithmetic Shift Word

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register d<0:15> Shifted

Flags Affected:
C loaded from the last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significan bit of Register d = 1, reset otherwise
P/V set if sign of Register d changed, reset otherwise

Clock Cycles = 15 + 3n*
* n = number of places shifted

Description:
The SDA shifts the contents of Register 'd' the direction and number of places specified by the contents of Register 's'. The signed value in Register 'd' maintains its sign during right shifts.

Register 's' contains a signed two's complement integer. If the sign of the integer is positive, a left shift occurs. If the sign of the integer is negative, a right shift occurs. The number of places shifted can be in the range of -16 to +16.

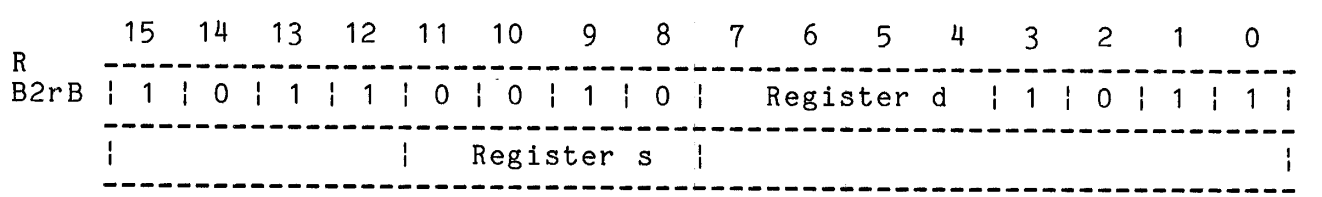
Example:

```
SDA     R7,R2           !Shift R7 in the direction and
                        !the number of places as determined
                        !by the contents of R2.  If R2
                        !contains +3, R7 will be shifted
                        !left 3 places.
```

SDAB
(Normal)

Definition:
Arithmetic Shift Byte

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register d<0:7> Shifted

Flags Affected:
C loaded from the last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significan bit of Register d = 1, reset otherwise
P/V set if sign of Register d changed, reset otherwise

Clock Cycles = 15 + 3n*
* n = number of places shifted

Description:
The SDAB shifts the contents of Register 'd' the direction and number of places specified by the contents of Register 's'. The signed value in Register 'd' maintains its sign during

right shifts.
Register 's' contains a signed two's complement integer. If the sign of the integer is positive, a left shift occurs. If the sign of the integer is negative, a right shift occurs. The number of places shifted can be in the range of -8 to +8.

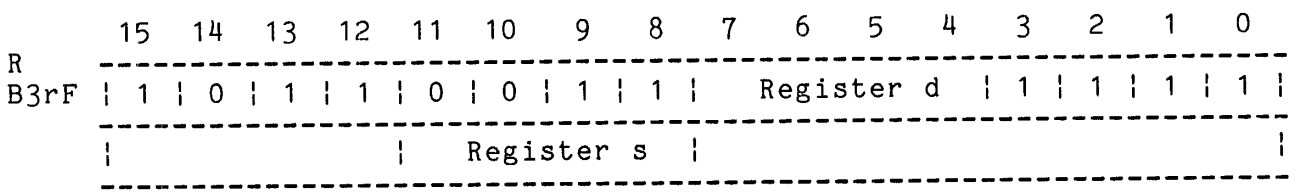
Example:

```
    SDAB    RL7,RL2         !Shift RL7 in the direction and
                            !the number of places as determined
                            !by the contents of RL2.  If RL2
                            !contains +3, RL7 will be shifted
                            !left 3 places.
```

SDAL
(Normal)

Definition:
Arithmetic Shift Long Word

Format:



Operation:
Z8001/2

Register d<0:31> <-- Register d<0:31> Shifted

Flags Affected:
C loaded from the last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significan bit of Register d = 1, reset otherwise
P/V set if sign of Register d changed, reset otherwise

Clock Cycles = 15 + 3n*
 * n = number of places shifted

Description:
The SDAL shifts the contents of Register 'd' the direction and number of places specified by the contents of Register 's'. The signed value in Register 'd' maintains its sign during right shifts.

Register 's' contains a signed two's complement integer. If the sign of the integer is positive, a left shift occurs. If the sign of the integer is negative, a right shift occurs. The number of places shifted can be in the range of -32 to +32.

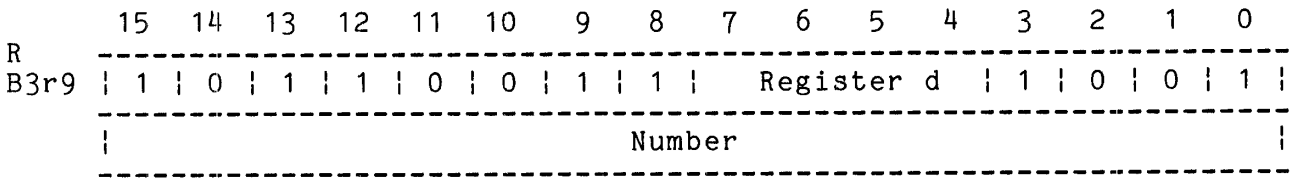
Example:

```
    SDAL    RQ0,RQ4         !Shift RQ0 in the direction and
                            !the number of places as determined
                            !by the contents of RQ4.  If RQ4
                            !contains +3, RQ0 will be shifted
                            !left 3 places.
```

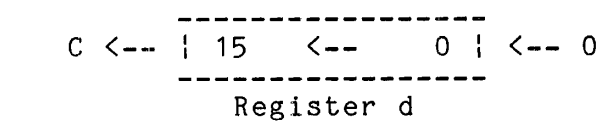
SLA
(Normal)

Definition:
Arithmetic Left Shift Word

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significan bit of Register d = 1, reset otherwise
P/V set if sign of Register d changed, reset otherwise

Clock Cycles = 13 + 3n*
* n = number of places shifted

Description:
The SLA instruction performs a left shift on the contents of Register 'd'. The number of places shifted depends on the contents of the "number" operand. The "number" value must be in the range of 0 to 16, and is a 16-bit two's complement positive integer. The SRA instruction has the same bit pattern as SLA, however SLA uses a positive integer, while SRA uses a negative integer.

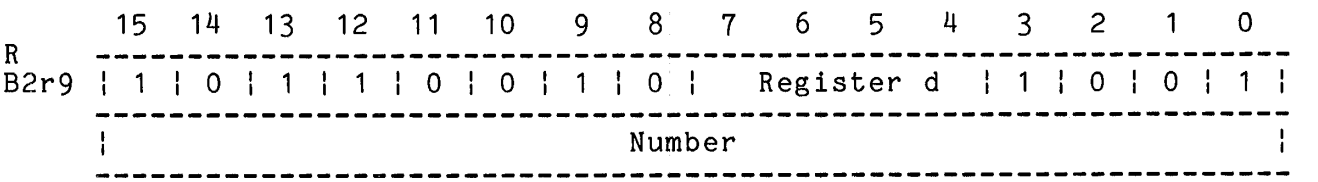
Example:

```
        SLA     R4,#4           !Shift the contents of R4
                                !to the left four places.
```

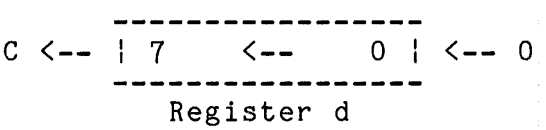
SLAB
(Normal)

Definition:
Arithmetic Left Shift Byte

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significan bit of Register d = 1, reset otherwise
P/V set if sign of Register d changed, reset otherwise

Clock Cycles = 13 + 3n*
* n = number of places shifted

Description:
The SLAB instruction performs a left shift on the contents of Register 'd'. The number of places shifted depends on the contents of the "number" operand. The "number" value must be in the range of 0 to 8, and is a 16-bit two's complement positive integer. The SRAB instruction has the same bit pattern an SLAB, however SRAB uses a negative integer "number".

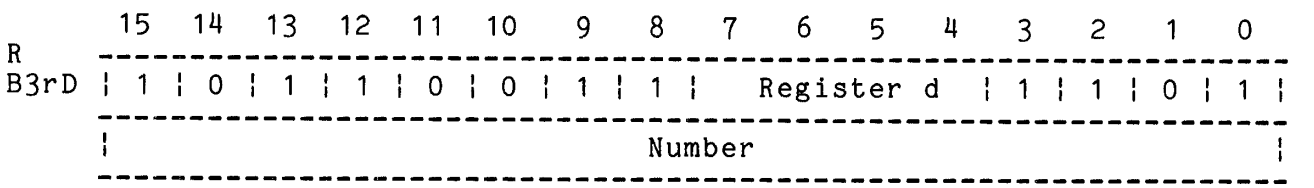
Example:

```
    SLAB    RH4,#4          !Shift the contents of RH4
                            !to the left four places.
```

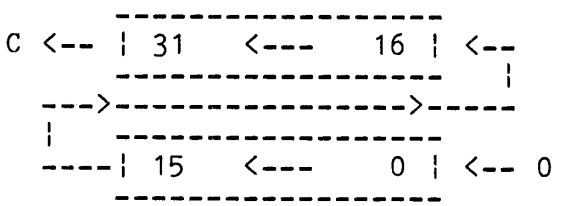
SLAL
(Normal)

Definition:
Arithmetic Left Shift Long Word

Format:



Operation:



Flags Affected:
C loaded from the last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significan bit of Register d = 1, reset otherwise
P/V set if sign of Register d changed, reset otherwise

Clock Cycles = 13 + 3n*
* n = number of places shifted

Description:
The SLAL instruction performs a left shift on the contents of Register 'd'. The number of places shifted depends on the contents of the "number" operand. The "number" value must be in the range of 0 to 32, and is a 16-bit two's complement positive integer. The SRAL instruction has the same bit pattern as SLAL, however SRAL uses a negative integer "number".

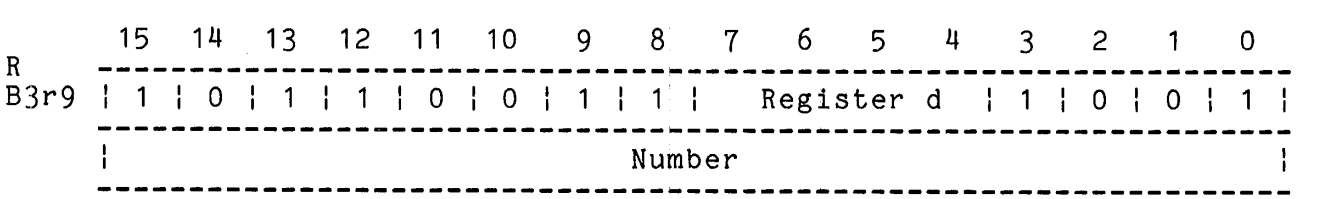
Example:

```
SLAL    RQ4,#4          !Shift the contents of RQ4
                        !to the left four places.
```

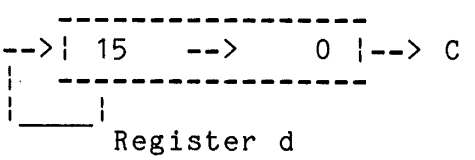
SRA
(Normal)

Definition:
Arithmetic Right Shift Word

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significant bit of Register d = 1, reset otherwise
P/V reset

Clock Cycles = 13 + 3n*
* n = number of places shifted

Description:
The SRA instruction performs a right shift on the contents of Register 'd'. The number of places shifted depends on the contents of the "number" operand. The "number" value must be in the range of 0 to 16, and is a 16-bit two's complement negative integer. The SLA instruction uses the same bit pattern as SRA, however the SLA "number" is a positive integer.

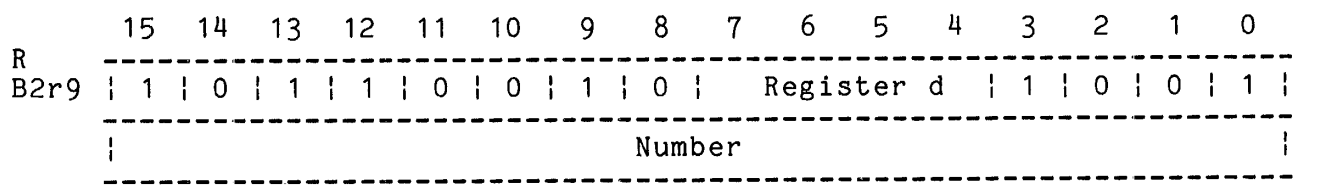
Example:

```
    SRA     R4,#4           !Shift the contents of R4
                            !to the right four places.
```

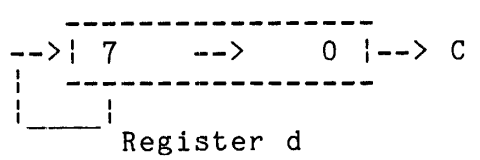
SRAB
(Normal)

Definition:
Arithmetic Right Shift Byte

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significant bit of Register d = 1, reset otherwise
P/V reset

Clock Cycles = 13 + 3n*
* n = number of places shifted

Description:
The SRAB instruction performs a right shift on the contents of Register 'd'. The number of places shifted depends on the contents of the "number" operand. The "number" value must be in the range of 0 to 8, and is a 16-bit two's complement negative integer. The SLAB instruction uses the same bit pattern as SRAB, however the SLAB "number" is a positive integer.

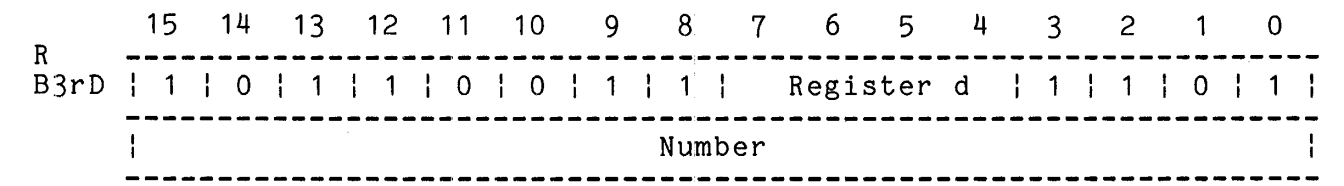
Example:

```
SRAB    RL4,#4          !Shift the contents of RL4
                        !to the right four places.
```

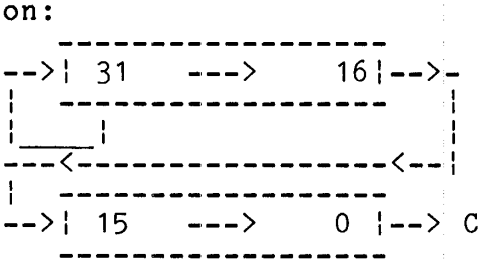
SRAL
(Normal)

Definition:
Arithmetic Right Shift Long Word

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if most significant bit of Register d = 1, reset otherwise
P/V reset

Clock Cycles = 13 + 3n*
* n = number of places shifted

Description:
The SRAL instruction performs a right shift on the contents of Register 'd'. The number of places shifted depends on the contents of the "number" operand. The "number" value must be in the range of 0 to 32, and is a 16-bit two's complement negative integer. The SLAL instruction uses the same bit pattern as SRAL, however the SLAL "number" is a positive integer.

Example:

```
        SRAL    RQ4,#4          !Shift the contents of RQ4
                                !to the right four places.
```

The Logic Group

The logic group is made up of the following instructions:

Table 2-9
The Logic Group

Mnemonic	Function
AND	Logically AND Words
ANDB	Logically AND Bytes
CP	Compare Words
CPB	Compare Bytes
CPL	Compare Long Words
OR	Logically OR Words
ORB	Logically OR Bytes
SDL	Logical Shift Word
SDLB	Logical Shift Byte
SDLL	Logical Shift Long Word
SLL	Logical Left Shift Word
SLLB	Logical Left Shift Byte
SLLL	Logical Left Shift Long Word
SRL	Logical Right Shift Word
SRLB	Logical Right Shift Byte
SRLL	Logical Right Shift Long Word
XOR	Exclusive-OR Words
XORB	Exclusive-OR Bytes

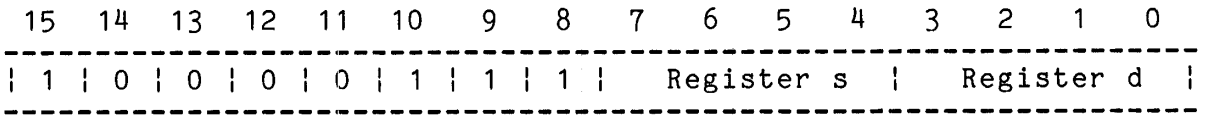
Although the Z8000's autoincrement and autodecrement compare instructions could be considered logical instructions, they are primarily used in string manipulation operations. Therefore, the CPD, CPDB, CPDR, CPDRB, CPI, CPIB, CPIR, and CPIRB instructions are defined as Block Transfer and String Manipulation instructions.

AND
(Normal)

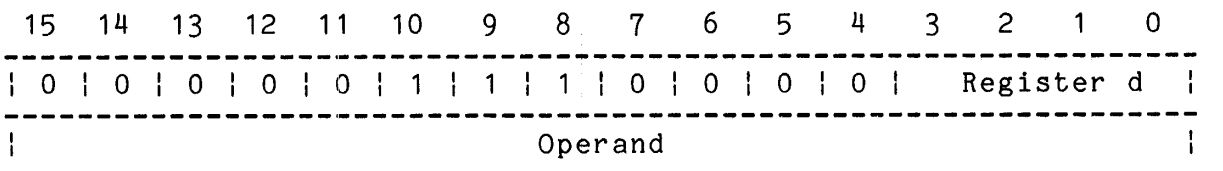
Definition:
Logically AND Word

Format:

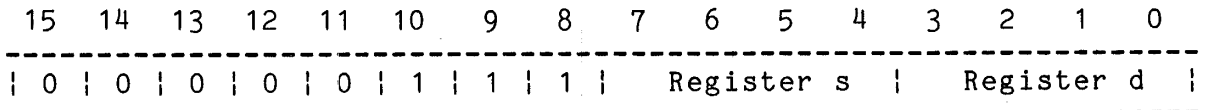
R
87rr



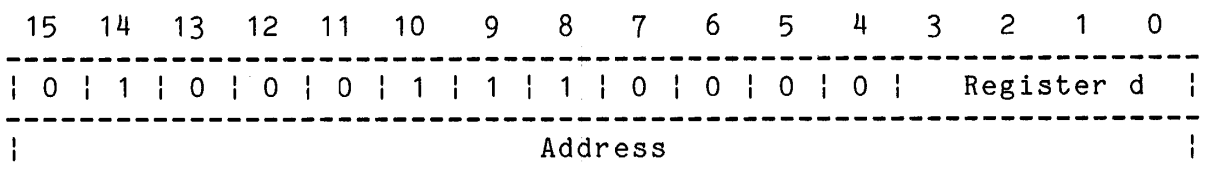
IM
070r



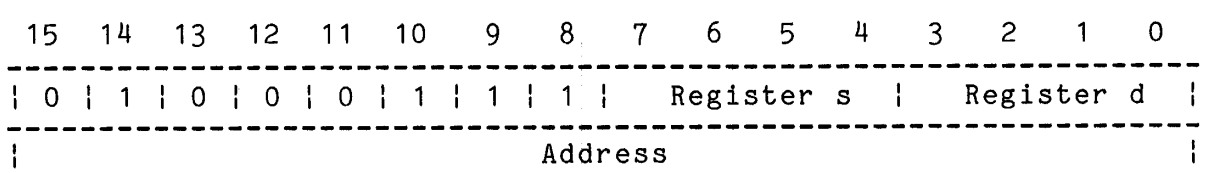
IR
07rr



DA
470r



X
47rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Register d<0:15> <-- Register d<0:15> . source<0:15>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The AND instruction logically ANDs the contents of a 16-bit destination register with the contents of the specified source. The result is placed in Register 'd'.

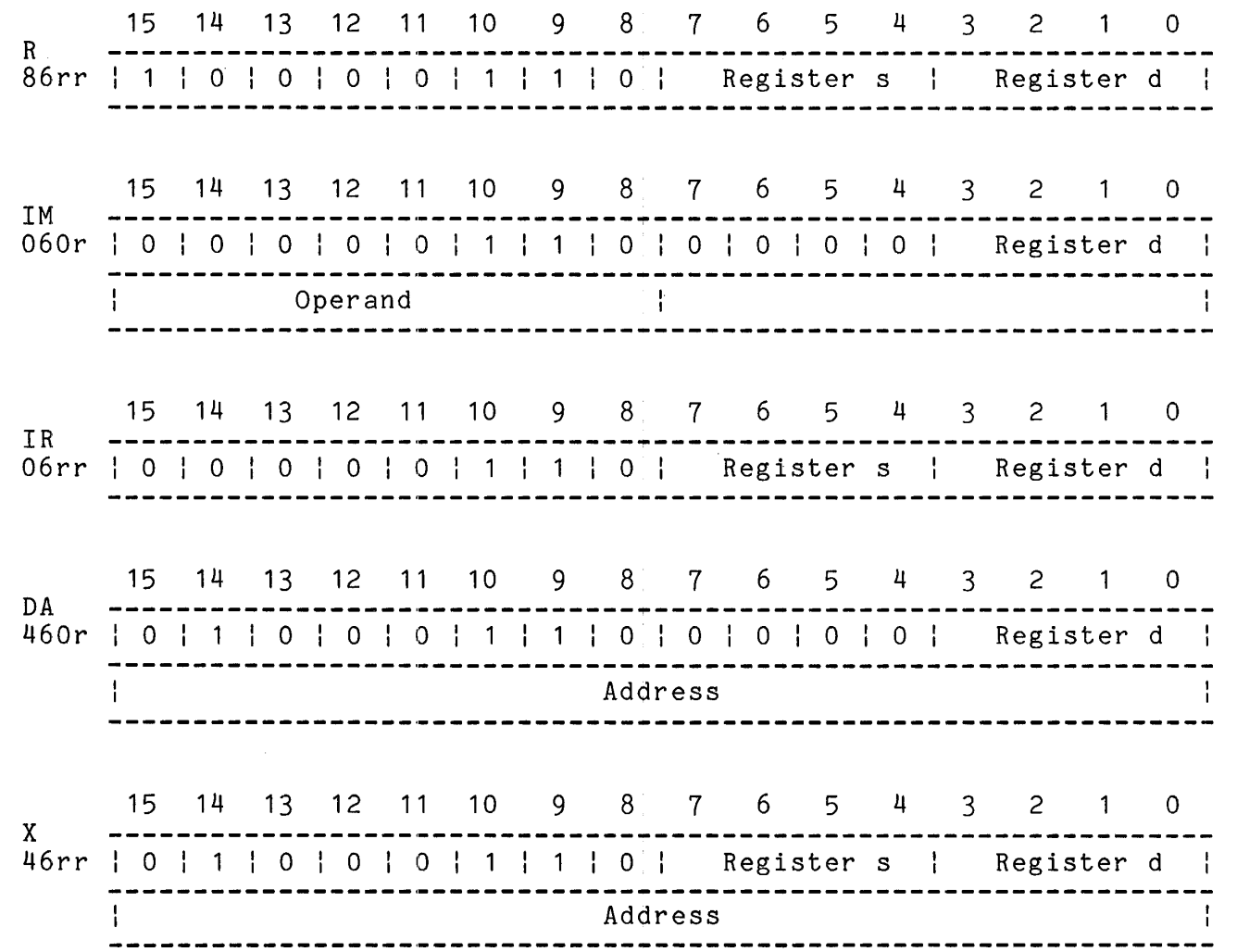
Example:

```
    AND     R2,%30FF        !AND the contents of R2 with the
                            !value 30FF, and place the result
                            !in R2.
                            !(IM mode)
```

ANDB
(Normal)

Definition:
Logically AND Byte

Format:



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Register d<0:7> <-- Register d<0:7> . source<0:7>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The ANDB instruction logically ANDs the contents of an 8-bit destination register with the contents of the specified source. The result is placed in Register 'd'.

Example:

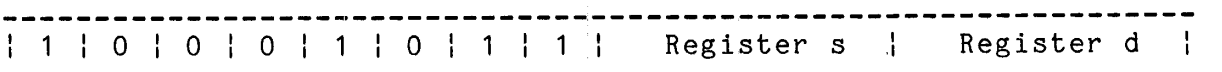
```
    ANDB    RL2,%30FF       !AND the contents of RL2 with the
                            !value 30FF, and place the result
                            !in R2.
                            !(IM mode)
```

CP
(Normal)

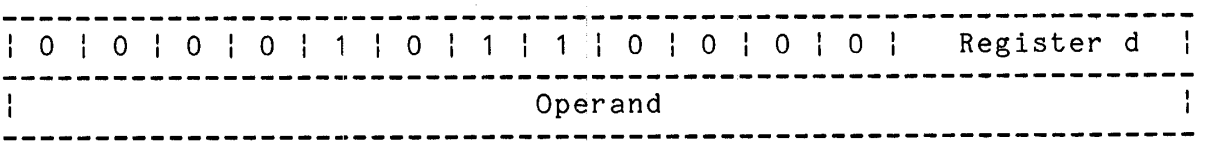
Definition:
Compare Words

Format:

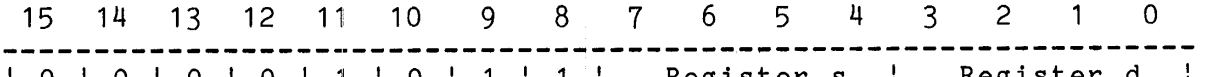
R
8Brr



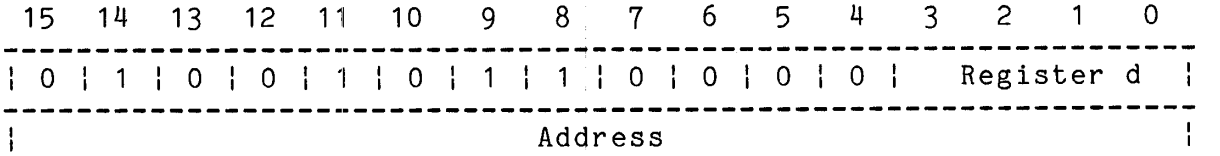
IM
0B0r



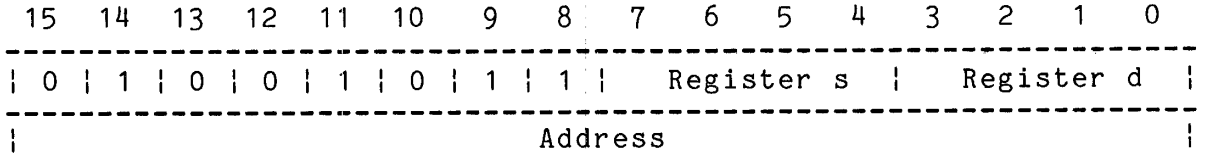
IR
0Brr



DA
4B0r



X
4Brr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

If Register d<0:15> = source, set flags
If Register d<0:15> ≠ source, set flags

Flags Affected:
C reset on carry from MSB of result, set otherwise
Z set if result = 0 (compare is true), reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The CP instruction compares (by subtraction) the source word with the contents of Register 'd'. The result of the operation is reflected in the flags. The CP operation does not affect the contents of either the source or Register 'd'.

Example:

```
CP      R2,<<%7A>>%13F0   !Compare the contents of R2
                          !with the value contained in
                          !segment 7A, location 13F0.
                          !Reset the C flag if a carry
                          !from the most significant bit
                          !of the result occured.  Set the
                          !Z flag if the compare is true,
                          !reset otherwise.
                          !(DA mode, Long Offset)
```

CPB
(Normal)

Definition:
Compare Bytes

Format:

R 8Arr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	Register s				Register d			

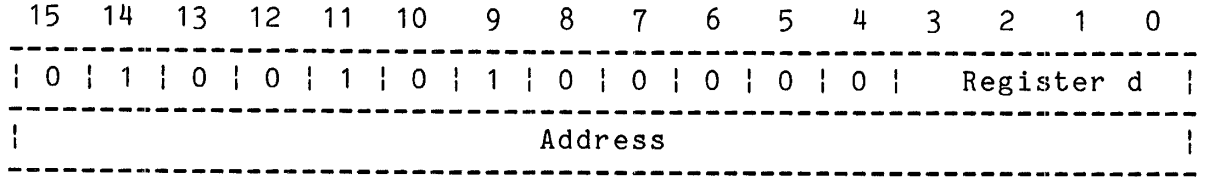
IM 0AOr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	0	0	Register d			
Operand															

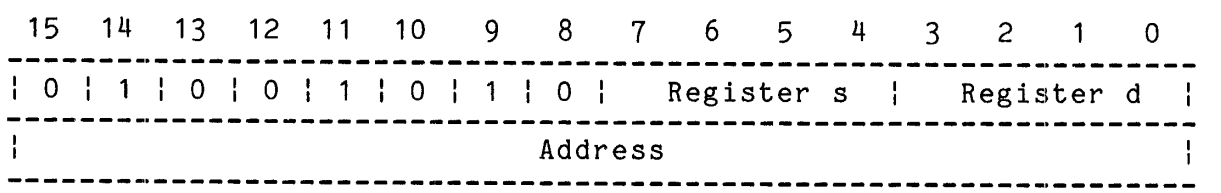
IR 0Arr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	Register s				Register d			

DA 4AOr



X 4Arr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

If Register d<0:7> = source, set flags
If Register d<0:7> ≠ source, set flags

Flags Affected:
C reset on carry from MSB of result, set otherwise
Z set if result = 0 (compare is true), reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The CPB instruction compares (by subtraction) the source byte with the contents of Register 'd'. The result of the operation is reflected in the flags. The CP operation does not affect the contents of either the source or Register 'd'.

Example:

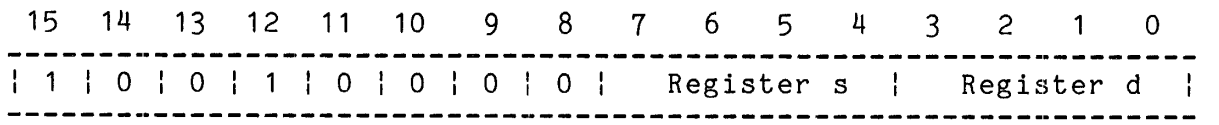
```
    CPB     RL2,<<%7A>>%13F0  !Compare the contents of RL2
                              !with the value contained in
                              !segment 7A, location 13F0.
                              !Reset the C flag if a carry
                              !from the most significant bit
                              !of the result occured.  Set the
                              !Z flag if the compare is true,
                              !reset otherwise.
                              !(DA mode, Long Offset)
```

CPL
(Normal)

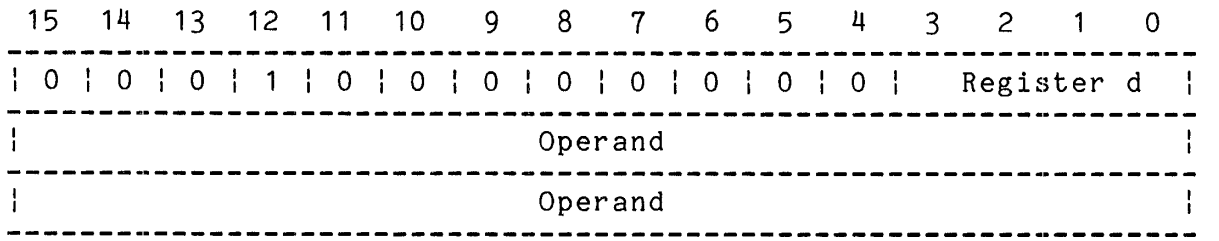
Definition:
Compare Long Words

Format:

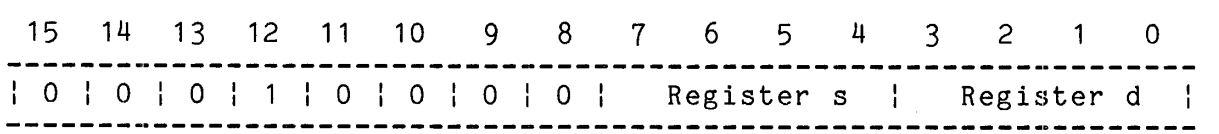
R
90rr



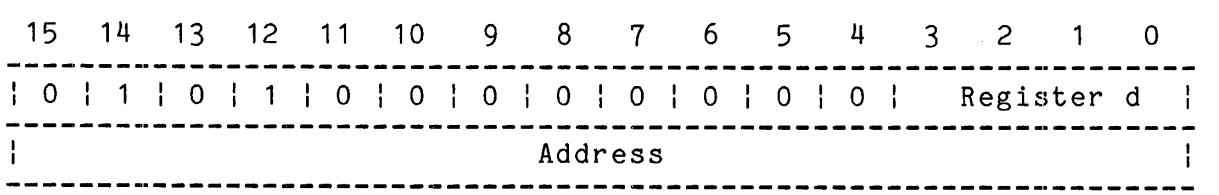
IM
100r



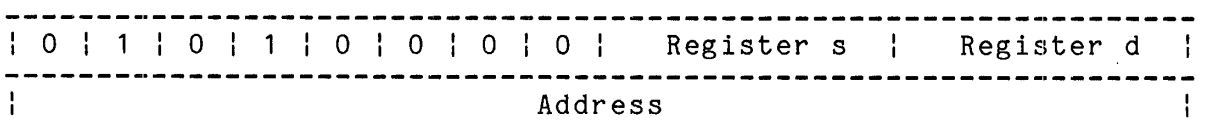
IR
10rr



DA
500r



X
50rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

If Register d<0:31> = source, set flags
If Register d<0:31> ≠ source, set flags

Flags Affected:
C reset on carry from MSB of result, set otherwise
Z set if result = 0 (compare is true), reset otherwise
S set if result is negative, reset otherwise
P/V set on arithmetic overflow, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	8
IM - S,NS	14
IR - S,NS	14
DA - NS	15
DA - SSO	16
DA - SLO	18
X - NS	16
X - SSO	16
X - SLO	19

Description:
The CPL instruction compares (by subtraction) the source long word with the contents of Register 'd'. The result of the operation is reflected in the flags. The CP operation does not affect the contents of either the source or Register 'd'.

Example:

```
CPL     RQ4,<<%7A>>%13F0  !Compare the contents of RQ4
                          !with the value contained in
                          !segment 7A, location 13F0.
                          !Reset the C flag if a carry
                          !from the most significant bit
                          !of the result occured.  Set the
                          !Z flag if the compare is true,
                          !reset otherwise.
                          !(DA mode, Long Offset)
```

OR
(Normal)

Definition:
OR Words

Format:

R 85rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	0	1	Register s				Register d			

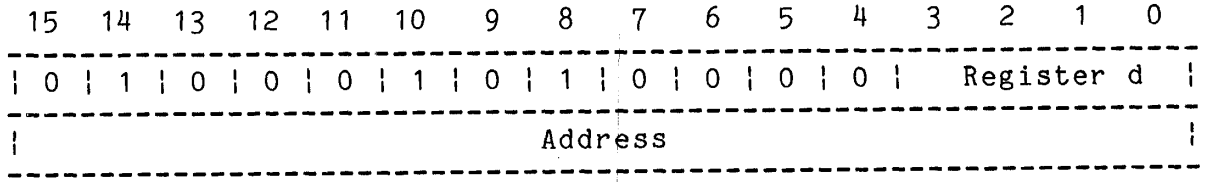
IM 050r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	0	0	Register d			
Operand															

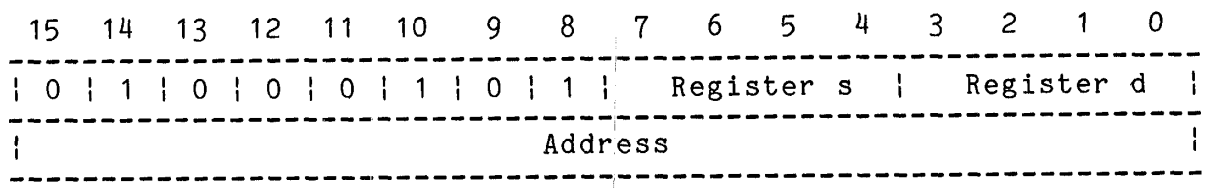
IR 05rr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	Register s				Register d			

DA 450r



X 45rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Register d<0:15> <-- source<0:15> + Register d<0:15>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The OR instruction logically ORs the contents of Register 'd' and the source operand. The result is placed in Register 'd'.

Example:

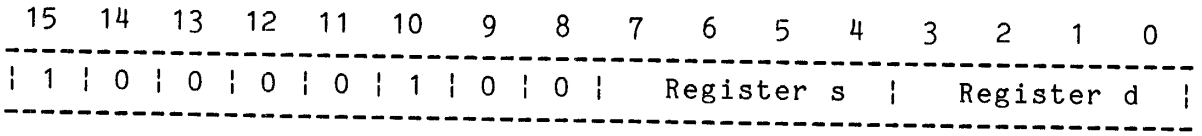
```
OR      R2,R4   !OR the contents of R2 with the
                !contents of R4, and place the
                !result in R2.
                !(R Mode)
```

ORB
(Normal)

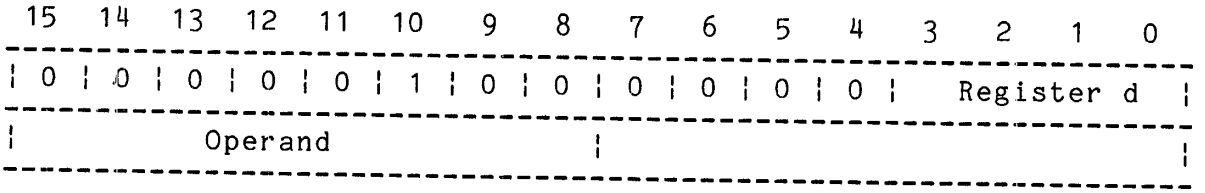
Definition:
OR Bytes

Format:

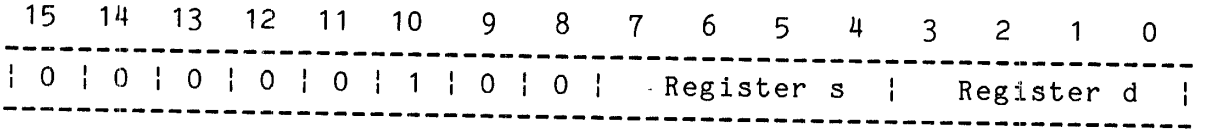
R
84rr



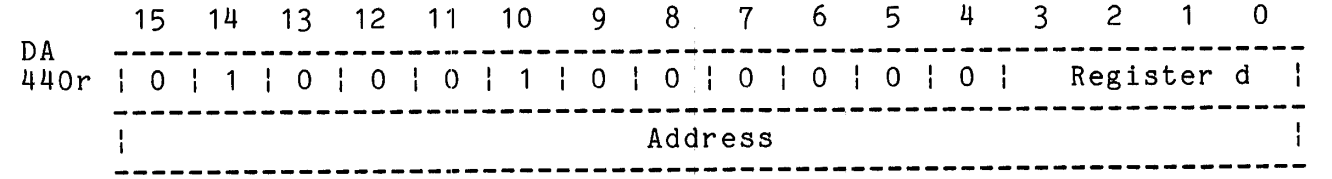
IM
040r



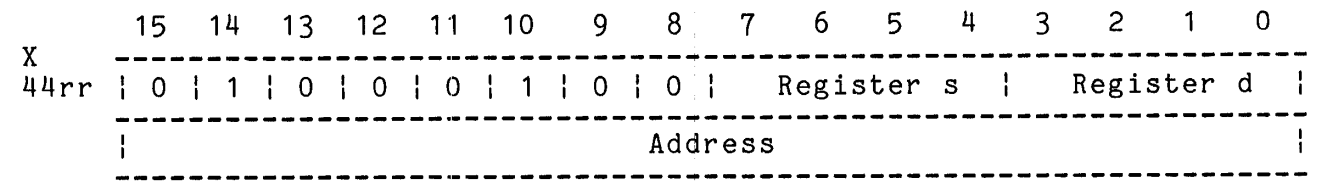
IR
04rr



DA



X



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Z8001/2

Register d<0:7> <-- source<0:7> + Register d<0:7>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise
P/V set if parity of result is even, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SSO	10
X - SLO	13

Description:
The ORB instruction logically ORs the contents of Register 'd' and the source operand. The result is placed in Register 'd'.

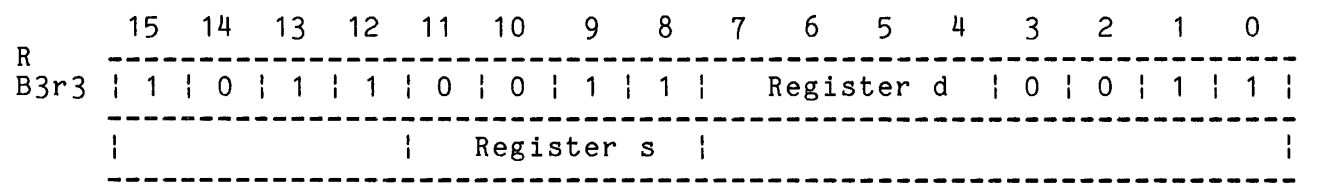
Example:

```
    ORB     RL2,RH4     !OR the contents of RL2 with the
                        !contents of RH4, and place the
                        !result in RL2.
                        !(R Mode)
```

SDL
(Normal)

Definition:
Logically Shift Word

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register d<0:15> shifted

Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if result negative, reset otherwise
P/V undefined

Clock Cycles = 15 + 3n*
* n = number of places shifted

Description:
The SDL instructions shifts the contents of Register 'd' either left or right, according to the contents of Register 's'. Register 's' contains a 16-bit **signed** 2's complement integer, ranging from -16 to +16. A negative number indicates that the contents of Register 'd' are shifted right, while a positive integer indicates a left shift.

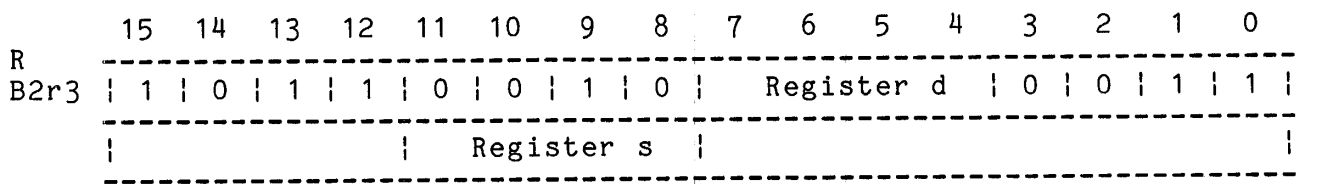
Example:

```
    SDL     R2,R4           !Shift the contents of R2 either
                            !left or right, depending on the
                            !signed value of R4.
```

SDLB
(Normal)

Definition:
Logically Shift Byte

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register d<0:7> shifted

Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if result negative, reset otherwise
P/V undefined

Clock Cycles = 15 + 3n*
* n = number of places shifted

Description:
The SDLB instructions shifts the contents of Register 'd' either left or right, according to the contents of Register 's'. Register 's' contains a 16-bit **signed** 2's complement integer, ranging from -8 to +8. A negative number indicates that the contents of Register 'd' are shifted right, while a positive integer indicates a left shift.

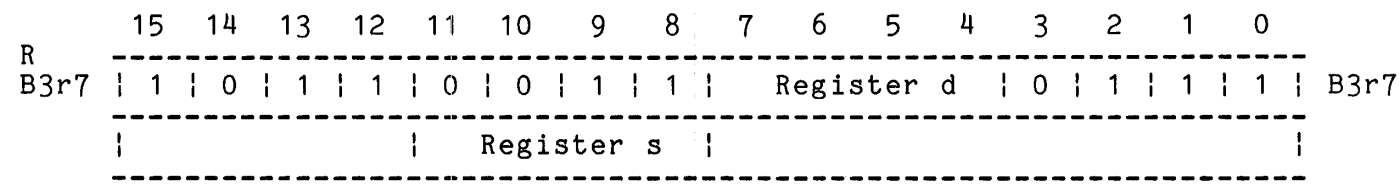
Example:

```
    SDLB    RL2,R4          !Shift the contents of RL2 either
                            !left or right, depending on the
                            !signed value of R4.
```

SDLL
(Normal)

Definition:
Logically Shift Long Word

Format:



Operation:
Z8001/2

Register d<0:31> <-- Register d<0:31> shifted

Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if result negative, reset otherwise
P/V undefined

Clock Cycles = 15 + 3n*
* n = number of places shifted

Description:
The SDLL instructions shifts the contents of Register 'd' either left or right, according to the contents of Register 's'. Register 's' contains a 16-bit **signed** 2's complement integer, ranging from -32 to +32. A negative number indicates that the contents of Register 'd' are shifted right, while a positive integer indicates a left shift.

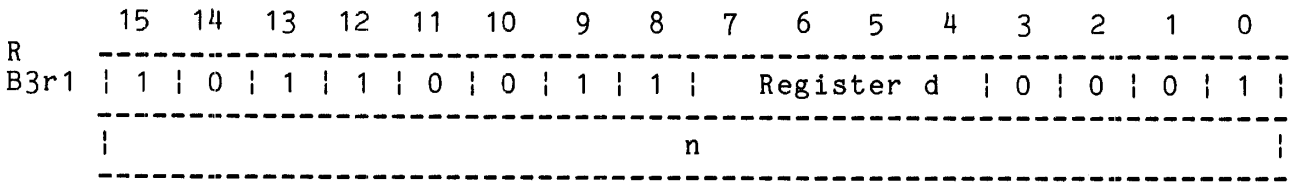
Example:

```
    SDLL    RQ4,R0          !Shift the contents of RL2 either
                            !left or right, depending on the
                            !signed value of R0.
```

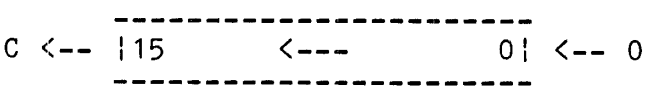
SLL
(Normal)

Definition:
Logically Left Shift Word

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if MSB of Register d = 1, reset otherwise
P/V undefined

Clock Cycles = 13 + 3n*
*n = number of places shifted

Description:
The SLL instruction shifts the contents of Register 'd' to the left "n" places. The "n" operand is a 16-bit positive 2's complement integer, ranging from 0 through 16. If "n" = 0, register contents are shifted one place.

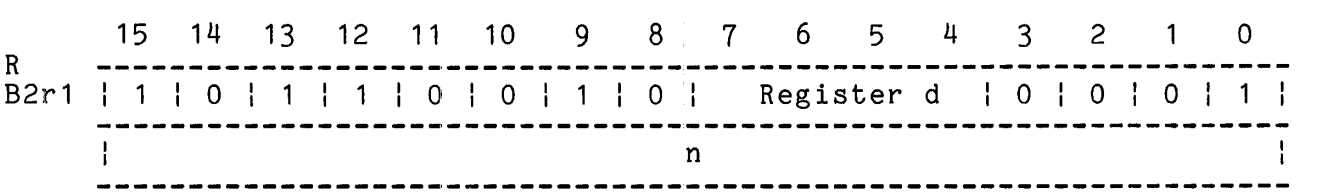
Example:

```
SLL     R2,#2           !Logically shift the contents
                        !of R2 two places to the left.
```

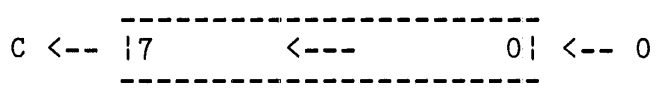
SLLB
(Normal)

Definition:
Logically Left Shift Byte

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if MSB of Register d = 1, reset otherwise
P/V undefined

Clock Cycles = 13 + 3n*
*n = number of places shifted

Description:
The SLLB instruction shifts the contents of Register 'd' to the left "n" places. The "n" operand is a 16-bit positive 2's complement integer, ranging from 0 through 8. If "n" = 0, register contents are shifted one place.

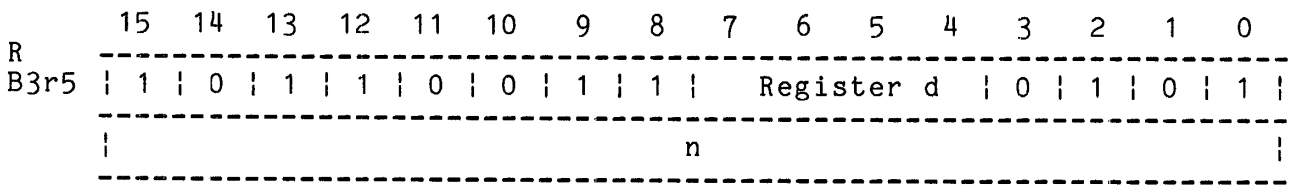
Example:

```
SLLB    RL2,#2          !Logically shift the contents
                        !of RL2 two places to the left.
```

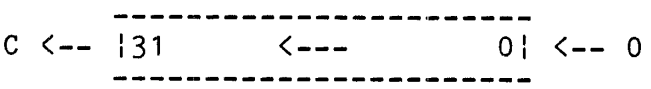
SLLL
(Normal)

Definition:
Logically Left Shift Long Word

Format:



Operation:



Flags Affected:
C loaded from last bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if MSB of Register d = 1, reset otherwise
P/V undefined

Clock Cycles = 13 + 3n*
*n = number of places shifted

Description:
The SLLL instruction shifts the contents of Register 'd' to the left "n" places. The "n" operand is a 16-bit positive 2's complement integer, ranging from 0 through 31. If "n" = 0, register contents are shifted one place.

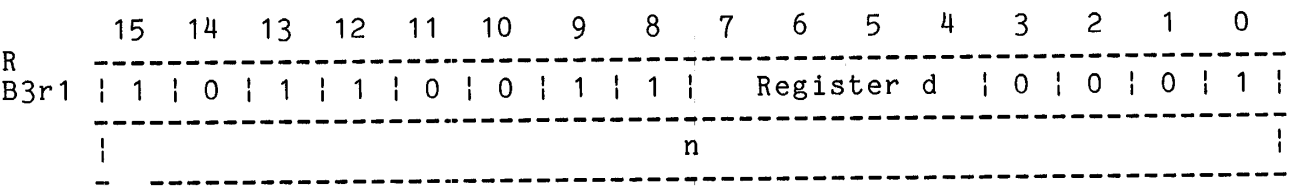
Example:

```
SLLL    RQ4,#2          !Logically shift the contents
                        !of RQ4 two places to the left.
```

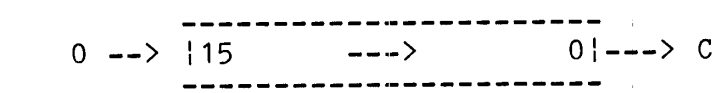
SRL
(Normal)

Definition:
Logically Right Shift Word

Format:



Operation:



Flags Affected:
C loaded from least significant bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if MSB of Register d = 1, reset otherwise
P/V undefined

Clock Cycles = 13 + 3n*
*n = number of places shifted

Description:
The SRL instruction shifts the contents of Register 'd' to the right "n" places. The "n" operand is a 16-bit negative 2's complement integer, ranging from 0 through 16. If "n" = 0, register contents are shifted one place.

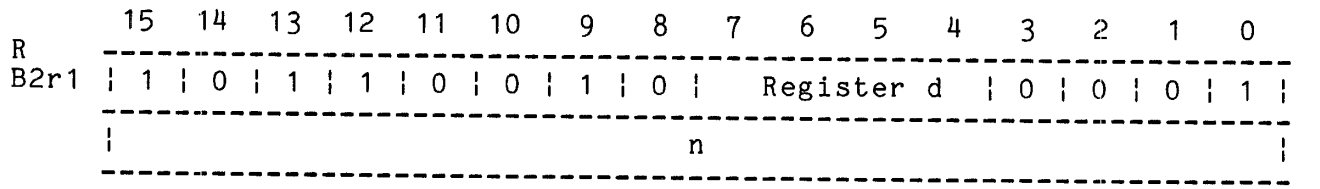
Example:

```
    SRL     R2,#2           !Logically shift the contents
                            !of R2 two places to the right.
```

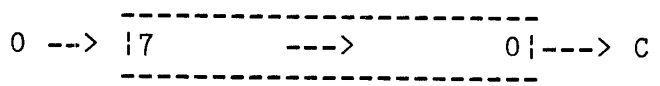
SRLB
(Normal)

Definition:
Logically Right Shift Byte

Format:



Operation:



Flags Affected:
C loaded from least significant bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if MSB of Register d = 1, reset otherwise
P/V undefined

Clock Cycles = 13 + 3n*
*n = number of places shifted

Description:
The SRLB instruction shifts the contents of Register 'd' to the right "n" places. The "n" operand is a 16-bit negative 2's complement integer, ranging from 0 through 7. If "n" = 0, register contents are shifted one place.

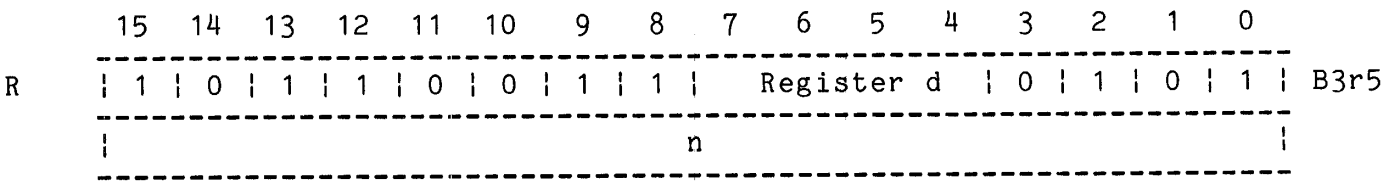
Example:

```
    SRLB    RL2,#2          !Logically shift the contents
                            !of RL2 two places to the right.
```

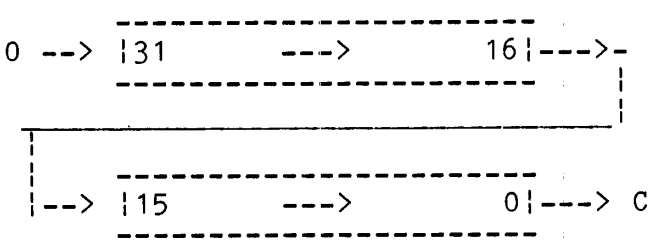
SRLL
(Normal)

Definition:
Logically Right Shift Long Word

Format:



Operation:



Flags Affected:
C loaded from least significant bit shifted out of Register d
Z set if result = 0, reset otherwise
S set if MSB of Register d = 1, reset otherwise
P/V undefined

Clock Cycles = 13 + 3n*
*n = number of places shifted

Description:
The SRLL instruction shifts the contents of Register 'd' to the right "n" places. The "n" operand is a 16-bit negative 2's complement integer, ranging from 0 through 31. If "n" = 0, register contents are shifted one place.

Example:

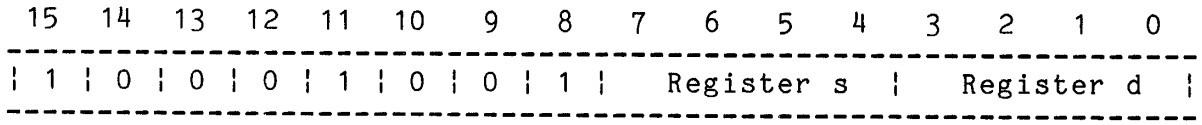
```
    SRLL    RQ4,#2          !Logically shift the contents
                            !of RQ4 two places to the right.
```

XOR
(Normal)

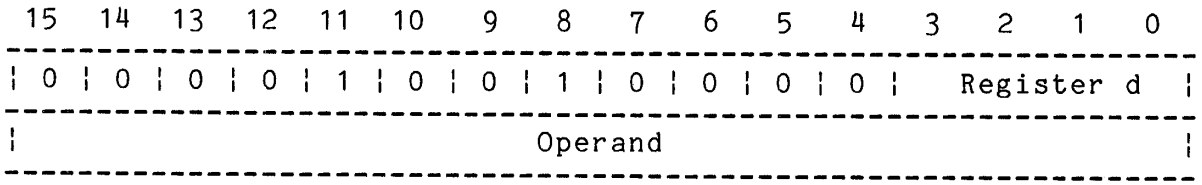
Definition:
Exclusive-OR Words

Format:

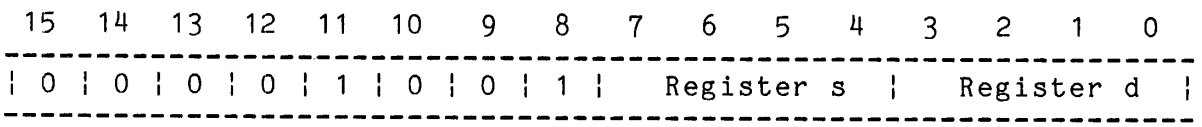
R
89rr



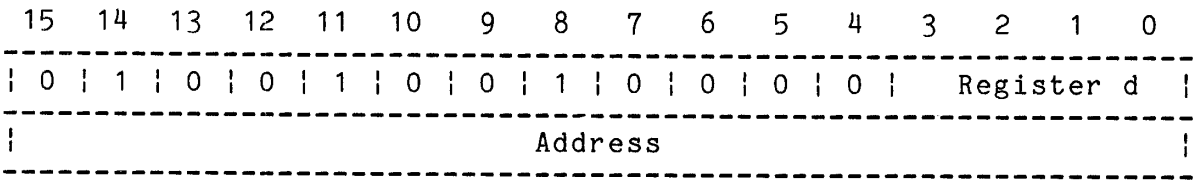
IM
090r



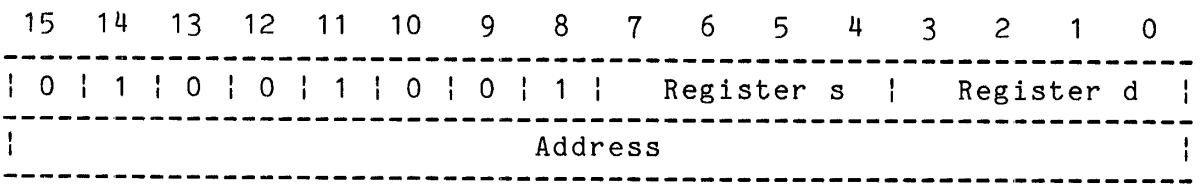
IR
09rr



DA
490r



X
49rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Register d<0:15> <-- source<0:15> ⊕ Register d<0:15>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The XOR instruction performs an exclusive-OR operation between Register 'd' and the source operand. The 16-bit result is loaded into Register 'd'.

Example:

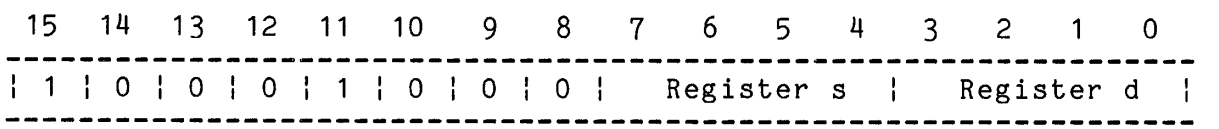
```
    XOR     R2,R4           !Exclusive-OR the contents
                            !of R2 and R4, and place
                            !the result in R2.
```

XORB
(Normal)

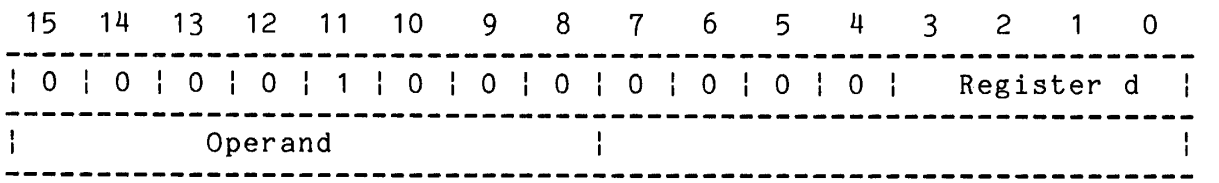
Definition:
Exclusive-OR Bytes

Format:

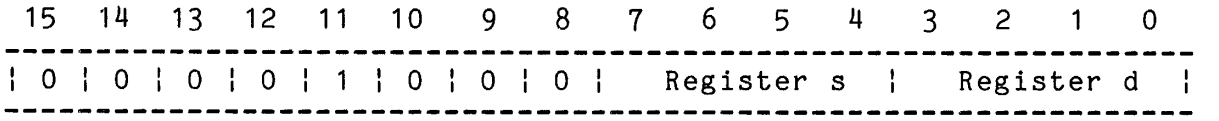
R
88rr



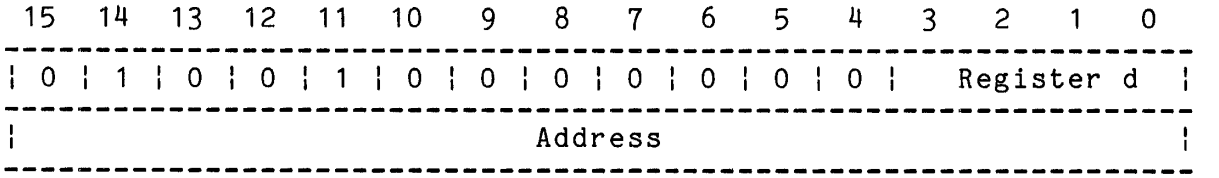
IM
080r



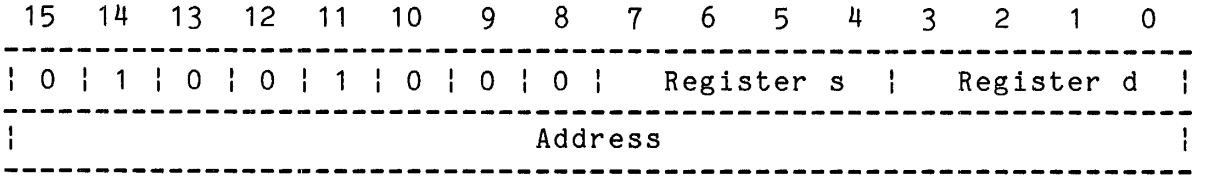
IR
08rr



DA
480r



X
48rr



Addressing Modes:
R - S,NS
IM - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
Register d<0:7> <-- source<0:7> ⊕ Register d<0:7>

Flags Affected:
Z set if result = 0, reset otherwise
S set if result is negative, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
IM - S,NS	7
IR - S,NS	7
DA - NS	9
DA - SSO	10
DA - SLO	12
X - NS	10
X - SSO	10
X - SLO	13

Description:
The XORB instruction performs an exclusive-OR operation between Register 'd' and the source operand. The 8-bit result is loaded into Register 'd'.

Example:

```
    XORB    RL2,RL4         !Exclusive-OR the contents
                            !of RL2 and RL4, and place
                            !the result in RL2.
```

The Bit Manipulation Group

The Bit Manipulation Group consists of the following instructions:

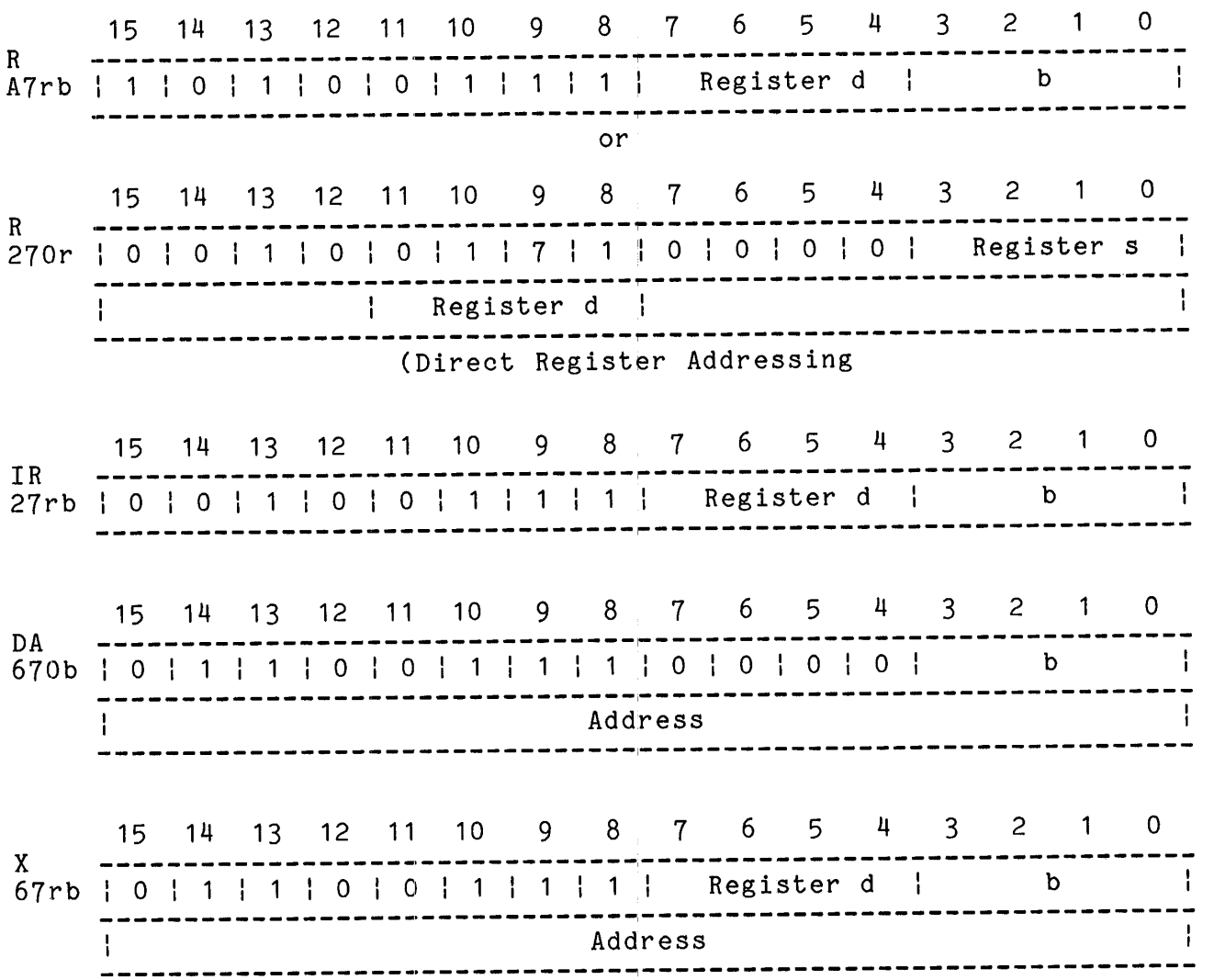
Table 2-10
Bit Manipulation Group

Mnemonic	Description
BIT	Test a Bit in a Word
BITB	Test a Bit in a Byte
RES	Reset a Bit in a Word
RESB	Reset a Bit in a Byte
SET	Set a Bit in a Word
SETB	Set a Bit in a Byte

BIT
(Normal)

Definition:
Test a Bit in a Word

Format:



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected:
Z set if specified bit of word = 0, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
R* - S,NS	10
IR - S,NS	8
DA - NS	10
DA - SSO	11
DA - SLO	13
X - NS	11
X - SSO	11
X - SLO	14

* If direct register addressing is used.

Description:
The BIT instruction tests a single bit in a 16-bit destination word. You'll note in the Format that there are two methods of using the R addressing mode. In the first method, the particular bit to be tested is identified in the "b" field of the instruction. In the second method, the bit to be tested is identified in the Register 's' field of the instruction, while the destination register is identified in a second instruction word.

The bit to be tested is identified in a binary manner, and the destination is selected by the appropriate addressing mode. The Z flag is set if bit = 0, reset if bit = 1.

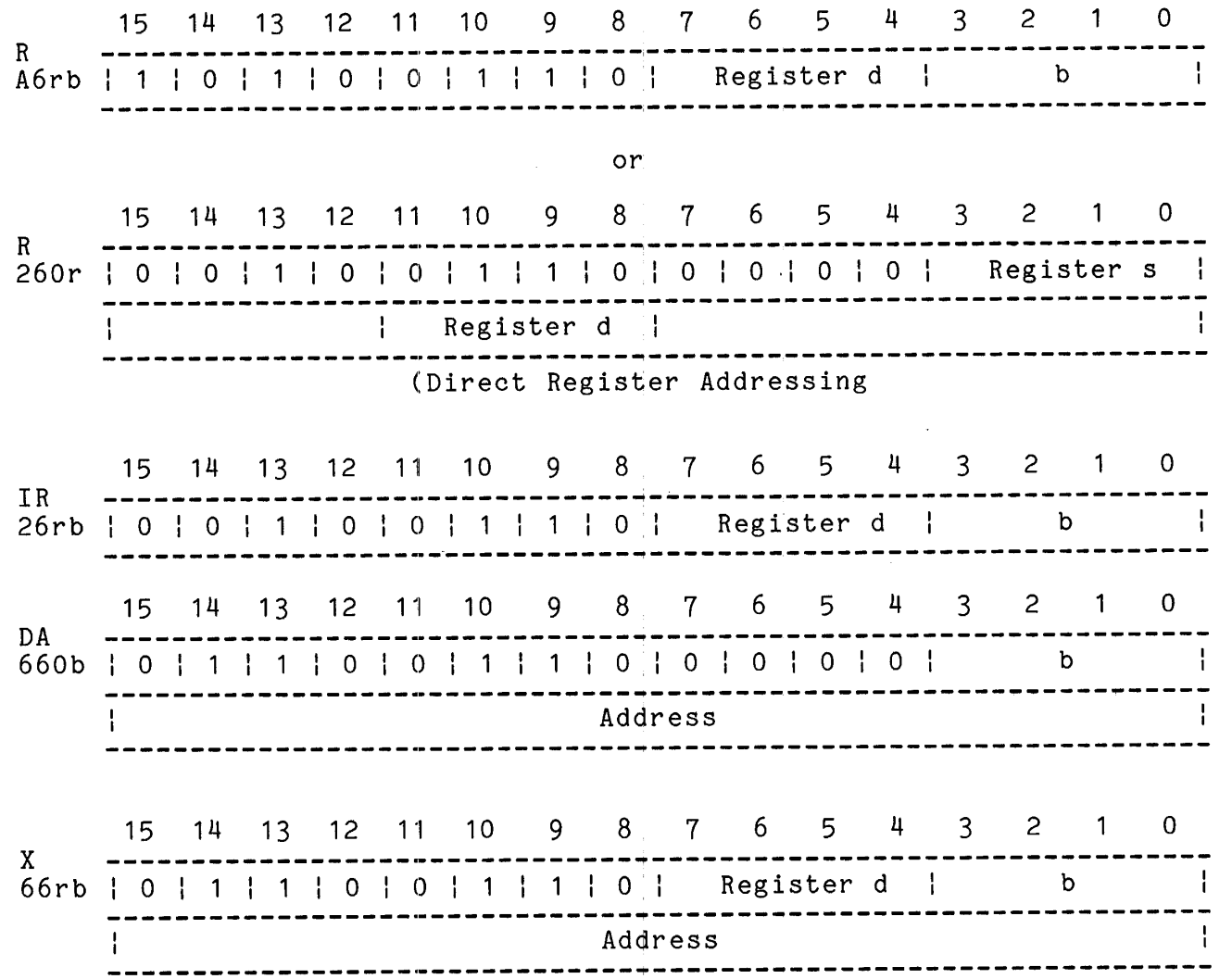
Example:

```
BIT     %30FF,#5        !Test bit 5 of location 30FF
                        !to 1.

BIT     R4,R2           !Test the bit (identified by
                        !bits 0-3 within R2) of R4.
```

BITB
(Normal)

Definition:
Test a Bit in a Byte

Format:



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected:
Z set if bit tested = 0, reset otherwise

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
R* - S,NS	10
IR - S,NS	8
DA - NS	10
DA - SSO	11
DA - SLO	13
X - NS	11
X - SSO	11
X - SLO	14

* If direct register addressing is used.

Description:
The BITB instruction tests a single bit in a 8-bit destination byte. You'll note in the Format that there are two methods of using the R addressing mode. In the first method, the particular bit to be tested is identified in the "b" field of the instruction. In the second method, the bit to be tested is identified in the Register 's' field of the instruction, while the destination register is identified in a second instruction word.

The bit to be tested is identified in a binary manner, and the destination is selected by the appropriate addressing mode. If the bit tested = 0, the Z flag is set. If the bit test = 1, the Z flag is reset.

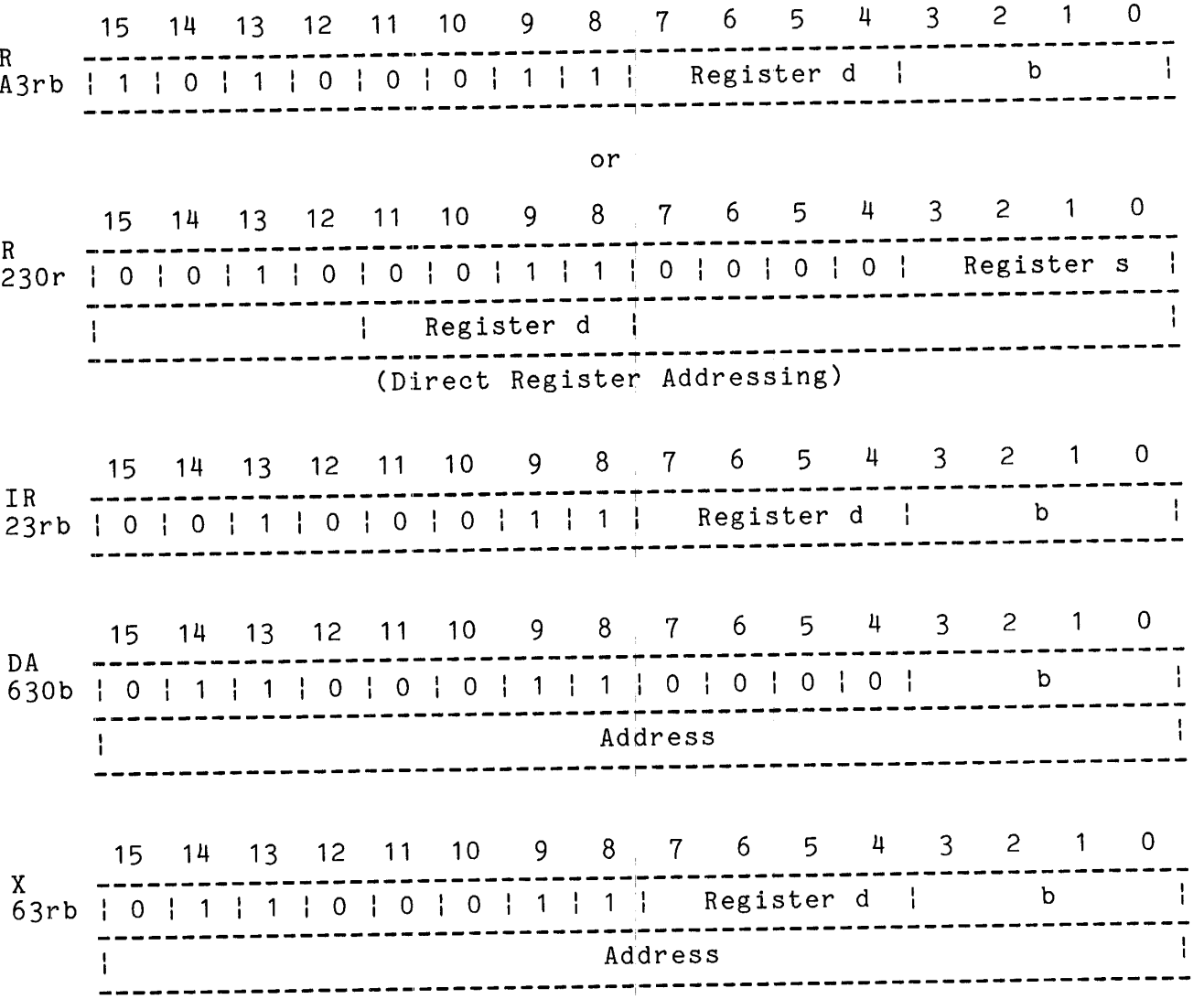
Example:

```
BITB    %30FF,#5        !Test bit 5 of location 30FF

BITB    RH4,R2          !Test the bit (identified by
                        !bits 0-3 within R2) of
                        !register RH4.
```

RES
(Normal)

Definition:
Reset a Bit in a Word

Format:



Addressing Modes
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
R* - S,NS	10*
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

* If direct register addressing is used

Description:
The RES instruction resets a single bit in a 16-bit destination word. You'll note in the Format, that there are two methods of using the R addressing mode. In the first method, the particular bit to be set is identified in the "b" field of the instruction. In the second method, the bit to be reset is identified in the Register 's' field of the instruction, while the destination register is identified in a second instruction word.

The bit to be reset is identified in a binary manner, and the destination is selected by the appropriate addressing mode.

Example:

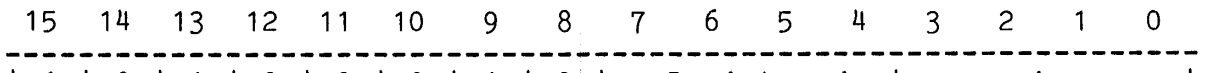
```
RES     %30FF,#5        !Reset bit 5 of location 30FF
                        !to 0.

RES     R4,R2           !Reset a bit (identified
                        !by bits 0-3 of R2) within
                        !register R4.
```

RESB
(Normal)

Definition:
Reset a Bit in a Byte

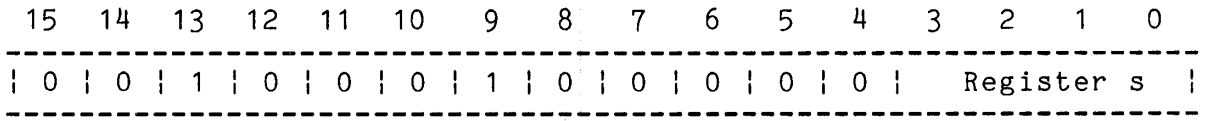
Format:

R
A2rb



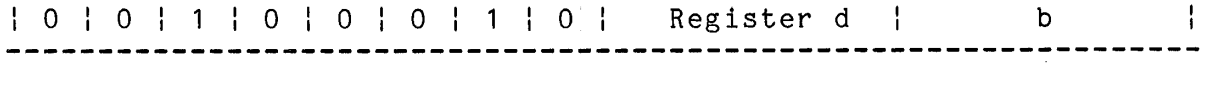
or

R
220r

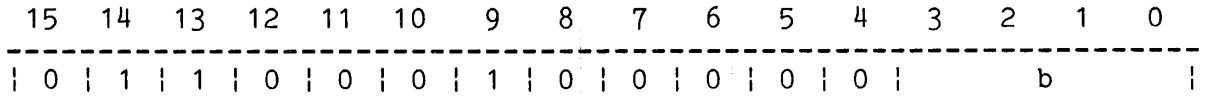


(Direct Register Addressing)

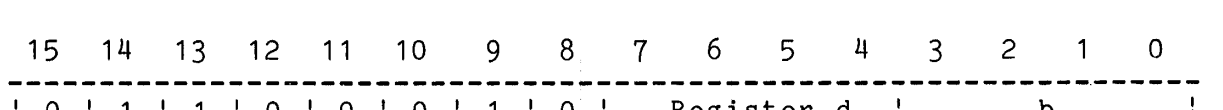
IR
22rb



DA
620b



X
62rb



Addressing Modes
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
R* - S,NS	10*
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

* If direct register addressing is used

Description:
The RESB instruction resets a single bit in a 8-bit destination byte. You'll note in the Format, that there are two methods of using the R addressing mode. In the first method, the particular bit to be set is identified in the "b" field of the instruction. In the second method, the bit to be reset is identified in the Register 's' field of the instruction, while the destination register is identified in a second instruction word. The source register is always a 16-bit word register.

The bit to be reset is identified in a binary manner, and the destination is selected by the appropriate addressing mode.

Example:

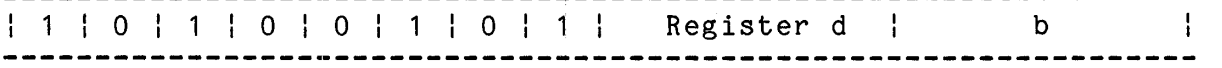
```
    RESB    %30FF,#5        !Reset bit 5 of location 30FF
                            !to 0.

    RESB    RH4,R2          !Reset a bit (identified
                            !by bits 0-3 of R2) within
                            !register RH4.
```

SET
(Normal)

Definition:
Set a Bit in a Word

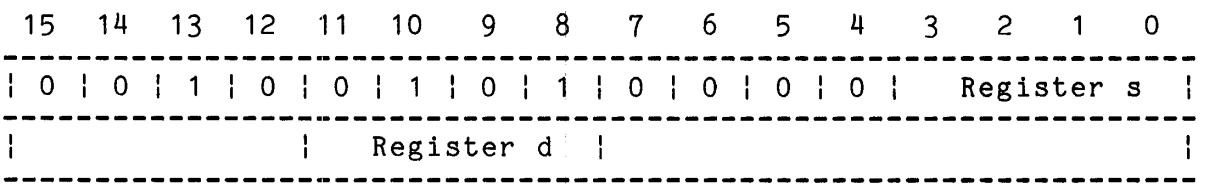
Format:

R A5rb



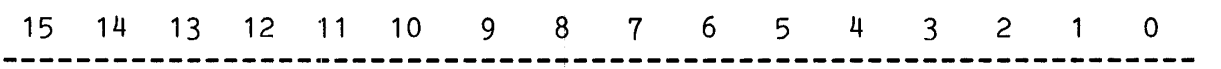
or

R 250r

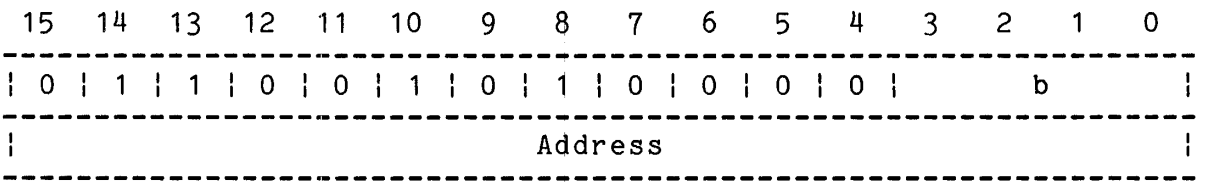


(Direct Register Addressing

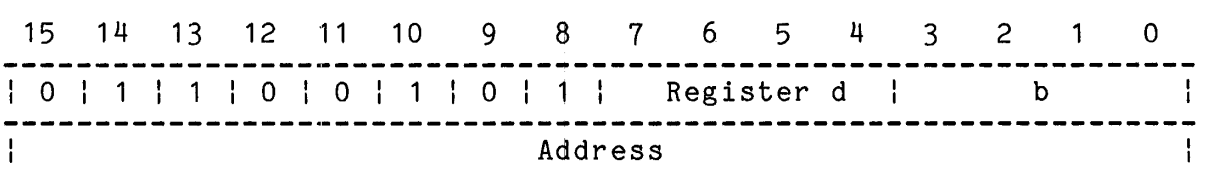
IR 25rb



DA 650b



X 65rb



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
R* - S,NS	10
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

* If direct register addressing is used.

Description:
The SET instruction sets a single bit in a 16-bit destination word. You'll note in the Format that there are two methods of using the R addressing mode. In the first method, the particular bit to be set is identified in the "b" field of the instruction. In the second method, the bit to be set is identified in the Register 's' field of the instruction, while the destination register is identified in a second instruction word.

The bit to be set is identified in a binary manner, and the destination is selected by the appropriate addressing mode.

Example:

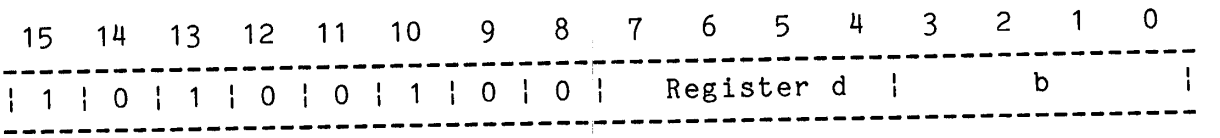
```
    SET     %30FF,#5        !Set bit 5 of location 30FF
                            !to 1.

    SET     R4,R2           !Set the bit (identified by
                            !bits 0-3 within R2) of
                            !register R4.
```

SETB
(Normal)

Definition:
Set a Bit in a Byte

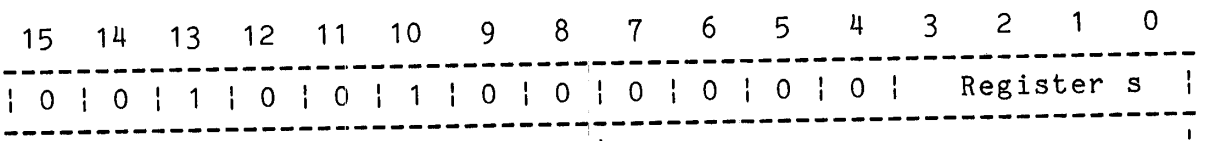
Format:

R
A4rb



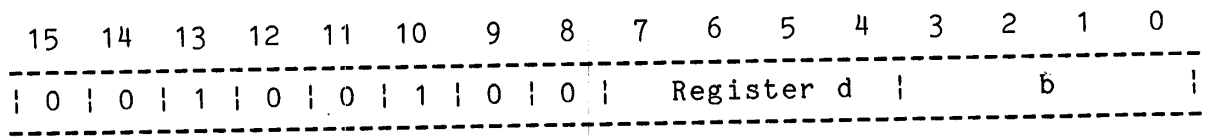
or

R
240r

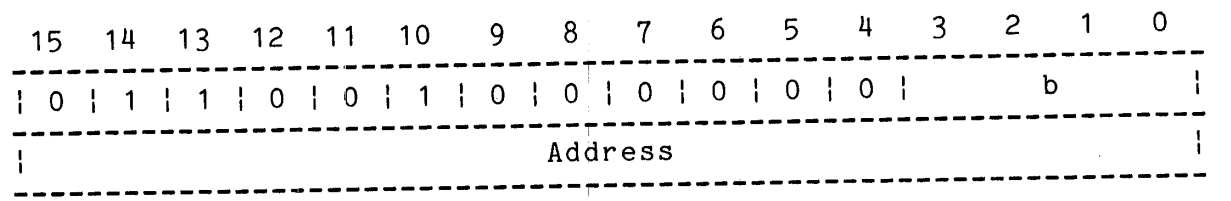


(Direct Register Addressing

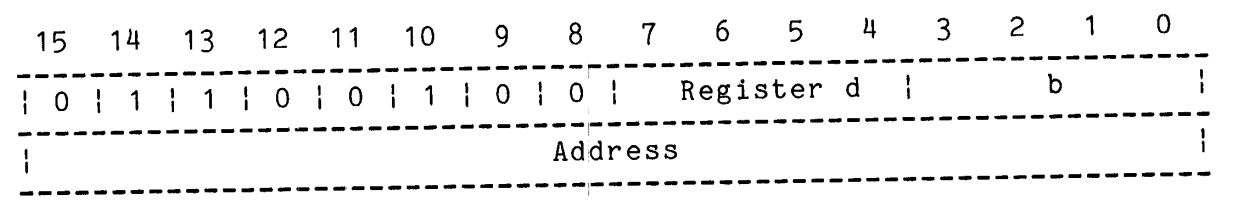
IR
24rb



DA
640b



X
64rb



Addressing Modes:
R - S,NS
IR - S,NS
DA - NS,SSO,SLO
X - NS,SSO,SLO

Operation:
See Description.

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
R - S,NS	4
R* - S,NS	10
IR - S,NS	11
DA - NS	13
DA - SSO	14
DA - SLO	16
X - NS	14
X - SSO	14
X - SLO	17

* If direct register addressing is used.

Description:
The SET instruction sets a single bit in a 8-bit destination byte. You'll note in the Format that there are two methods of using the R addressing mode. In the first method, the particular bit to be set is identified in the "b" field of the instruction. In the second method, the bit to be set is identified in the Register 's' field of the instruction, while the destination register is identified in a second instruction word.

The bit to be set is identified in a binary manner, and the destination is selected by the appropriate addressing mode.

Example:

```
SETB    %30FF,#5        !Set bit 5 of location 30FF
                        !to 1.

SETB    RH4,R2          !Set the bit (identified by
                        !bits 0-3 within R2) of
                        !register RH4.
```

Block Transfer and String Manipulation Group

The Block Transfer and String Manipulation Group consists of the following instructions:

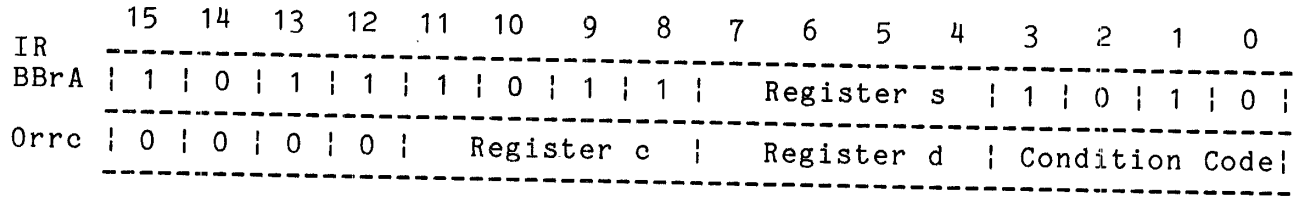
Table 2-11
Block Transfer and String Manipulation Group

Mnemonic	Description
CPSD	Compare Word Strings and Decrement
CPSDB	Compare Byte Strings and Decrement
CPSDR	Compare Word Strings, Decrement, and Repeat
CPSDRB	Compare Byte Strings, Decrement, and Repeat
CPSI	Compare Word Strings and Increment
CPSIB	Compare Byte Strings and Increment
CPSIR	Compare Word Strings, Increment, and Repeat
CPSIRB	Compare Byte Strings, Increment, and Repeat
TRDB	Translate Byte and Decrement
TRDRB	Translate Byte, Decrement, and Repeat
TRIB	Translate Byte and Increment
TRIRB	Translate Byte, Increment, and Repeat
TRTDB	Translate Byte, Test, and Decrement
TRTDRB	Translate Byte, Test, Decrement, and Repeat
TRTIB	Translate Byte, Test, and Increment
TRTIRB	Translate Byte, Test, Increment, and Repeat

CPSD (Normal)

Definition:
Compare Word Strings and Decrement

Format:



Operation:
Z8001/2

If result of Destination<0:15> - Source<0:15> meets
Condition Code:
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> - 2
Register d<0:15> <-- Register d<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
Z set if comparison results in flags meeting Condition Code specified, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The CPSD instruction compares (by subtraction) the contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set. Register 'd' and Register 's' are both decremented by two, and Register 'c' is decremented by one. If Register 'c' = 0 at completion of the instruction, the P/V flag is set.

The CPSD instruction is typically used to compare strings contained in memory for a specific bit pattern. All registers are 16-bits long.

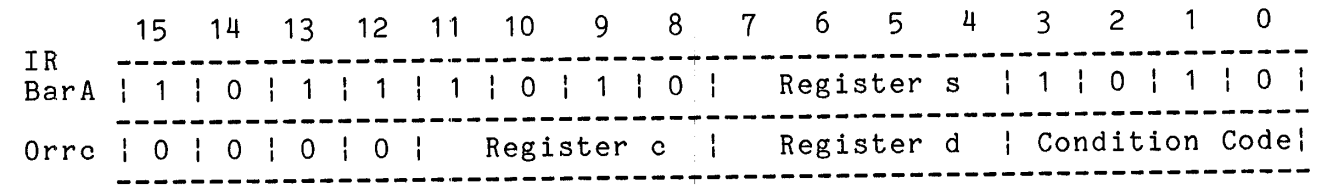
Example:

```
CPSD    @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count.  If the
                        !condition code NE is met, set
                        !the Z flag.  Decrement R2 and
                        !R4 by 2, R0 by 1.
```

CPSDB (Normal)

Definition:
Compare Byte Strings and Decrement

Format:



Operation:
Z8001/2

```
-------
If result of Destination<0:7> - Source(0:7> meets
Condition Code:
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> - 1
Register d<0:15> <-- Register d<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1
```

Flags Affected:
Z set if comparison results in flags meeting Condition
  Code specified, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The CPSDB instruction compares (by subtraction) the byte contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set. Register 'd' and Register 's' are both decremented by one, and Register 'c' is decremented by one. If Register 'c' = 0 at completion of the instruction, the P/V flag is set.

The CPSDB instruction is typically used to compare byte strings contained in memory for a specific bit pattern. All registers are 16-bits long.

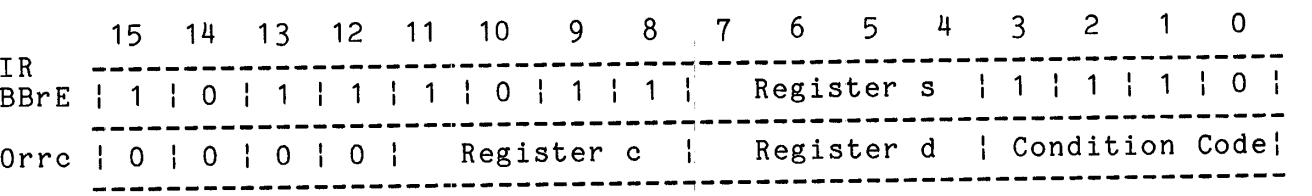
Example:

```
CPSDB   @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count. If the
                        !condition code NE is met, set
                        !the Z flag. Decrement R2, R4,
                        !and R0 by 1.
```

CPSDR (Normal)

Definition:
Compare Word Strings, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
If result of Destination<0:15> - Source<0:15> meets
   Condition Code:
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> - 2
Register d<0:15> <-- Register d<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c<0:15> = 0
```

Flags Affected:
Z set if comparison results in flags meeting Condition Code specified, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 11 + 14n *
* n = number of iterations

Description:
The CPSDR instruction is the same as the SPSD instruction, except that the compare operation is repeated automatically until Register 'c' = 0, or the condition of the Condition Code field is met.

The CPSDR instruction compares (by subtraction) the word contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set, and the instruction is terminated. If the condition code is not met, Register 'd' and Register 's' are both decremented by two, and Register 'c' is decremented by one. If Register 'c' = 0 at completion of

the instruction, the P/V flag is set and the instruction is terminated.

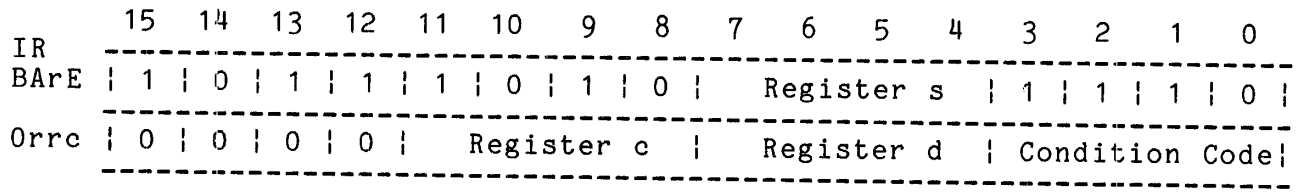
Example:

```
CPSDR   @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count.  If the
                        !specified condition code (NE)
                        !is met, or if R0 = 0, terminate
                        !the instruction.  Otherwise,
                        !decrement R2 and R4 by 2, R0 by 1,
                        !and repeat.
```

CPSDRB
(Normal)

Definition:
Compare Byte Strings, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
If result of Destiantion<0:7> - Source<0:7> meets
   Condition Code:
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> - 1
Register d<0:15> <-- Register d<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c<0:15> = 0
```

Flags Affected:
Z set if comparison results in flags meeting Condition Code specified, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 11 + 14n *
* n = number of iterations

Description:
The CPSDRB instruction is the same as the CPSDB instruction, except that the compare operation is repeated automatically until Register 'c' = 0, or the condition of the Condition Code field is met.

The CPSDRB instruction compares (by subtraction) the byte contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set, and the instruction is terminated. If the condition code is not met, Register 'd' and Register 's' are both decremented by one, and Register 'c' is decremented by one. If Register 'c' = 0 at completion of the instruction, the P/V flag is set and the instruction is terminated.

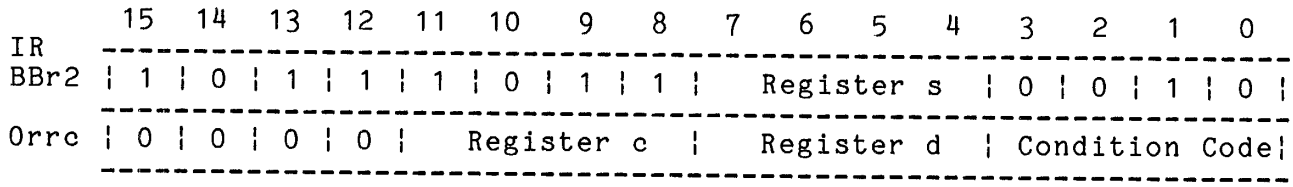
Example:

```
CPSDRB  @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count.  If the
                        !specified condition code (NE)
                        !is met, or if R0 = 0, terminate
                        !the instruction.  Otherwise,
                        !decrement R2, R4, and R0 by 1,
                        !and repeat.
```

CPSI
(Normal)

Definition:
Compare Word Strings and Increment

Format:



Operation:
Z8001/2

If result of Destination<0:15> - Source<0:15>
meets Condition Code:
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> + 2
Register d<0:15> <-- Register d<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
Z set if comparison results in flags meeting Condition Code specified, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The CPSI instruction compares (by subtraction) the contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set. Register 'd' and Register 's' are both incremented by two, and Register 'c' is decremented by one. If Register 'c' = 0 at completion of the instruction, the P/V flag is set.

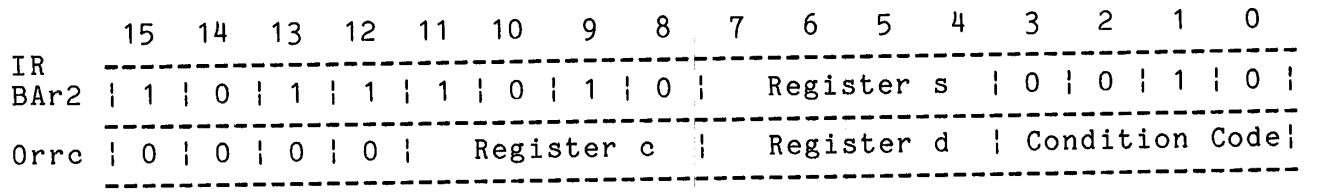
Example:

```
CPSI    @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count.  If the
                        !condition code NE is met, set
                        !the Z flag.  Increment R2 and
                        !R4 by 2, and decrement R0 by 1.
```

CPSIB
(Normal)

Definition:
Compare Byte Strings and Increment

Format:



Operation:
Z8001/2

```
If result of Destination<0:7> - Source(0:7> meets
Condition Code:
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> + 1
Register d<0:15> <-- Register d<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1
```

Flags Affected:
Z set if comparison results in flags meeting Condition Code
  Code specified, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The CPSIB instruction compares (by subtraction) the byte contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set. Register 'd' and Register 's' are both incremented by one, and Register 'c' is decremented by one. If Register 'c' = 0 at completion of the instruction, the P/V flag is set.

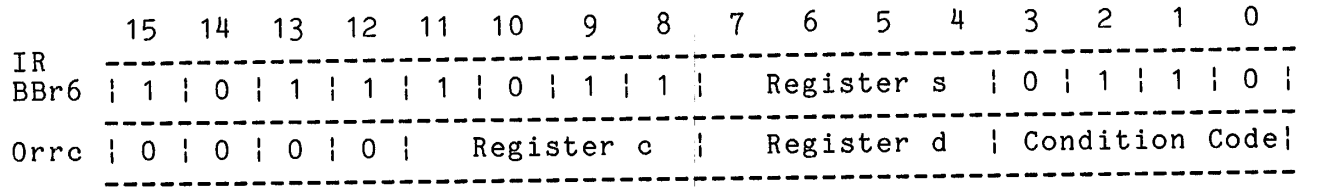
Example:

```
CPSIB   @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count. If the
                        !condition code NE is met, set
                        !the Z flag. Increment R2 and R4,
                        !and decrement R0 by 1.
```

CPSIR
(Normal)

Definition:
Compare Word Strings, Increment, and Repeat

Format:



Operation:
Z8001/2

If result of Destination<0:15> - Source<0:15> meets
Condition Code:
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> + 2
Register d<0:15> <-- Register d<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c<0:15> = 0

Flags Affected:
Z set if comparison results in flags meeting Condition
Code specified, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 11 + 14n *
* n = number of iterations

Description:
The CPSIR instruction is the same as the SPSD instruction, except that the compare operation is repeated automatically until Register 'c' = 0, or the condition of the Condition Code field is met.

The CPSIR instruction compares (by subtraction) the word contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set, and the instruction is terminated. If the condition code is not met, Register 'd' and Register 's' are both incremented by two, and Register 'c'

is decremented by one. If Register 'c' = 0 at completion of the instruction, the P/V flag is set and the instruction is terminated.

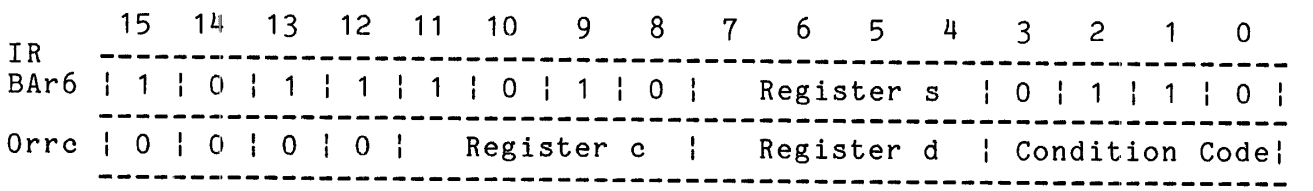
Example:

```
CPSIR   @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count.  If the
                        !specified condition code (NE)
                        !is met, or if R0 = 0, terminate
                        !the instruction.  Otherwise,
                        !increment R2 and R4 by 2, decrement
                        !R0 by 1, and repeat.
```

CPSIRB
(Normal)

Definition:
Compare Byte Strings, Increment, and Repeat

Format:



Operation:
Z8001/2

If result of Destination<0:7> - Source<0:7> meets Condition Code,
Z flag <-- 1
Register s<0:15> <-- Register s<0:15> + 1
Register d<0:15> <-- Register d<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c<0:15> = 0

Flags Affected:
Z set if comparison results in flags meeting Condition Code specified,
reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 11 + 14n *
* n = number of iterations

Description:
The CPSIRB instruction is the same as the CPSIB instruction, except that the compare operation is repeated automatically until Register 'c' = 0, or the condition of the Condition Code field is met.

The CPSIRB instruction compares (by subtraction) the byte contents of the memory locations pointed to by Register 'd' and Register 's'. If, at the end of the comparison, the flags are in the states specified in the instruction's Condition Code field, the Z flag is set, and the instruction is terminated. If the condition code is not met, Register 'd' and Register 's' are both incremented by one, and Register 'c' is decremented by one. If Register 'c' = 0 at completion of the instruction, the P/V flag is set and the instruction is terminated.

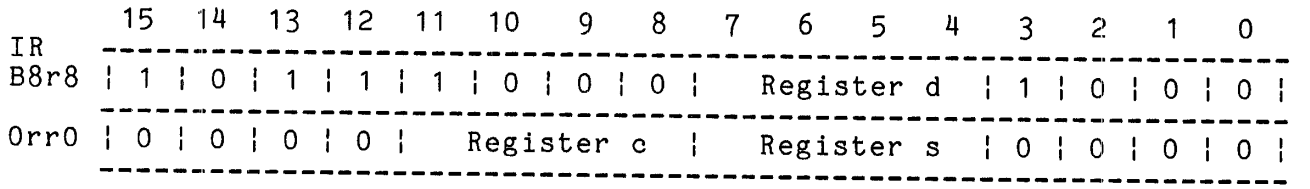
Example:

```
CPSIRB  @R2,@R4,R0,NE   !Compare the memory locations
                        !pointed to by R2 and R4.
                        !R0 contains the count.  If the
                        !specified condition code (NE)
                        !is met, or if R0 = 0, terminate
                        !the instruction.  Otherwise,
                        !increment R2 and R4, decrement
                        !R0 by 1, and repeat.
```

TRDB
(Normal)

Definition:
Translate Byte and Decrement

Format:



Operation:
See Description

Flags Affected:
Z undefined
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The TRDB instruction is used to translate a byte from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte within memory to be translated. Register 'c' contains a count of the number of bytes to be translated.

The TRDB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into the address pointed to by Register 'd'.

Once the original byte is replaced by the translated byte, Register 'd' is decremented by 1, as is the count register, Register 'c'. The TRDB instruction terminates after one iteration. The TRDRB instruction continues until Register 'c' = 0.

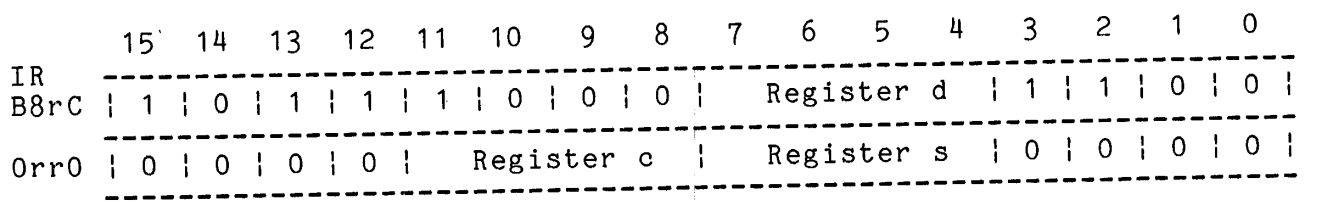
Example:

```
TRDB    @R2,@R4,R0      !Translate the byte pointed
                        !to by R2 according to the
                        !table based at R4.  Decrement
                        !R2 and R0 (the count register),
                        !and terminate.
```

TRDRB
(Normal)

Definition:
Translate Byte, Decrement, and Repeat

Format:



Operation:
See Description

Flags Affected:
Z undefined
P/V set

Clock Cycles = 11 + 14n*
* n = number of iterations

Description:
The TRDRB instruction is used to translate a byte string from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte within memory to be translated. Register 'c' contains a

count of the number of bytes to be translated.

The TRDRB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into the address pointed to by Register 'd'.

Once the original byte is replaced by the translated byte, Register 'd' is decremented by 1, as is the count register, Register 'c'. This operation continues until Register 'c' = 0. The TRDRB instruction is interruptable at the end of each iteration.

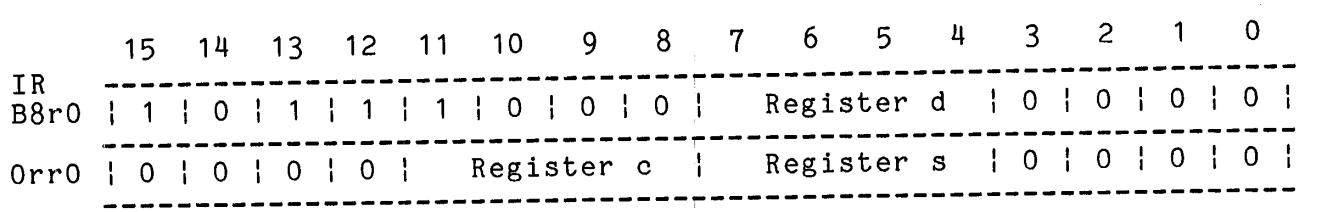
Example:

```
TRDRB    @R2,@R4,R0      !Translate the byte pointed
                         !to by R2 according to the
                         !table based at R4.  Decrement
                         !R2 and R0 (the count register),
                         !and repeat.
```

TRIB
(Normal)

Definition:
Translate Byte and Increment

Format:



Operation:
See Description

Flags Affected:
Z undefined
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The TRIB instruction is used to translate a byte from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte within memory to be translated. Register 'c' contains a count of the number of bytes to be translated.

The TRIB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into the address pointed to by Register 'd'.

Once the original byte is replaced by the translated byte, Register 'd' is incremented by 1, and Register 'c' is decremented. The TRIB instruction terminates after one iteration. The TRIRB instruction continues until Register 'c' = 0.

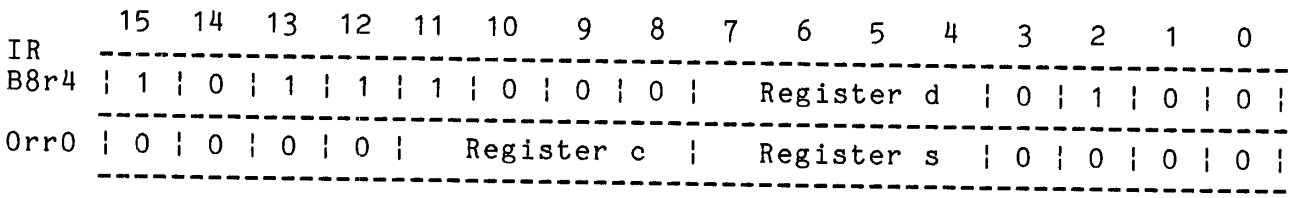
Example:

```
TRIB    @R2,@R4,R0      !Translate the byte pointed
                        !to by R2 according to the
                        !table based at R4.  Increment
                        !R2, decrement R0 (the count register),
                        !and terminate.
```

TRIRB
(Normal)

Definition:
Translate Byte, Increment, and Repeat

Format:



Operation:
See Description

Flags Affected:
Z undefined
P/V set

Clock Cycles = 11 + 14n *
* n = number of iterations

Description:
The TRIRB instruction is used to translate a byte from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte within memory to be translated. Register 'c' contains a count

of the number of bytes to be translated.

The TRIRB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into the address pointed to by Register 'd'.

Once the original byte is replaced by the translated byte, Register 'd' is incremented by 1, and Register 'c' is decremented. The TRIRB instruction then repeats the operation until Register 'c' = 0.

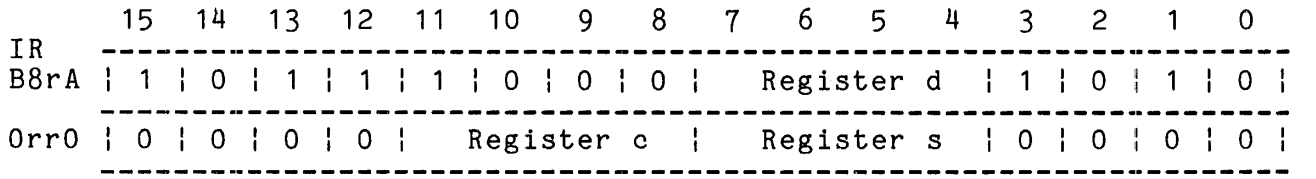
Example:

```
TRIRB    @R2,@R4,R0      !Translate the byte pointed
                         !to by R2 according to the
                         !table based at R4.  Increment
                         !R2, decrement R0 (the count register),
                         !and repeat.
```

TRTDB
(Normal)

Definition:
Translate Byte, Test, and Decrement

Format:



Operation:
See Description.

Flags Affected:
Z set if translated byte = 0, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The TRTDB instruction is used to translate a byte from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte within memory to be translated. Register 'c' contains a count of the number of bytes to be translated.

The TRTDB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into register RH1 for testing. The actual routine to test the byte must be separately provided.

Once the translated byte is loaded into RH1, Register 'd' is decremented by one to point to the next byte to be translated. Register 'c' is decremented by one to indicate the number of bytes remaining. The instruction is then terminated. The source byte is not altered.

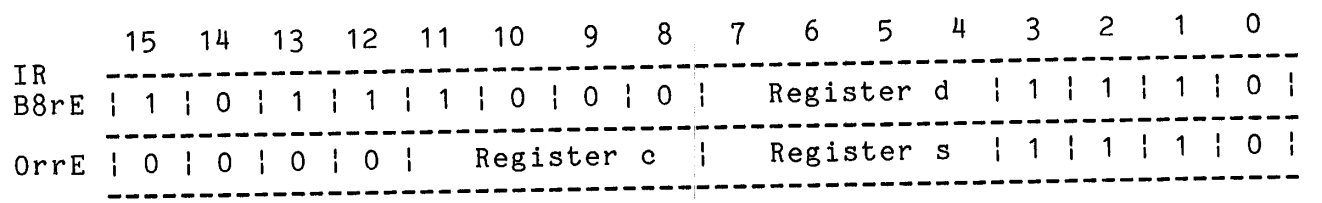
Example:

```
    TRTDB   @R2,@R8,R6      !Translate the byte located
                            !at R2 according to the table
                            !based at R8.  Place the translated
                            !byte in RH1.  Decrement R2 and
                            !R6 (the count register).
                            !Terminate the instruction.
```

TRTDRB
(Normal)

Definition:
Translate Byte, Test, Decrement, and Repeat

Format:



Operation:
See Description.

Flags Affected:
Z set if translated byte = 0, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 11 + 14n *
* n = number of iterations

Description:
The TRTDRB instruction is used to translate a byte from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte

within memory to be translated. Register 'c' contains a count of the number of bytes to be translated.

The TRTDRB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into register RH1 for testing. The actual routine to test the byte must be separately provided.

Once the translated byte is loaded into RH1, Register 'd' is decremented by one to point to the next byte to be translated. Register 'c' is decremented by one to indicate the number of bytes remaining. The instruction is then terminated. The source byte is not altered.

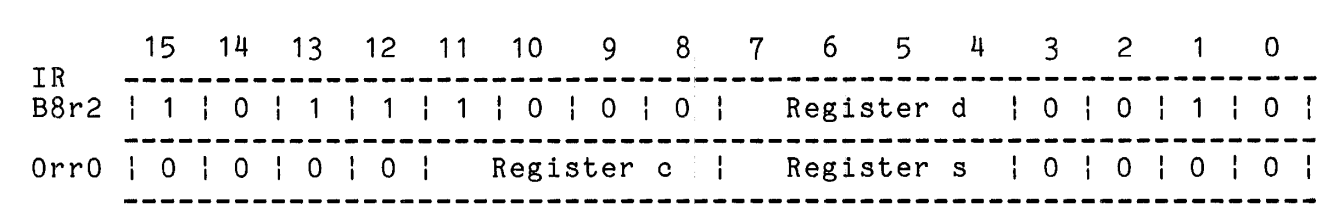
Example:

```
TRTDRB  @R2,@R8,R6      !Translate the byte located
                        !at R2 according to the table
                        !based at R8.  Place the translated
                        !byte in RH1.  Decrement R2 and
                        !R6 (the count register).
                        !Repeat the operation until R6 = 0.
```

TRTIB
(Normal)

Definition:
Translate Byte, Test, and Increment

Format:



Operation:
See Description.

Flags Affected:
Z set if translated byte = 0, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 25

Description:
The TRTIB instruction is used to translate a byte from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte within memory to be translated. Register 'c' contains a count of the number of bytes to be translated.

The TRTIB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into register RH1 for testing. The actual routine to test the byte must be separately provided.

Once the translated byte is loaded into RH1, Register 'd' is incremented by one to point to the next byte to be translated. Register 'c' is decremented by one to indicate the number of bytes remaining. The instruction is then terminated. The source byte is not altered.

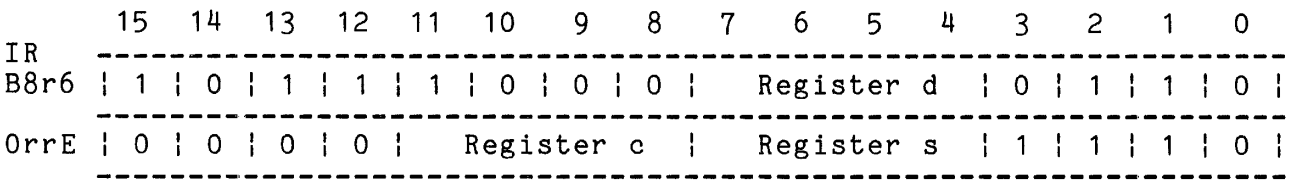
Example:

```
    TRTIB   @R2,@R8,R6      !Translate the byte located
                            !at R2 according to the table
                            !based at R8.  Place the translated
                            !byte in RH1.  Increment R2 and
                            !decrement R6 (the count register).
                            !Terminate the instruction.
```

TRTIRB
(Normal)

Definition:
Translate Byte, Test, Increment, and Repeat

Format:



Operation:
See Description.

Flags Affected:
Z set if translated byte = 0, reset otherwise
P/V set if Register c = 0, reset otherwise

Clock Cycles = 11 + 14n *
***** n = number of iterations

Description:
The TRTIRB instruction is used to translate a byte from one code to another (for example, ASCII to Baudot). Translation is performed according to a translation table set up in memory. Register 's' points to the starting (lowest) address of the translation table. Register 'd' points to the byte

within memory to be translated. Register 'c' contains a count of the number of bytes to be translated.

The TRTIRB instruction uses the value of the byte to be translated as an index pointer to within the translation table. That is, the value of the byte is added to the address contained in Register 's' to point to the translated byte. The translated byte is loaded from the table into register RH1 for testing. The actual routine to test the byte must be separately provided.

Once the translated byte is loaded into RH1, Register 'd' is incremented by one to point to the next byte to be translated. Register 'c' is decremented by one to indicate the number of bytes remaining. The operation is then repeated until Register 'c' = 0. The source byte is not altered.

Example:

```
TRTIRB  @R2,@R8,R6      !Translate the byte located
                        !at R2 according to the table
                        !based at R8.  Place the translated
                        !byte in RH1.  Increment R2 and
                        !decrement R6 (the count register).
                        !Repeat until R6 = 0.
```

The I/O Group

The I/O Group is made up of the following instructions:

Table 2-12
I/O Group

Mnemonic	Description
IN	Input a Word
INB	Input a Byte
IND	Input a Word and Decrement
INDB	Input a Byte and Decrement
INDR	Input a Word, Decrement, and Repeat
INDRB	Input a Byte, Decrement, and Repeat
INI	Input a Word and Increment
INIB	Input a Byte and Increment
INIR	Input a Word, Increment, and Repeat
INIRB	Input a Byte, Increment, and Repeat
OTDR	Output a Word, Decrement, and Repeat
OTDRB	Output a Byte, Decrement, and Repeat
OTIR	Output a Word, Increment, and Repeat
OTIRB	Output a Byte, Increment, and Repeat
OUT	Output a Word
OUTB	Output a Byte
OUTD	Output a Word and Decrement
OUTDB	Output a Byte and Decrement
OUTI	Output a Word and Increment
OUTIB	Output a Byte and Increment
SIN	Special Input a Word
SINB	Special Input a Byte
SIND	Special Input a Word and Decrement
SINDB	Special Input a Byte and Decrement
SINDR	Special Input a Word, Decrement, and Repeat
SINDRB	Special Input a Byte, Decrement, and Repeat
SINI	Special Input a Word and Increment
SINIB	Special Input a Byte and Increment

Table 2-12 (Cont.)
I/O Group

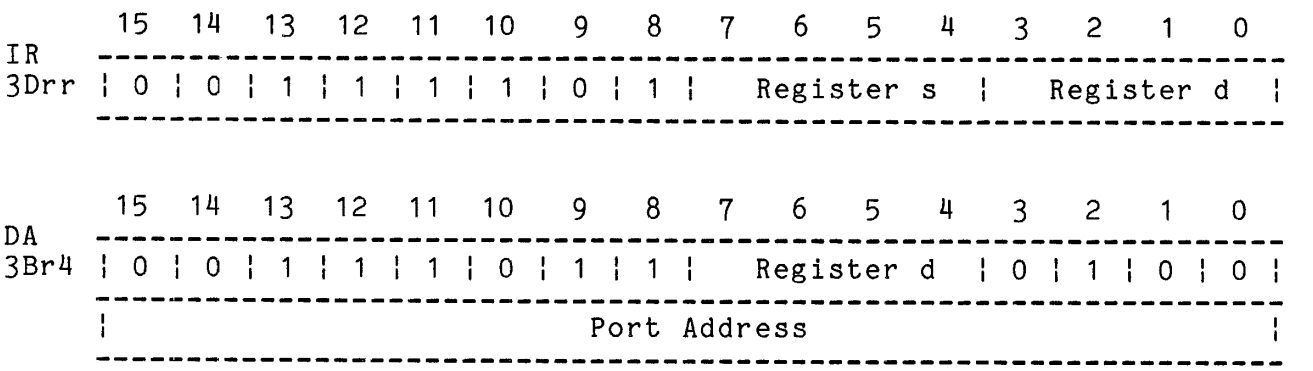
Mnemonic	Description
SINIR	Special Input a Word, Increment, and Repeat
SINIRB	Special Input a Byte, Increment, and Repeat
SOTDR	Special Output a Word, Decrement, and Repeat
SOTDRB	Special Output a Byte, Decrement, and Repeat
SOTIR	Special Output a Word, Increment, and Repeat
SOTIRB	Special Output a Byte, Increment, and Repeat
SOUT	Special Output a Word
SOUTB	Special Output a Byte
SOUTD	Special Output a Word and Decrement
SOUTDB	Special Output a Byte and Decrement
SOUTI	Special Output a Word and Increment
SOUTIB	Special Output a Byte and Increment

The "special" I/O instructions (ie: SIN, SOUT, etc.) are used primarily for communicating with the Z8010 Memory Management Unit. Special I/O is indicated on the Z8000 status lines. All I/O instructions are considered system instructions.

IN
(System)

Definition:
Input a Word

Format:



Operation:
Z8001/2

Register d<0:15> <-- Source<0:15>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The IN instruction inputs a 16-bit word from the designated I/O source, and places the word in Register 'd'.

Example:

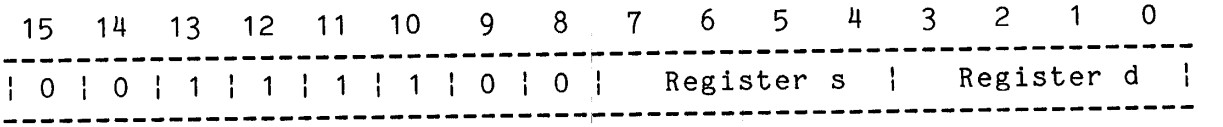
```
    IN      R4,@R2          !Load the contents of the I/O
                            !port pointed to by R2 into R4.
                            !(IR mode)
```

INB
(System)

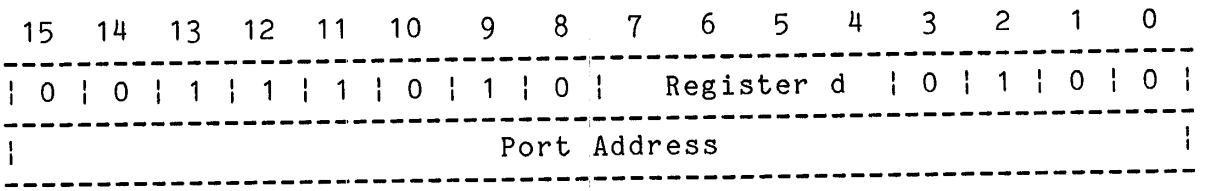
Definition:
Input a Byte

Format:

IR 3Crr



DA 3Ar4



Operation:
Z8001/2

```
Register d<0:7> <-- Source<0:7>
```

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The INB instruction inputs an 8-bit byte from the designated I/O source, and places the word in Register 'd'. The INB instruction requires that the source address be an odd, 16-bit address

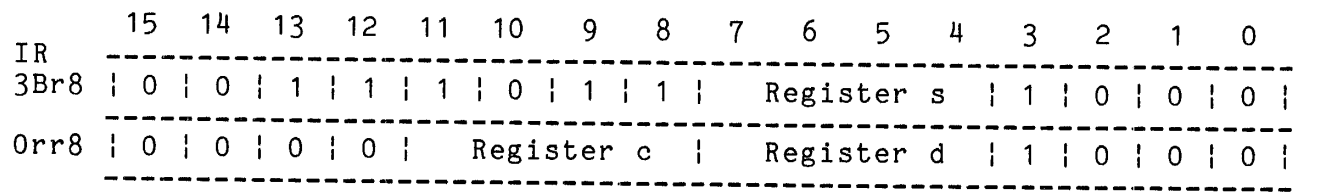
Example:

```
    IN      RL4,@R2         !Load the contents of the I/O
                            !port pointed to by R2 into RL4.
                            !(IR mode)
```

IND
(System)

Definition:
Input a Word and Decrement

Format:



Operation:
Z8001/2

Register d<0:15> <-- Source<0:15>
Register d<0:15> <-- Register d<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The IND instruction is typically used to input a string of words from an I/O port. The IND instruction inputs a 16-bit word from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction,

Register 'c' = 0, the P/V flag is set.

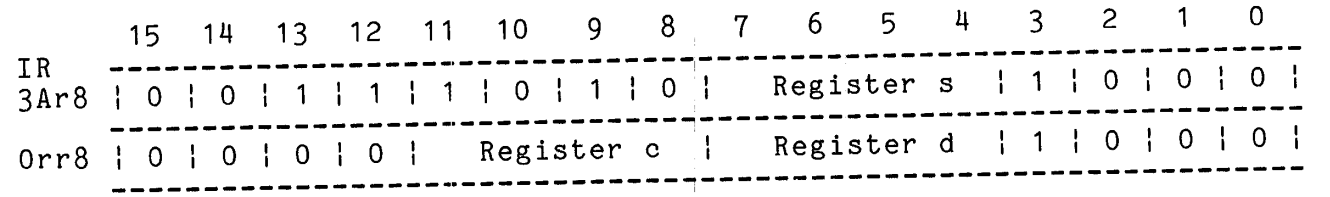
Example:

```
    IND     @R4,@R2,R0      !Load the contents of the I/O
                            !port pointed to by R2 into the
                            !memory location pointed to by
                            !R4.  Decrement R4 by 2, and
                            !R0 by 1.
```

INDB
(System)

Definition:
Input a Byte and Decrement

Format:



Operation:
Z8001/2

Register d<0:7> <-- Source<0:7>
Register d<0:15> <-- Register d<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The INDB instruction is typically used to input a string of bytes from an I/O port. The INDB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

Example:

```
INDB    @R4,@R2,R0      !Load the contents of the I/O
                        !port pointed to by R2 into the
                        !memory location pointed to by
                        !R4. Decrement R4 by 1, and
                        !R0 by 1.
```

INDR
(System)

Definition:
Input a Word, Decrement, and Repeat

Format:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR																
3Br8	0	0	1	1	1	0	1	1	Register s				1	0	0	0
0rr0	0	0	0	0	Register c				Register d				0	0	0	0

Operation:
Z8001/2

Register d<0:15> <-- Source<0:15>
Register d<0:15> <-- Register d<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
* n = number of iterations

Description:
The INDR instruction is typically used to input a string of bytes from an I/O port. The INDR instruction inputs a 16-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The INDR instruction is interruptable at the end of each iteration.

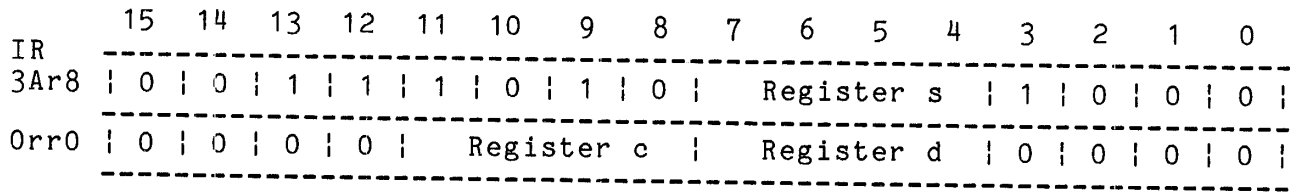
Example:

```
INDR    @R4,@R2,R0      !Load the contents of the I/O
                        !port pointed to by R2 into the
                        !memory location pointed to by
                        !R4.  Decrement R4 by 2, and
                        !R0 by 1.  Repeat until
                        !R0 = 0.
```

INDRB
(System)

Definition:
Input a Byte, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:7> <-- Source<0:7>
Register d<0:15> <-- Register d<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The INDRB instruction is typically used to input a string of bytes from an I/O port. The INDRB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The INDRB instruction is interruptable at the end of each iteration.

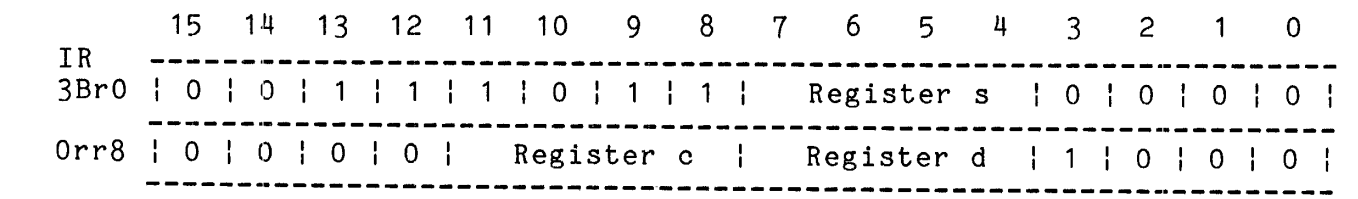
Example:

```
INDRB   @R4,@R2,R0      !Load the contents of the I/O
                        !port pointed to by R2 into the
                        !memory location pointed to by
                        !R4. Decrement R4 by 1, and
                        !R0 by 1. Repeat until
                        !R0 = 0.
```

INI
(System)

Definition:
Input a Word and Increment

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register d<0:15> <-- Register d<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1
```

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The INI instruction is typically used to input a string of words from an I/O port. The INI instruction inputs a 16-bit word from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

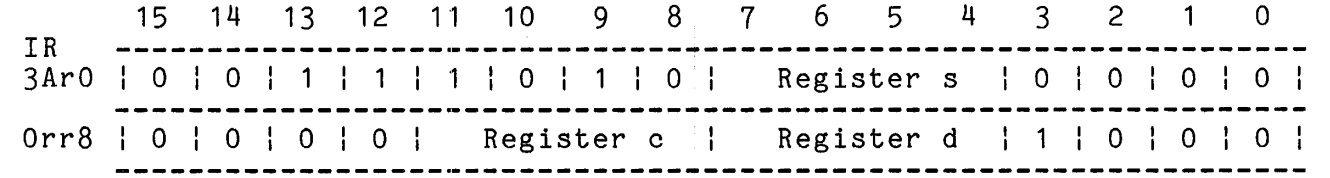
Example:

```
    INI     @R4,@R2,R0      !Load the contents of the I/O
                            !port pointed to by R2 into the
                            !memory location pointed to by
                            !R4.  Increment R4 by 2, and
                            !decrement R0 by 1.
```

INIB
(System)

Definition:
Input a Byte and Increment

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register d<0:15> <-- Register d<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The INIB instruction is typically used to input a string of bytes from an I/O port. The INIB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

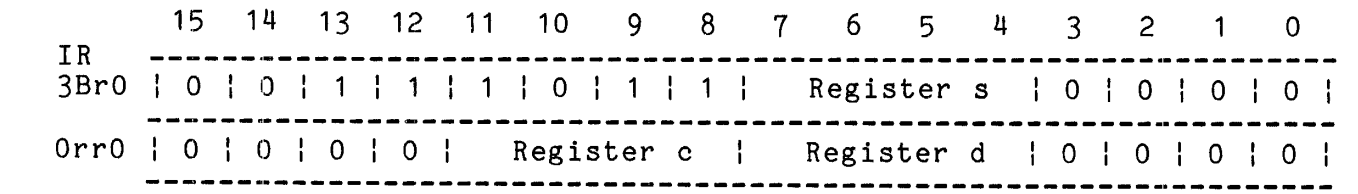
Example:

```
    INIB    @R4,@R2,R0      !Load the contents of the I/O
                            !port pointed to by R2 into the
                            !memory location pointed to by
                            !R4.  Increment R4 by 1, and
                            !decrement R0 by 1.
```

INIR
(System)

Definition:
Input a Word, Increment, and Repeat

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15>
Register d<0:15> <-- Register d<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
* n = number of iterations

Description:
The INIR instruction is typically used to input a string of words from an I/O port. The INIR instruction inputs a 16-bit word from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The INIR instruction is interruptible at the end of each iteration.

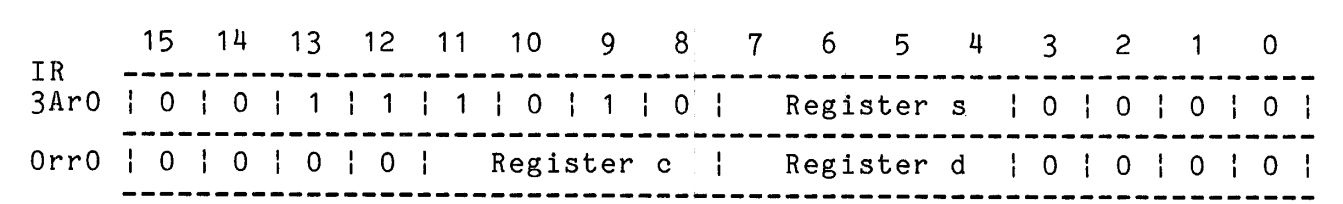
Example:

```
INIR    @R4,@R2,R0      !Load the contents of the I/O
                        !port pointed to by R2 into the
                        !memory location pointed to by
                        !R4.  Increment R4 by 2, and
                        !decrement R0 by 1.  Repeat until
                        !R0 = 0.
```

INIRB
(System)

Definition:
Input a Byte, Increment, and Repeat

Format:



Operation:
Z8001/2

```
-------
Register d<0:7> <-- Register s<0:7>
Register d<0:15> <-- Register d<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
* n = number of iterations

Description:
The INIRB instruction is typically used to input a string of bytes from an I/O port. The INIRB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The INIRB instruction is interruptible at the end of each iteration.

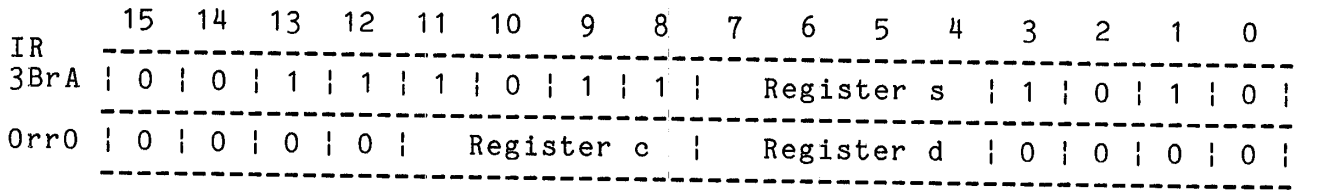
Example:

```
INIRB   @R4,@R2,R0      !Load the contents of the I/O
                        !port pointed to by R2 into the
                        !memory location pointed to by
                        !R4. Increment R4 by 1, and
                        !decrement R0 by 1. Repeat until
                        !R0 = 0.
```

OTDR
(System)

Definition:
Output a Word, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The OTDR instruction is typically used to output a string of words to an I/O port. The OTDR instruction outputs a 16-bit word from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The OTDR instruction is interruptible at the end of each iteration.

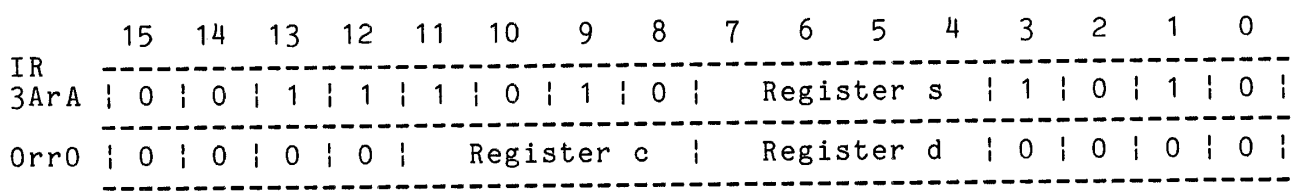
Example:

```
    OTDR    @R4,@R2,R0      !Move the contents of the memory
                            !location pointed to by R2 into the
                            !I/O port pointed to by
                            !R4.  Decrement R4 by 2, and
                            !decrement R0 by 1.  Repeat until
                            !R0 = 0.
```

OTDRB
(System)

Definition:
Output a Byte, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
-------
Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The OTDRB instruction is typically used to output a string of bytes to an I/O port. The OTDRB instruction outputs an 8-bit byte from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The OTDRB instruction is interruptible at the end of each iteration.

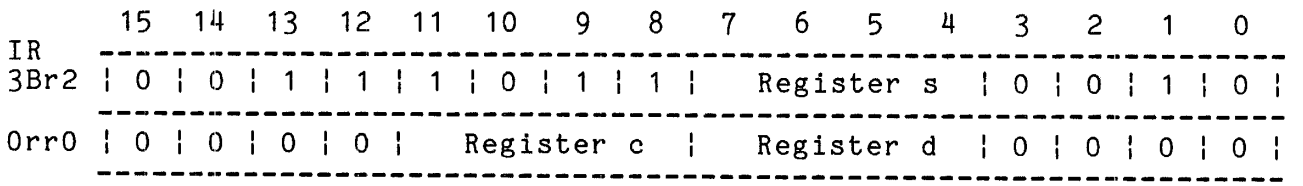
Example:

```
    OTDRB   @R4,@R2,R0      !Move the contents of the memory
                            !location pointed to by R2 into the
                            !I/O port pointed to by R4.
                            !Decrement R4 by 1, and
                            !decrement R0 by 1.  Repeat until
                            !R0 = 0.
```

OTIR
(System)

Definition:
Input a Word, Increment, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The OTIR instruction is typically used to output a string of words to an I/O port. The OTIR instruction outputs a 16-bit word from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The OTIR instruction is interruptible at the end of each iteration.

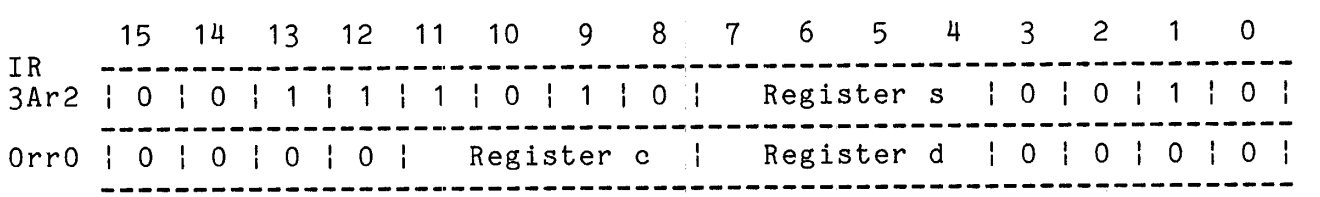
Example:

```
    OTIR    @R4,@R2,R0      !Move the contents of the memory
                            !location pointed to by R2 into the
                            !I/O port pointed to by
                            !R4.  Increment R4 by 2, and
                            !decrement R0 by 1.  Repeat until
                            !R0 = 0.
```

OTIRB
(System)

Definition:
Output a Byte, Increment, and Repeat

Format:



Operation:
Z8001/2

```
-------
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The OTIRB instruction is typically used to output a string of words to an I/O port. The OTIRB instruction outputs an 8-bit word from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The OTIRB instruction is interruptible at the end of each iteration.

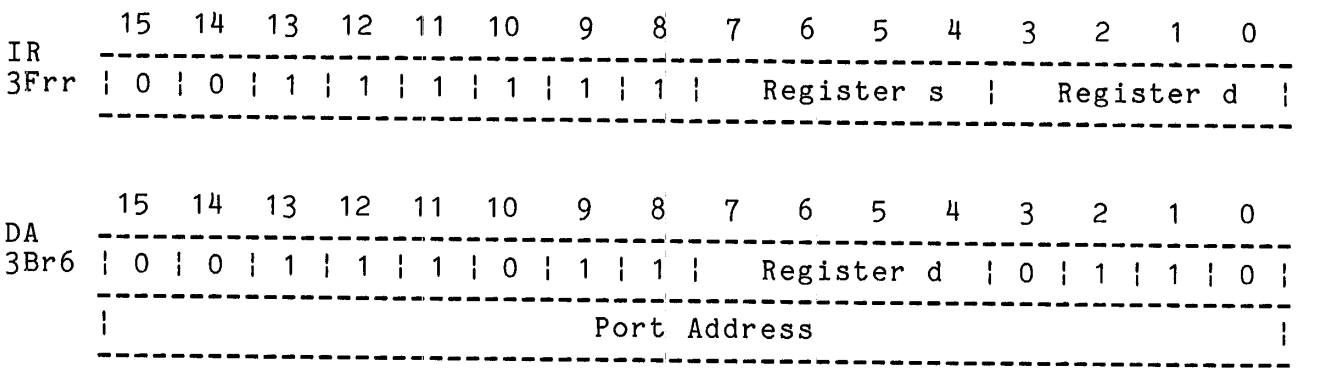
Example:

```
OTIRB   @R4,@R2,R0      !Move the contents of the memory
                        !location pointed to by R2 into the
                        !I/O port pointed to by
                        !R4.  Increment R4 by 1, and
                        !decrement R0 by 1.  Repeat until
                        !R0 = 0.
```

OUT
(System)

Definition:
Output a Word

Format:



Operation:
Z8001/2

Destination <0:15> <-- Register s<0:15>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The OUT instruction outputs a 16-bit word from the memory location pointed to by Register 's', to the I/O port specified.

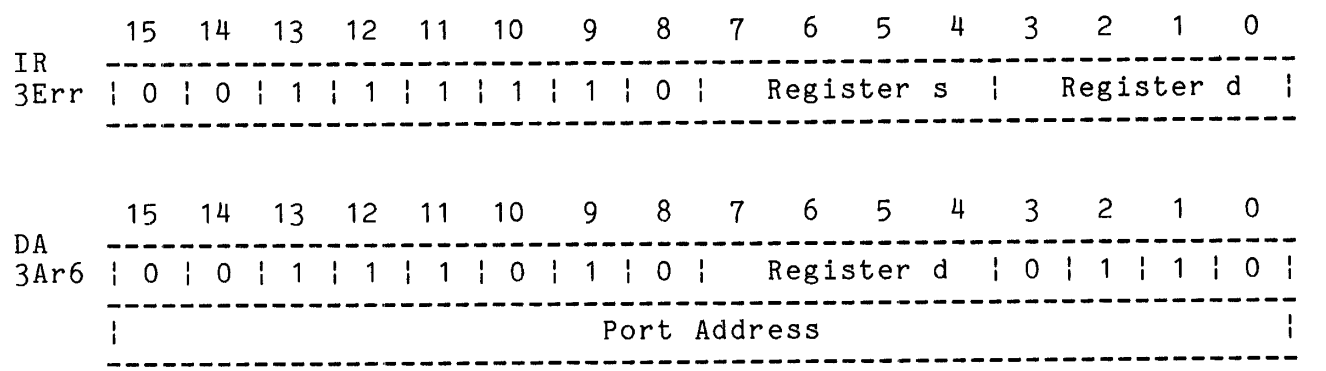
Example:

```
    OUT     @R4,R2          !Move the contents of the memory
                            !location pointed to by R4 to the I/O
                            !port pointed to by R2.
                            !(IR mode)
```

OUTB
(System)

Definition:
Output a Byte

Format:



Operation:
Z8001/2

Destination<0:7> <-- Register s<0:7>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The OUTB instruction outputs an 8-bit byte from memory pointed to by Register 's', to the I/O port specified as the destination. The OUTB instruction requires that the source address be an odd, 16-bit address.

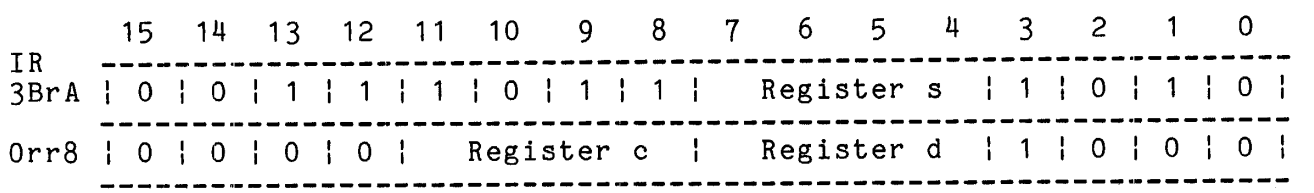
Example:

```
    OUTB      @RL4,R2        !Move the contents of the memory
                             !location pointed to by RL4 to the
                             !I/O port pointed to by R2.
                             !(IR mode)
```

OUTD
(System)

Definition:
Output a Word and Decrement

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1
```

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The OUTD instruction is typically used to output a string of words to an I/O port. The OUTD instruction moves a 16-bit word pointed to by Register 's' to the I/O port pointed to by Register 'd'. Register 's' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

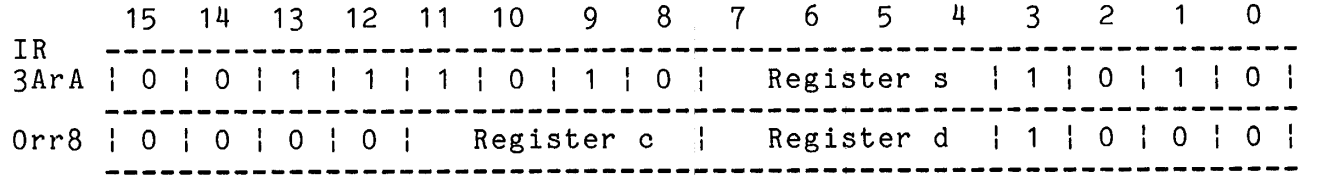
Example:

```
OUTD    @R4,@R2,R0      !Move the contents of the memory
                        !location pointed to by R2 into the
                        !I/O port pointed to by
                        !R4.  Decrement R4 by 2, and
                        !R0 by 1.
```

OUTDB
(System)

Definition:
Output a Byte and Decrement

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The OUTDB instruction is typically used to output a string of bytes to an I/O port. The OUTDB instruction outputs an 8-bit byte from memory pointed to by Register 's' to an I/O port pointed to by Register 'd'. Register 's' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

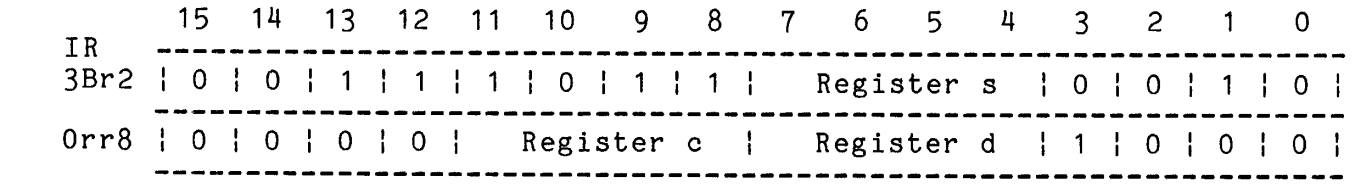
Example:

```
OUTDB    @R4,@R2,R0      !Move the contents of the memory
                         !location pointed to by R2 into the
                         !I/O port pointed to by
                         !R4.  Decrement R4 by 1, and
                         !R0 by 1.
```

OUTI
(System)

Definition:
Output a Word and Increment

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The OUTI instruction is typically used to output a string of words to an I/O port. The OUTI instruction moves a 16-bit word pointed to by Register 's' to the I/O port pointed to by Register 'd'. Register 's' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

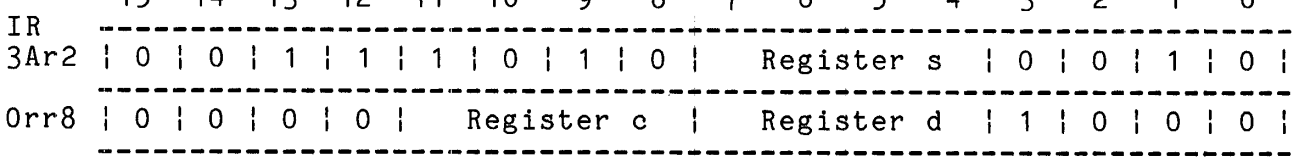
Example:

```
    OUTI    @R4,@R2,R0      !Move the contents of the memory
                            !location pointed to by R2 into the
                            !I/O port pointed to by
                            !R4.  Increment R4 by 2, and
                            !decrement R0 by 1.
```

OUTIB
(System)

Definition:
Output a Byte and Increment

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The OUTIB instruction is typically used to output a string of bytes to an I/O port. The OUTIB instruction outputs an 8-bit byte from memory pointed to by Register 's' to an I/O port pointed to by Register 'd'. Register 's' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

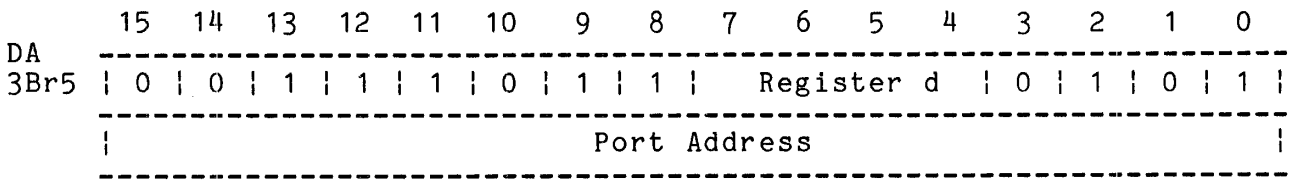
Example:

```
OUTIB    @R4,@R2,R0      !Move the contents of the memory
                         !location pointed to by R2 into the
                         !I/O port pointed to by
                         !R4.  Increment R4 by 1, and
                         !decrement R0 by 1.
```

SIN
(System)

Definition:
Special Input a Word

Format:



Operation:
Z8001/2

Register d<0:15> <-- Source<0:15>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The SIN instruction inputs a 16-bit word from the designated I/O source, and places the word in Register 'd'.

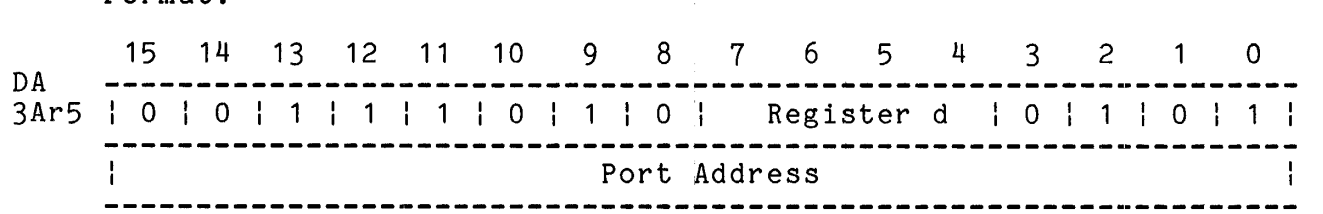
Example:

```
    SIN      R4,@R2          !Load the contents of the I/O
                             !port pointed to by R2 into R4.
                             !(IR mode)
```

SINB
(System)

Definition:
Special Input a Byte

Format:



Operation:
Z8001/2

Register d<0:7> <-- Source<0:7>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The SINB instruction inputs an 8-bit byte from the designated I/O source, and places the word in Register 'd'. The SINB instruction requires that the source address be an odd, 16-bit address

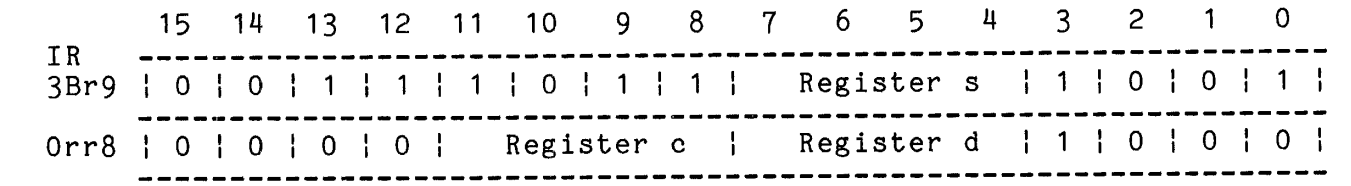
Example:

```
    SIN     RL4,@R2         !Load the contents of the I/O
                            !port pointed to by R2 into RL4.
                            !(IR mode)
```

SIND
(System)

Definition:
Special Input a Word and Decrement

Format:



Operation:
Z8001/2

Register d<0:15> <-- Source<0:15>
Register d<0:15> <-- Register d<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SIND instruction is typically used to input a string of words from an I/O port. The SIND instruction inputs a 16-bit word from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

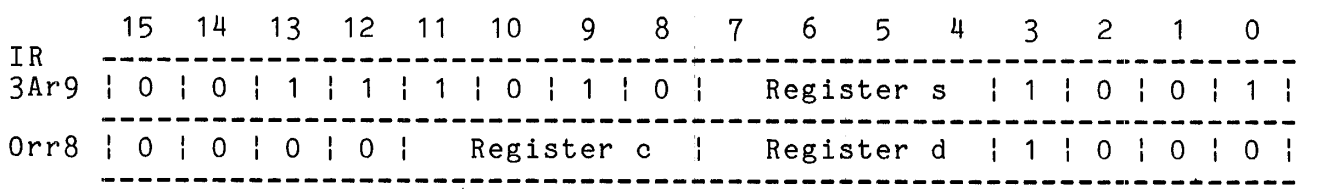
Example:

```
    SIND     @R4,@R2,R0      !Load the contents of the I/O
                             !port pointed to by R2 into the
                             !memory location pointed to by
                             !R4.  Decrement R4 by 2, and
                             !R0 by 1.
```

SINDB
(System)

Definition:
Special Input a Byte and Decrement

Format:



Operation:
Z8001/2

Register d<0:7> <-- Source<0:7>
Register d<0:15> <-- Register d<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SINDB instruction is typically used to input a string of bytes from an I/O port. The SINDB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

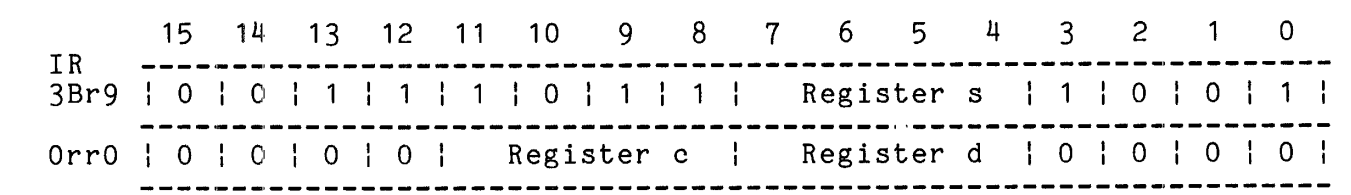
Example:

```
SINDB    @R4,@R2,R0      !Load the contents of the I/O
                         !port pointed to by R2 into the
                         !memory location pointed to by
                         !R4.  Decrement R4 by 1, and
                         !R0 by 1.
```

SINDR
(System)

Definition:
Special Input a Word, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Source<0:15>
Register d<0:15> <-- Register d<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The SINDR instruction is typically used to input a string of bytes from an I/O port. The SINDR instruction inputs a 16-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SINDR instruction is interruptable at the end of each iteration.

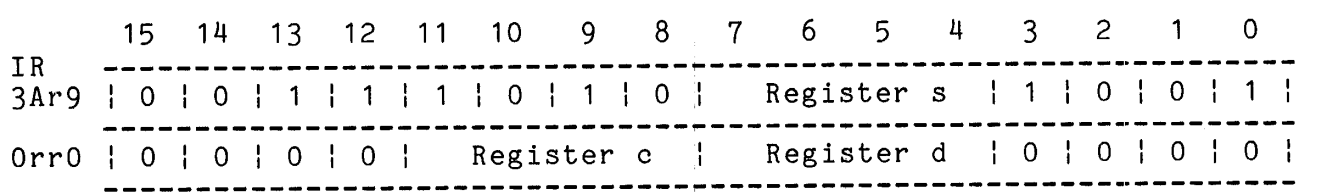
Example:

```
SINDR     @R4,@R2,R0      !Load the contents of the I/O
                          !port pointed to by R2 into the
                          !memory location pointed to by
                          !R4.  Decrement R4 by 2, and
                          !R0 by 1.  Repeat until
                          !R0 = 0.
```

SINDRB
(System)

Definition:
Special Input a Byte, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
-------
Register d<0:7> <-- Source<0:7>
Register d<0:15> <-- Register d<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The SINDRB instruction is typically used to input a string of bytes from an I/O port. The SINDRB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SINDRB instruction is interruptable at the end of each iteration.

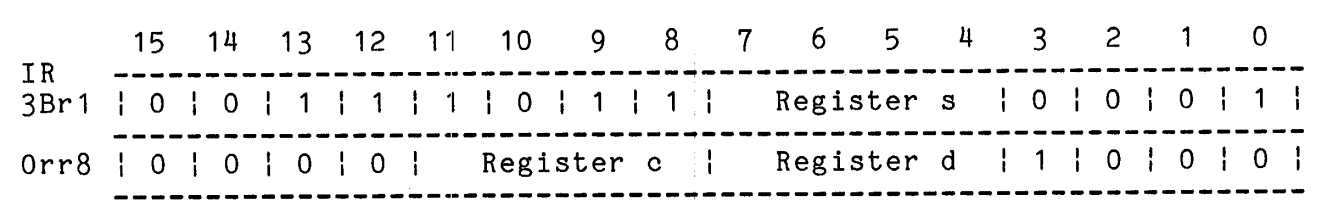
Example:

```
SINDRB   @R4,@R2,R0      !Load the contents of the I/O
                         !port pointed to by R2 into the
                         !memory location pointed to by
                         !R4. Decrement R4 by 1, and
                         !R0 by 1. Repeat until
                         !R0 = 0.
```

SINI
(System)

Definition:
Special Input a Word and Increment

Format:



Operation:
Z8001/2

Register d<0:15> <-- Register s<0:15>
Register d<0:15> <-- Register d<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SINI instruction is typically used to input a string of words from an I/O port. The SINI instruction inputs a 16-bit word from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

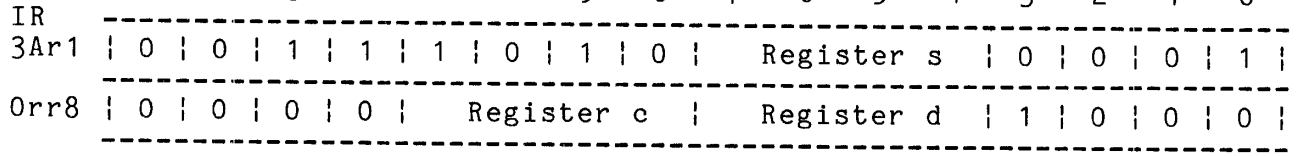
Example:

```
SINI    @R4,@R2,R0      !Load the contents of the I/O
                        !port pointed to by R2 into the
                        !memory location pointed to by
                        !R4. Increment R4 by 2, and
                        !decrement R0 by 1.
```

SINIB
(System)

Definition:
Special Input a Byte and Increment

Format:



Operation:
Z8001/2

```
Register d<0:7> <-- Register s<0:7>
Register d<0:15> <-- Register d<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1
```

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SINIB instruction is typically used to input a string of bytes from an I/O port. The SINIB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

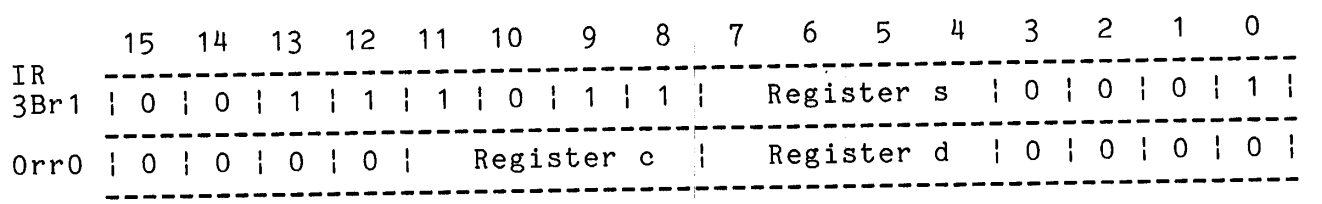
Example:

```
    SINIB    @R4,@R2,R0      !Load the contents of the I/O
                             !port pointed to by R2 into the
                             !memory location pointed to by
                             !R4.  Increment R4 by 1, and
                             !decrement R0 by 1.
```

SINIR
(System)

Definition:
Special Input a Word, Increment, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register d<0:15> <-- Register d<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
* n = number of iterations

Description:
The SINIR instruction is typically used to input a string of words from an I/O port. The SINIR instruction inputs a 16-bit word from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SINIR instruction is interruptible at the end of each iteration.

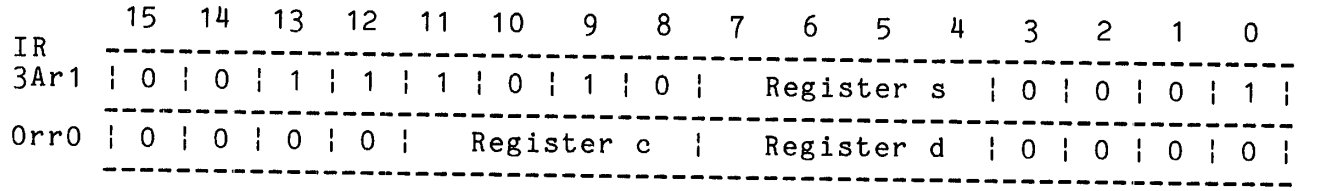
Example:

```
SINIR    @R4,@R2,R0      !Load the contents of the I/O
                         !port pointed to by R2 into the
                         !memory location pointed to by
                         !R4.  Increment R4 by 2, and
                         !decrement R0 by 1.  Repeat until
                         !R0 = 0.
```

SINIRB
(System)

Definition:
Special Input a Byte, Increment, and Repeat

Format:



Operation:
Z8001/2

```
-------
Register d<0:7> <-- Register s<0:7>
Register d<0:15> <-- Register d<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The SINIRB instruction is typically used to input a string of bytes from an I/O port. The SINIRB instruction inputs an 8-bit byte from the I/O port pointed to by Register 's', and places the word in the memory location pointed to by Register 'd'. Register 'd' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SINIRB instruction is interruptible at the end of each iteration.

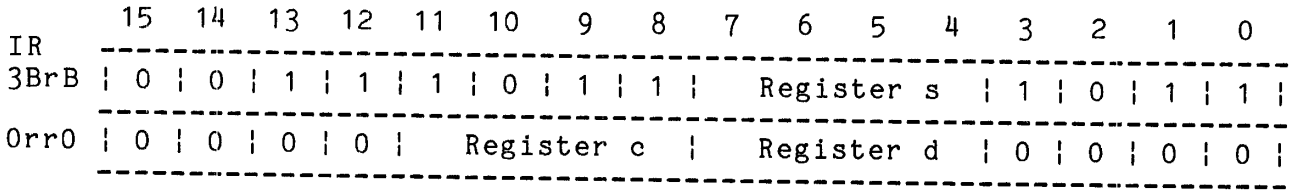
Example:

```
SINIRB   @R4,@R2,R0      !Load the contents of the I/O
                         !port pointed to by R2 into the
                         !memory location pointed to by
                         !R4.  Increment R4 by 1, and
                         !decrement R0 by 1.  Repeat until
                         !R0 = 0.
```

SOTDR
(System)

Definition:
Special Output a Word, Decrement, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
* n = number of iterations

Description:
The SOTDR instruction is typically used to output a string of words to an I/O port. The SOTDR instruction outputs a 16-bit word from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SOTDR instruction is interruptible at the end of each iteration.

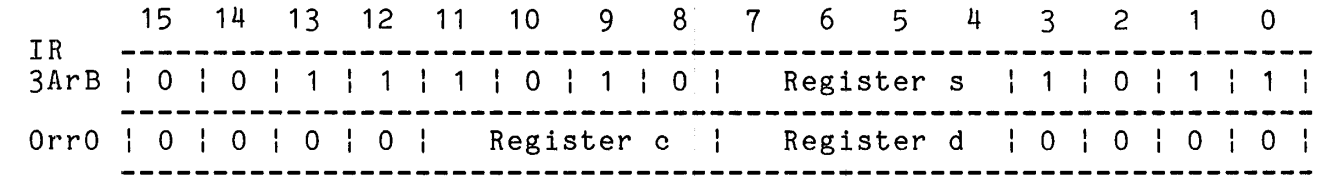
Example:

```
    SOTDR    @R4,@R2,R0      !Move the contents of the memory
                             !location pointed to by R2 into the
                             !I/O port pointed to by
                             !R4.  Decrement R4 by 2, and
                             !decrement R0 by 1.  Repeat until
                             !R0 = 0.
```

SOTDRB
(System)

Definition:
Special Output a Byte, Decrement, and Repeat

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The SOTDRB instruction is typically used to output a string of bytes to an I/O port. The SOTDRB instruction outputs an 8-bit byte from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SOTDRB instruction is interruptible at the end of each iteration.

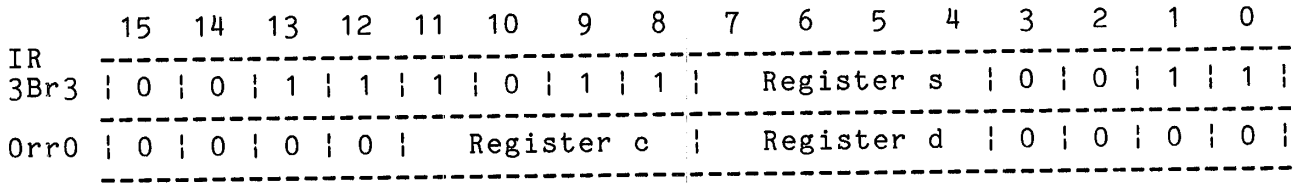
Example:

```
SOTDRB   @R4,@R2,R0      !Move the contents of the memory
                         !location pointed to by R2 into the
                         !I/O port pointed to by R4.
                         !Decrement R4 by 1, and
                         !decrement R0 by 1.  Repeat until
                         !R0 = 0.
```

SOTIR
(System)

Definition:
Special Input a Word, Increment, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
* n = number of iterations

Description:
The SOTIR instruction is typically used to output a string of words to an I/O port. The SOTIR instruction outputs a 16-bit word from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SOTIR instruction is interruptible at the end of each iteration.

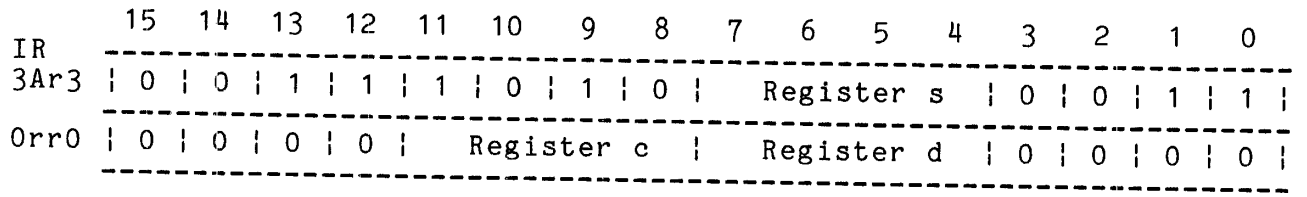
Example:

```
SOTIR    @R4,@R2,R0      !Move the contents of the memory
                         !location pointed to by R2 into the
                         !I/O port pointed to by
                         !R4.  Increment R4 by 2, and
                         !decrement R0 by 1.  Repeat until
                         !R0 = 0.
```

SOTIRB
(System)

Definition:
Special Output a Byte, Increment, and Repeat

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1
Repeat until Register c = 0
```

Flags Affected:
P/V set at termination

Clock Cycles = 11 + 10n *
 * n = number of iterations

Description:
The SOTIRB instruction is typically used to output a string of words to an I/O port. The SOTIRB instruction outputs an 8-bit word from the memory location pointed to by Register 's', to the I/O port pointed to by Register 'd'. Register 's' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of an iteration, Register 'c' = 0, the P/V flag is set and the instruction terminates. The SOTIRB instruction is interruptible at the end of each iteration.

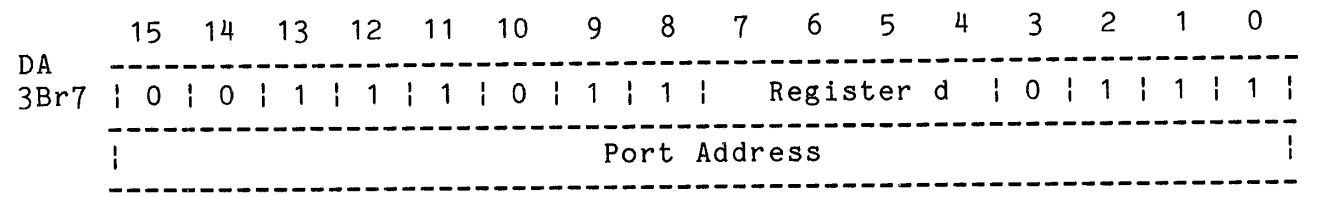
Example:

```
SOTIRB   @R4,@R2,R0      !Move the contents of the memory
                         !location pointed to by R2 into the
                         !I/O port pointed to by
                         !R4.  Increment R4 by 1, and
                         !decrement R0 by 1.  Repeat until
                         !R0 = 0.
```

SOUT
(System)

Definition:
Special Output a Word

Format:



Operation:
Z8001/2

Destination<0:15> <-- Register s<0:15>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The SOUT instruction outputs a 16-bit word from the memory location pointed to by Register 's', to the I/O port specified.

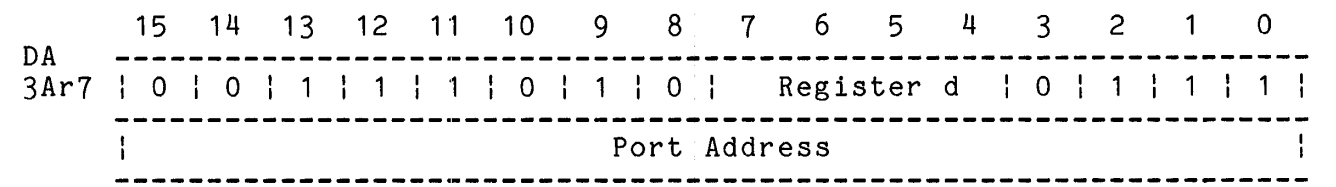
Example:

```
    SOUT    @R4,R2          !Move the contents of the memory
                            !location pointed to by R4 to the I/O
                            !port pointed to by R2.
                            !(IR mode)
```

SOUTB
(System)

Definition:
Special Output a Byte

Format:



Operation:
Z8001/2

Destination<0:7> <-- Register s<0:7>

Flags Affected: None

Clock Cycles =

Addressing Mode	Clock Cycles
IR - S,NS	10
DA - S,NS	12

Description:
The SOUTB instruction outputs an 8-bit byte from memory pointed to by Register 's', to the I/O port specified as the destination. The SOUTB instruction requires that the source address be an odd, 16-bit address.

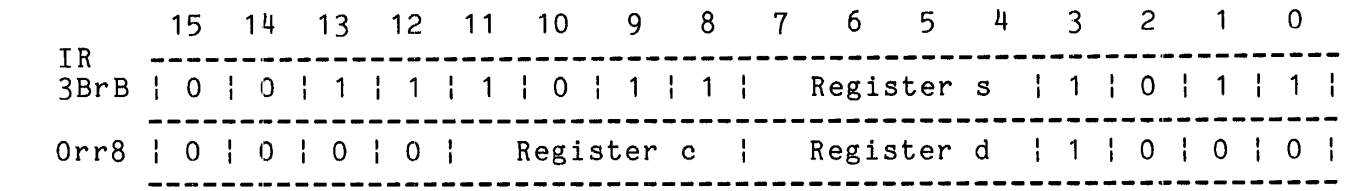
Example:

```
    SOUTB     @RL4,R2        !Move the contents of the memory
                             !location pointed to by RL4 to the
                             !I/O port pointed to by R2.
                             !(IR mode)
```

SOUTD
(System)

Definition:
Special Output a Word and Decrement

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> - 2
Register c<0:15> <-- Register c<0:15> - 1
```

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SOUTD instruction is typically used to output a string of words to an I/O port. The SOUTD instruction moves a 16-bit word pointed to by Register 's' to the I/O port pointed to by Register 'd'. Register 's' is decremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

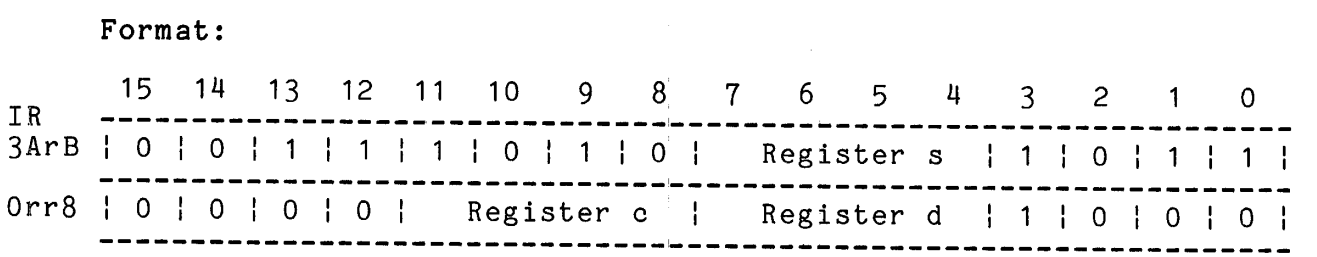
Example:

```
    SOUTD     @R4,@R2,R0     !Move the contents of the memory
                             !location pointed to by R2 into the
                             !I/O port pointed to by
                             !R4.  Decrement R4 by 2, and
                             !R0 by 1.
```

SOUTDB
(System)

Definition:
Special Output a Byte and Decrement

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> - 1
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SOUTDB instruction is typically used to output a string of bytes to an I/O port. The SOUTDB instruction outputs an 8-bit byte from memory pointed to by Register 's' to an I/O port pointed to by Register 'd'. Register 's' is decremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

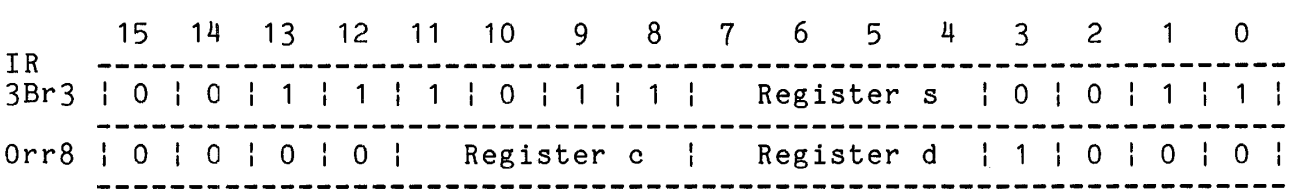
Example:

```
    SOUTDB    @R4,@R2,R0     !Move the contents of the memory
                             !location pointed to by R2 into the
                             !I/O port pointed to by
                             !R4.  Decrement R4 by 1, and
                             !R0 by 1.
```

SOUTI
(System)

Definition:
Special Output a Word and Increment

Format:



Operation:
Z8001/2

```
Register d<0:15> <-- Register s<0:15>
Register s<0:15> <-- Register s<0:15> + 2
Register c<0:15> <-- Register c<0:15> - 1
```

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SOUTI instruction is typically used to output a string of words to an I/O port. The SOUTI instruction moves a 16-bit word pointed to by Register 's' to the I/O port pointed to by Register 'd'. Register 's' is incremented by 2, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

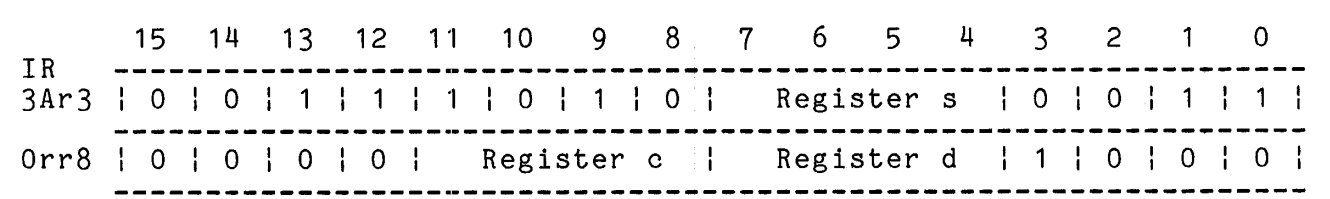
Example:

```
    SOUTI     @R4,@R2,R0     !Move the contents of the memory
                             !location pointed to by R2 into the
                             !I/O port pointed to by
                             !R4.  Increment R4 by 2, and
                             !decrement R0 by 1.
```

SOUTIB
(System)

Definition:
Special Output a Byte and Increment

Format:



Operation:
Z8001/2

Register d<0:7> <-- Register s<0:7>
Register s<0:15> <-- Register s<0:15> + 1
Register c<0:15> <-- Register c<0:15> - 1

Flags Affected:
P/V set if Register c = 0, reset otherwise

Clock Cycles = 21

Description:
The SOUTIB instruction is typically used to output a string of bytes to an I/O port. The SOUTIB instruction outputs an 8-bit byte from memory pointed to by Register 's' to an I/O port pointed to by Register 'd'. Register 's' is incremented by 1, and Register 'c' is decremented by 1. If, at the completion of the instruction, Register 'c' = 0, the P/V flag is set.

Example:

```
    SOUTIB    @R4,@R2,R0    !Move the contents of the memory
                            !location pointed to by R2 into the
                            !I/O port pointed to by
                            !R4.  Increment R4 by 1, and
                            !decrement R0 by 1.
```

3

DEVICE INTERFACE

INTRODUCTION

In this chapter, we discuss how the Z8000 devices interface to the outside world. Each of the Z8000 signal lines is discussed in terms of function and timing. We also look at system timing, interrupt structure, memory implementation, and input/output (I/O) requirements and capabilities.

First, let's look at the Z8001 and Z8002 device pin-outs.

THE Z8001 PIN-OUT

The Z8001 is a 48-pin device. Of those 48 pins, 23 are used in address and data manipulation, while the remainder serve a control function. Figure 3-1 illustrates the pinout of the Z8001. The following describes the function of each pin.

AD0-AD15 Address and data information are multiplexed over these lines. AD0-AD15 are tri-state lines and are used for all memory and I/O transactions. The lines are positive true (1 = HIGH, 0 = LOW). Data transactions are indicated when the DS (data strobe) line is asserted, and address

transactions are indicated when the $\overline{AS}$ (address strobe) line is asserted.

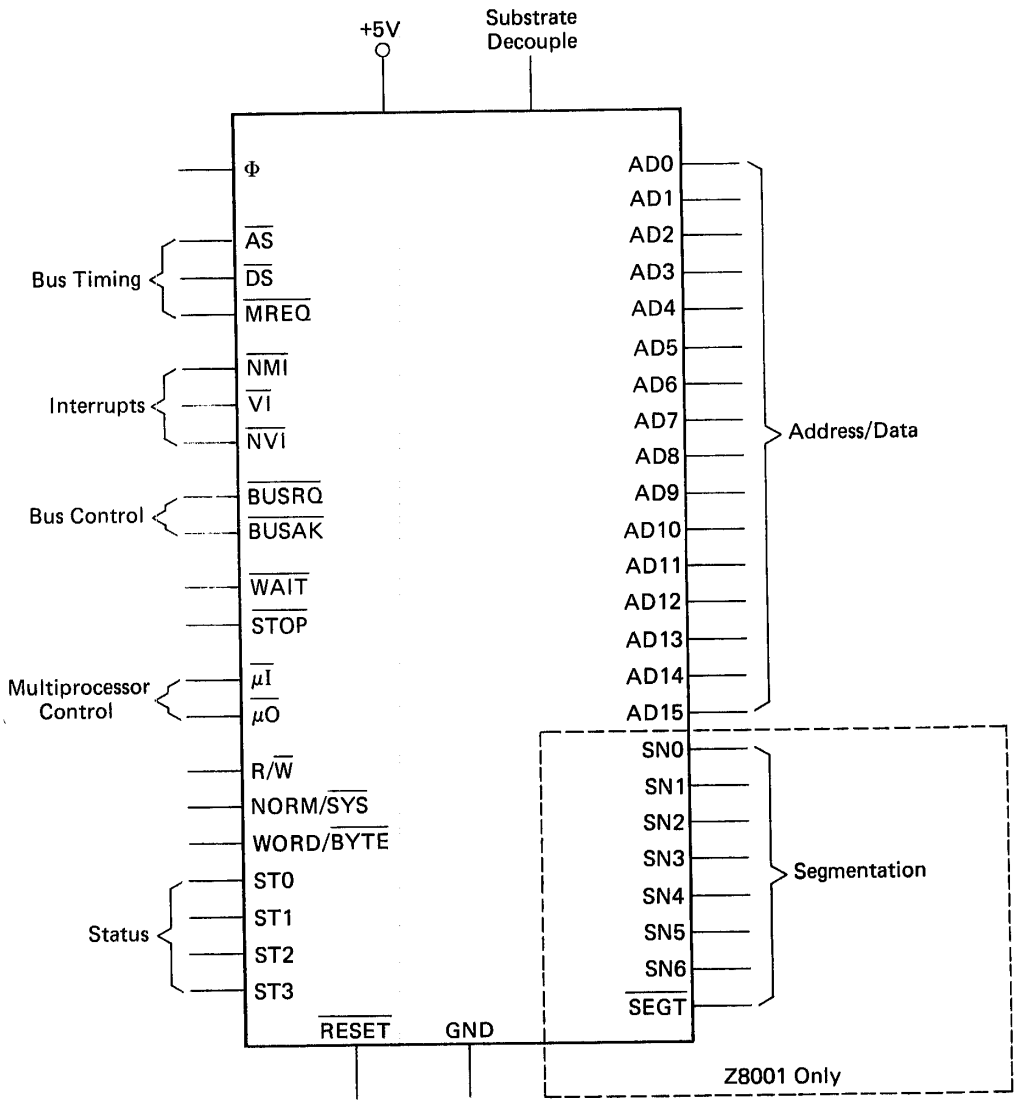


Figure 3-1

$\overline{DS}$ The data strobe line is an output indicating that the Z8001 is using AD0-AD15 for data transfer. During a read operation, the Z8001 has accepted data when the $\overline{DS}$ line makes the transition from low-to-high. During a write operation, the Z8001 sends the $\overline{DS}$ line low when data is stable on AD0-AD15.

$\overline{AS}$ The address strobe line is used to indicate that

AD0-AD15 are being used as address lines. The Z8001 status lines (ST0-ST3) indicate whether the Z8001 is addressing memory or I/O. The address presented by the Z8001 is stable when the $\overline{AS}$ line makes a low-to-high transition.

R/$\overline{W}$ — The R/$\overline{W}$ (read/write) line indicates the direction of data flow. When the R/$\overline{W}$ line is asserted high, a read operation is taking place. When the R/$\overline{W}$ line is asserted low, a write operation is taking place. The R/$\overline{W}$ line is asserted at the same time as the $\overline{AS}$ line and remains asserted during the entire read or write operation.

$\overline{MREQ}$ — The $\overline{MREQ}$ (memory request) line is an output used to indicate that a Z8001 memory transaction is taking place. $\overline{MREQ}$ is asserted when the $\overline{AS}$ line makes its low-to-high transition, indicating a stable address exists on AD0-AD15.

B/$\overline{W}$ — The B/$\overline{W}$ (byte/word) line is used to indicate whether a byte (8-bit) or word (16-bit) addressing mode is being used. When the B/$\overline{W}$ line is asserted low, the Z8001 is addressing memory in a "word-aligned" fashion. That is, a 16-bit address is placed on the address bus; however, bit AD0 is held in a low state, allowing only even addresses (and their associated data words) to be accessed. When the B/$\overline{W}$ line is asserted high, AD0 determines which byte of a 16-bit word is being addressed.

$\overline{WAIT}$ — The $\overline{WAIT}$ line is asserted by memory or I/O logic when they are not yet ready to perform a data transaction. The Z8001 "stretches" the the current data transaction by going into a wait state. Wait states are repeated until the $\overline{WAIT}$ line goes high.

$\overline{STOP}$ — The $\overline{STOP}$ line is used for single-cycle operations. The use of $\overline{STOP}$ is described later in this chapter.

$\overline{BUSRQ}$-$\overline{BUSAK}$ — The bus request ($\overline{BUSRQ}$) and bus acknowledge ($\overline{BUSAK}$) lines allow coordinated transfer of bus control between the Z8001 and another device on the bus. $\overline{BUSRQ}$ can be asserted by another device at any time because the Z8001 handles $\overline{BUSRQ}$ in an asynchronous fashion.

When the Z8001 is prepared to relinquish the bus, the $\overline{BUSAK}$ line is asserted, and the AD0-AD15, $\overline{AS}$, $\overline{DS}$, B/$\overline{W}$, R/$\overline{W}$, N/$\overline{S}$, ST0-ST3, and

$\overline{MREQ}$ lines are switched to their high-impedance state.

After the Z8001 asserts the $\overline{BUSAK}$ line, the bus requesting device must hold the $\overline{BUSRQ}$ line low all the time it is using the bus. The Z8001 monitors the state of the $\overline{BUSRQ}$ line and reasserts control of the bus as soon as the $\overline{BUSRQ}$ line goes high. This means the requesting device must have all its lines in a high-impedance state again before releasing $\overline{BUSRQ}$. When $\overline{BUSRQ}$ goes high, the Z8001 sends the $\overline{BUSAK}$ line high and takes back bus control.

ST0-ST3 — The status line indicate the current state of the Z8001. Table 3-1 lists the outputs of ST0-ST3.

Table 3-1
ST0-ST3 Definitions

ST3	ST2	ST1	ST0	Definition
0	0	0	0	Internal Operation
0	0	0	1	Memory Refresh
0	0	1	0	Normal I/O Transaction
0	0	1	1	Special I/O Transaction
0	1	0	0	Segment Trap Acknowledge
0	1	0	1	Non-Maskable Interrupt Acknowledge
0	1	1	0	Non-Vectored Interrupt Acknowledge
0	1	1	1	Vectored Interrupt Acknowledge
1	0	0	0	Memory Transaction for Operand
1	0	0	1	Memory Transaction for Stack
1	0	1	0	Reserved
1	0	1	1	Reserved
1	1	0	0	Memory Transaction for Instruction Fetch (subsequent word)
1	1	0	1	Memory Transaction for Instruction Fetch (first word)
1	1	1	0	Reserved
1	1	1	1	Reserved

N/$\overline{S}$ — The normal/system line is an output indicating whether the Z8001 is operating in normal mode or system mode. The output state of this line is derived from the Z8001 Flag and Control Word.

$\overline{NMI}$ — A non-maskable interrupt causes the Z8001 to issue an acknowledge on ST0-ST3 and then enter the $\overline{NMI}$ sequence. The Z8001 reacts to the high-to-low transition of the $\overline{NMI}$ line.

$\overline{VI}$ — The $\overline{VI}$ input is the vectored interrupt request input to the Z8000. When the $\overline{VI}$ line is low, another device is requesting service. If vectored interrupts are enabled in the Flag and Control Word, the Z8001 first responds with the vectored interrupt acknowledge on ST0-ST3 and then begins the interrupt sequence. The $\overline{VI}$ input can be driven low at any time and is held low until acknowledged.

$\overline{NVI}$ — The non-vectored interrupt input line is handled in much the same way as the $\overline{VI}$ line. If non-vectored interrupts are enabled in the Flag and Control Word, the Z8001 acknowledges the interrupt and begins the interrupt sequence. The $\overline{NVI}$ line should be held low until acknowledged.

$\overline{\mu I}$-$\overline{\mu O}$ — The $\overline{\mu I}$ input is used by other devices to request resources, while the $\overline{\mu O}$ output indicates an acknowledgment of the request. $\overline{\mu I}$ and $\overline{\mu O}$ are discussed in more detail later in this chapter.

$\overline{RESET}$ — The reset line is the master reset for the Z8001. When $\overline{RESET}$ goes low, the Z8001 begins an initialization process as described later in this chapter.

CLK — The clock input accepts a single-phase TTL clock.

DECOUPLE — The DECOUPLE pin is attached to the Z8001 substrate bias generator and should not be used.

SN0-SN6 — The segment pins of the Z8001 indicate which of 128 memory segments the Z8001 is accessing. SN0 is the least significant segment line.

$\overline{SEGT}$ — The segment trap line is, in effect, a segment trap request. If the $\overline{SEGT}$ line goes low, the Z8001 responds with a segment trap acknowledge on ST0-ST3 and then begins a trap sequence. The $\overline{SEGT}$ input has priority over all the interrupts.

THE Z8002 PIN-OUT

The Z8002 is a 40-pin device. Of those fourty pins, fifteen are used in address and data manipulation, while the remainder serve a control function. Timing requirements for the Z8002 are exactly the same as for the Z8001. See the timing

diagrams later in this chapter.

AD0-AD15 — Address and data information are multiplexed over these lines. AD0-AD15 are tri-state lines and are used for all memory and I/O transactions. The lines are positive true (1 = HIGH, 0 = LOW). Data transactions are indicated when the $\overline{DS}$ (data strobe) line is asserted, and address transactions are indicated when the $\overline{AS}$ (address strobe) line is asserted.

$\overline{DS}$ — The data strobe line is an output indicating that the Z8002 is using AD0-AD15 for data transfer. During a read operation, the Z8002 has accepted data when the $\overline{DS}$ line makes the transition from low-to-high. During a write operation, the Z8002 sends the $\overline{DS}$ line low when data is stable on AD0-AD15.

$\overline{AS}$ — The address strobe line is used to indicate that AD0-AD15 are being used as address lines. The Z8002 status lines (ST0-ST3) indicate whether the Z8002 is addressing memory or I/O. The address presented by the Z8002 is stable when the $\overline{AS}$ line makes a low-to-high transition.

R/$\overline{W}$ — The R/$\overline{W}$ (read/write) line indicates the direction of data flow. When the R/$\overline{W}$ line is asserted high, a read operation is taking place. When the R/$\overline{W}$ line is asserted low, a write operation is taking place. The R/$\overline{W}$ line is asserted at the same time as the $\overline{AS}$ line and remains asserted during the entire read or write operation.

$\overline{MREQ}$ — The $\overline{MREQ}$ (memory request) line is an output used to indicate that a Z8002 memory transaction is taking place. $\overline{MREQ}$ is asserted when the $\overline{AS}$ line makes its low-to-high transition, indicating a stable address exists on AD0-AD15.

B/$\overline{W}$ — The B/$\overline{W}$ (byte/word) line is used to indicate whether a byte (8-bit) or word (16-bit) addressing mode is being used. When the B/$\overline{W}$ line is asserted low, the Z8002 is addressing memory in a "word-aligned" fashion. That is, a 16-bit address is placed on the address bus; however, bit AD0 is held in a low state, allowing only even addresses (and their associated data **words**) to be accessed. When the B/$\overline{W}$ line is asserted high, AD0 determines which **byte** of a 16-bit word is being addressed.

$\overline{WAIT}$ — The $\overline{WAIT}$ line is asserted by memory or I/O logic when they are not yet ready to perform a data transaction. The Z8002 "stretches" the current data transaction by going into a wait state.

$\overline{STOP}$ — The $\overline{STOP}$ line is used for single-cycle operations. The use of $\overline{STOP}$ is described later in this chapter.

$\overline{BUSRQ}$-$\overline{BUSAK}$ — The bus request ($\overline{BUSRQ}$) and bus acknowledge ($\overline{BUSAK}$) lines allow coordinated transfer of bus control between the Z8002 and another device on the bus. $\overline{BUSRQ}$ can be asserted by another device at any time because the Z8002 handles $\overline{BUSRQ}$ in an asynchronous fashion.

When the Z8002 is prepared to relinquish the bus, the $\overline{BUSAK}$ line is asserted, and the AD0-AD15, $\overline{AS}$, $\overline{DS}$, B/$\overline{W}$, R/$\overline{W}$, N/$\overline{S}$, ST0-ST3, and $\overline{MREQ}$ lines are switched to their high-impedance state.

After the Z8002 asserts the $\overline{BUSAK}$ line, the bus requesting device must hold the $\overline{BUSRQ}$ line low all the time it is using the bus. The Z8002 monitors the state of the $\overline{BUSRQ}$ line and reasserts control of the bus as soon as the $\overline{BUSRQ}$ line goes high. This means the requesting device must have all its lines in a high-impedance state again before releasing $\overline{BUSRQ}$. When $\overline{BUSRQ}$ goes high the Z8002 sends the $\overline{BUSAK}$ line high, and takes back bus control.

ST0-ST3 — The status lines indicate the current state of the Z8002. Table 3-1 lists the outputs of ST0-ST3.

N/$\overline{S}$ — The normal/system line is an output indicating whether the Z8002 is operating in normal mode or system mode. The output state of this line is derived from the Z8002 Flag and Control Word.

$\overline{NMI}$ — A non-maskable interrupt causes the Z8002 to issue an acknowledge on ST0-ST3 and then enter the $\overline{NMI}$ sequence. The Z8002 reacts to the high-to-low transition of the $\overline{NMI}$ line.

$\overline{VI}$ — The $\overline{VI}$ input is the vectored interrupt request input to the Z8000. When the $\overline{VI}$ line is low, another devices is requesting service. If vectored interrupts are enabled in the Flag and Control Word, the Z8002 first responds with the vectored interrupt acknowledge on ST0-ST3 and then begins the interrupt sequence. The $\overline{VI}$ input

can be driven low at any time and is held low until acknowledged.

$\overline{NVI}$ — The non-vectored interrupt input line is handled in much the same way as the $\overline{VI}$ line. If non-vectored interrupts are enabled in the Flag and Control Word, the Z8002 acknowledges the interrupt and begins the interrupt sequence. The $\overline{NVI}$ line should be held low until acknowledged.

$\overline{\mu I}$-$\overline{\mu O}$ — The $\overline{\mu I}$ input is used by other devices to request resources, while the $\overline{\mu O}$ output indicates an acknowledgment of the request. $\overline{\mu I}$ and $\overline{\mu O}$ are discussed in more detail later in this chapter.

$\overline{RESET}$ — The reset line is the master reset for the Z8002. When $\overline{RESET}$ goes low, the Z8002 begins an initialization process as described later in this chapter.

CLK — The clock input accepts a single-phase TTL clock.

DECOUPLE — The DECOUPLE pin is attached to the Z8002 substrate bias generator and should not be used.

Z8000 TIMING

Like all microprocessors, the Z8000 devices have specific timing requirements. And since we're interested in Z8000 timing in terms of connecting other devices to the Z8000, we will look at timing from that viewpoint. First, we examine Z8000 initialization, then the timing requirements during each major operation.

Apart from initialization, the Z8000 reports all its activities through the status lines (ST0-ST3). Therefore, the states of the status lines are used as pointers to Z8000 timing.

Initialization

The Z8000 begins its initialization process whenever the $\overline{RESET}$ line is pulled low. $\overline{RESET}$ is usually asserted by power-up reset logic; typically, a resistor-capacitor network that forces reset low a few milliseconds following application of power. This is sometimes called a "cold" start. Additionally, most applications require a master reset capability that doesn't require the power to be turned off and

then on. Usually, some piece of logic is used to pull down the RESET line under extraordinary circumstances, causing the Z8000 to re-initialize. This is sometimes called a "warm" start.

The initialization process is shown in Figure 3-2.

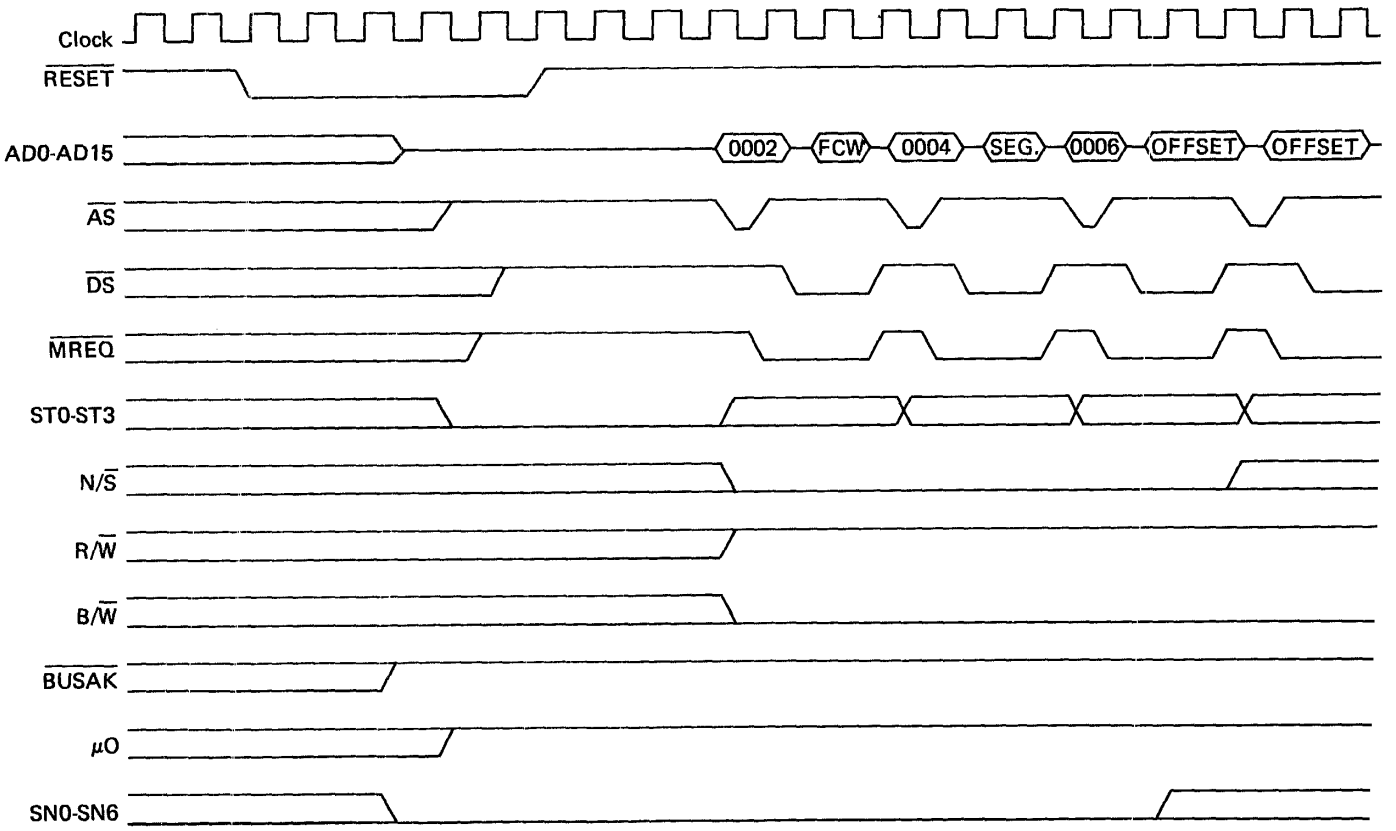


Figure 3-2 Initialization timing.

When the RESET line goes low, the following sequence of events occurs within five clock cycles:

- o AD0-AD15 move to their high-impedance state.
- o The AS, DS, MREQ, BUSAK, and μO outputs go to their high state.
- o Status lines ST0-ST3 go to a low state.
- o Memory refresh is disabled.
- o If a warm start is occurring, the states of the R/W, B/W, and N/S lines remain in the same condition as they were before reset. If a cold start occurs, the states of these lines are undefined.

o In the Z8001, the SN0-SN6 segment lines go low.

Three clock periods after the $\overline{\text{RESET}}$ line goes back high, the Z8000 fetches its status information from memory. This occurs in the Z8001 in a slightly different manner than it does in the Z8002.

Z8001. The Z8001 performs three 16-bit memory reads from segment 00: location 0002 is read into the Flag and Control Word; location 0004 is read into the program counter segment register; and location 0006 is read into the program counter offset register.

Z8002. The Z8002 performs two 16-bit memory reads: location 0002 is read into the Flag and Control Word; and location 0004 is read into the program counter register.

This completes the Z8000 initialization cycle. Following initialization, the Z8000 will fetch the first instruction pointed to by the program counters, and begin execution.

General Operation Timing

The easiest way to describe and understand Z8000 timing is to relate that timing to the states of the ST0-ST3 status lines. Refer back to the status line decoding in Table 3-1 while reading the following text. Each operation title is followed by the state of the status lines.

Internal Operations (0000). When the Z8000 is performing an internal operation, ST0-ST3 all go low. Figure 3-3 illustrates a timing cycle for an internal operation. Internal operations are three clock cycles long, and the cycles are labeled T1, T2, and T3.

During T1, the beginning of the internal operation is marked by $\overline{\text{AS}}$ making a high-to-low transition. At the same time, the status lines all go low. The $\overline{\text{MREQ}}$, $\overline{\text{DS}}$, and R/$\overline{\text{W}}$ outputs are all high, while the N/$\overline{\text{S}}$, and SN0-SN6 outputs remain at the same level as in the previous machine cycle. During T1 the Z8000 drives AD0-AD15 with unspecified information.

During T2, AD0-AD15 are set to their high-impedance state and stay there for the remainder of the operation.

For the entire internal operation, the B/$\overline{\text{W}}$ output generates unspecified information, and the Z8000 ignores any $\overline{\text{WAIT}}$ inputs.

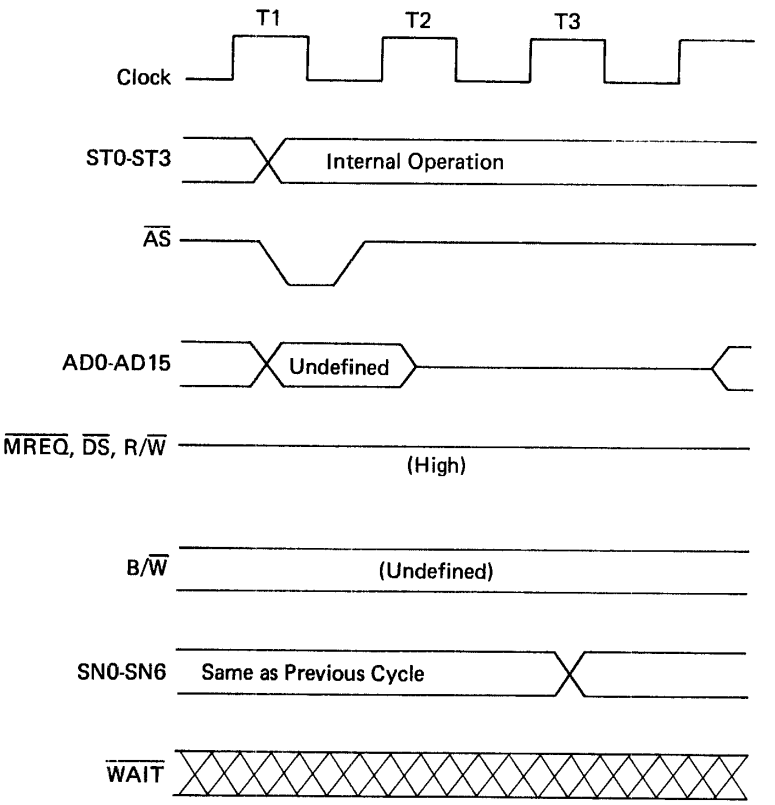


Figure 3-3 Internal operation timing.

Memory Refresh (0001). A memory refresh cycle is three clock periods long, as shown in Figure 3-4.

During T1, the beginning of the refresh cycle is marked by AS making a high-to-low transition. At the same time, the status lines indicate a refresh cycle is occuring. AD0-AD8 contain the refresh address. The contents of AD9-AD15 are unspecified.

During T2, AD0-AD15 go into their high-impedance state and stay there for the remainder of the refresh cycle.

During the entire refresh operation, the DS output remains high, while the R/W, B/W, SN0-SN6, and N/S outputs stay at the same level as in the previous machine cycle. The MREQ output is held low during most of the refresh cycle. During refresh, the Z8000 ignores any WAIT inputs.

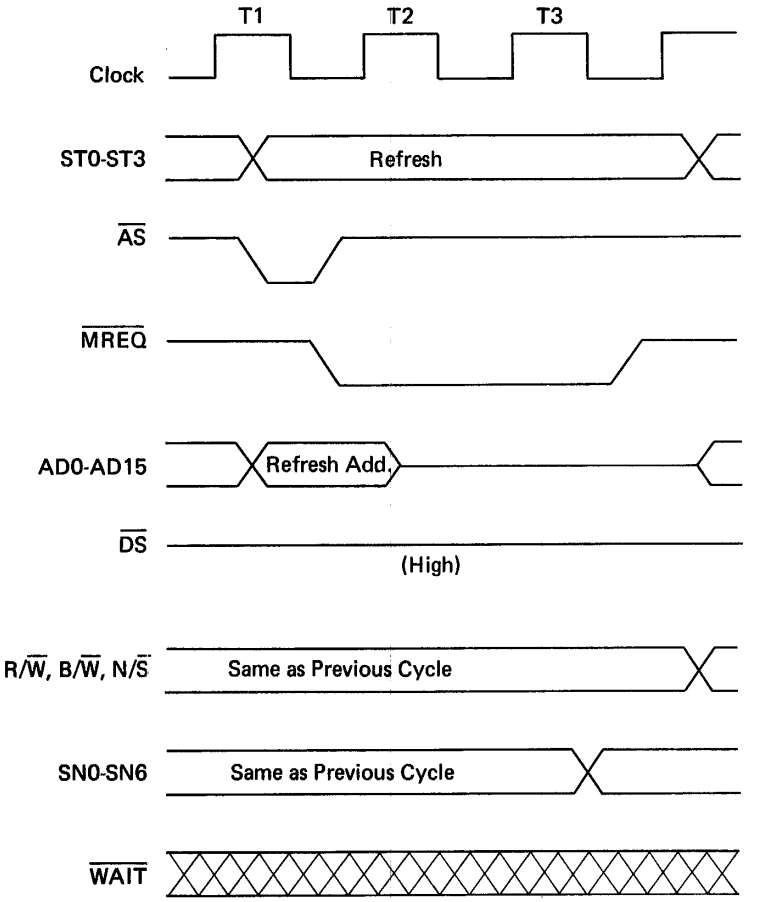


Figure 3-4 Refresh timing.

Memory Transactions (1000-1001-1100-1101). There are four possible memory transactions in the Z8000: read or write an operand, read or write stack information, fetch the first word of an instruction, and fetch subsequent words of an instruction.

Memory transactions involving operands or stack information could involve words or bytes, and so the following descriptions and the timing diagram in Figure 3-5 define the B/$\overline{W}$, N/$\overline{S}$, and ST0-ST3 lines as being unspecified.

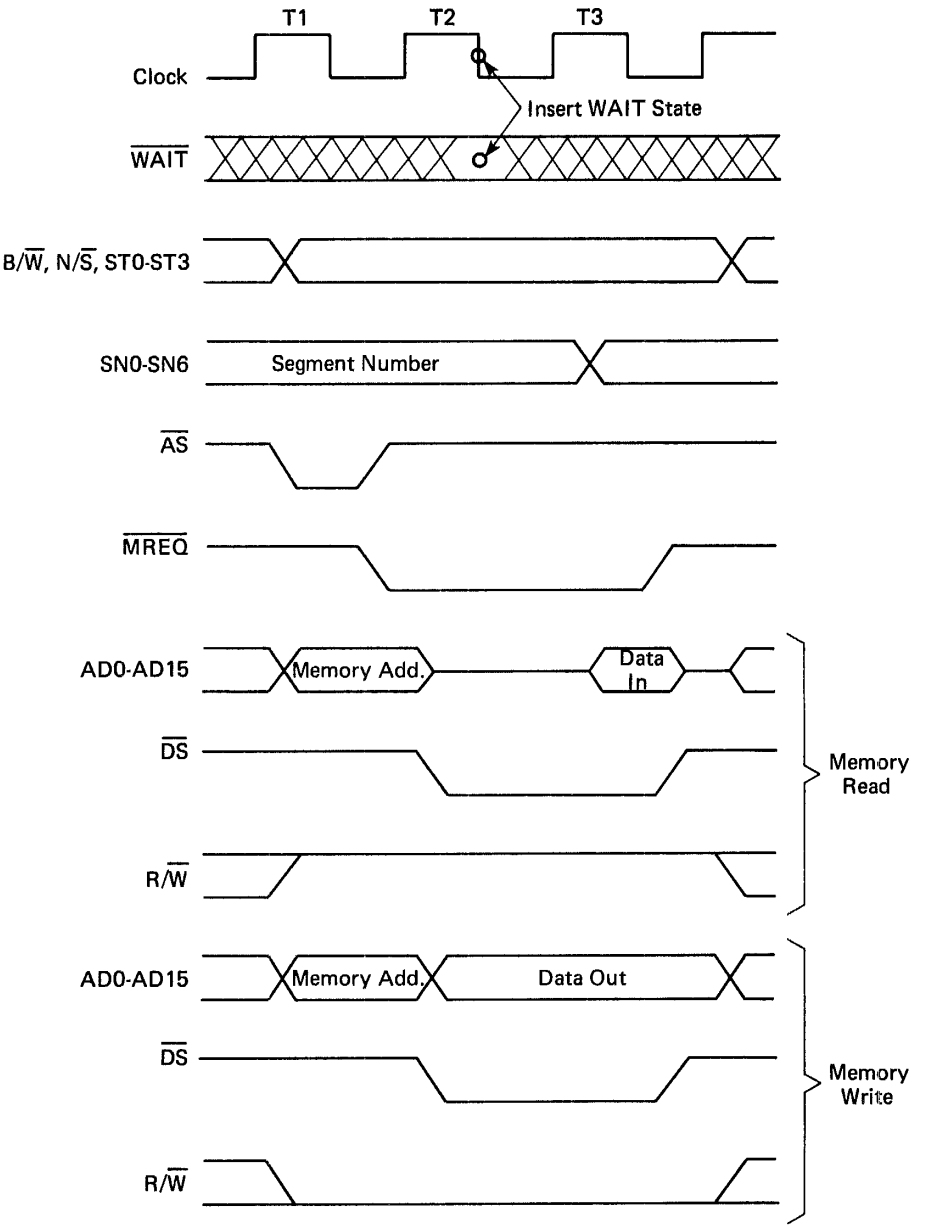


Figure 3-5 Memory transaction timing.

The memory transaction is marked by the $\overline{AS}$ line going low. $\overline{AS}$ goes back high at the end of T1, and $\overline{MREQ}$ is asserted indicating a memory operation. $\overline{MREQ}$ remains asserted throughout the remainder of the memory transaction.

During a read or write operation, the memory address is valid at the time $\overline{AS}$ goes low and becomes invalid during the last half of T2. During a read operation, incoming data is

assumed valid during T3. During a write operation, data is valid early on in T2 and throughout T3.

In a Z8001, segment lines SN0-SN6 are valid before the transaction begins and remain valid until the beginning of cycle T3.

Memory transactions typically take three clock cycles, although the number of cycles can increase if wait states are requested by the $\overline{WAIT}$ line. Wait states are always inserted in cycle T2 of the operation. The Z8000 samples the $\overline{WAIT}$ line during T2 and inserts a wait state if $\overline{WAIT}$ is low. $\overline{WAIT}$ is checked at the completion of every cycle in which a wait state was inserted. Wait states are inserted until $\overline{WAIT}$ goes high again, at which time the memory transaction is completed.

I/O Transactions (0010-0011). The Z8000 provides two I/O spaces: normal and special. I/O is addressed by a 16-bit port address and typically requires three clock cycles **plus** one wait state (Twa). In I/O transactions, a single wait state is always included. Additional wait states can be requested by pulling the $\overline{WAIT}$ line low. Timing for I/O transactions is shown in Figure 3-6.

As with memory operations, the $\overline{AS}$ line goes low to mark the beginning of the I/O transaction. When $\overline{AS}$ makes the high-to-low transition, the Z8000 places the I/O port address on AD0-AD15. The N/$\overline{S}$ line is low throughout the transaction (recall that all I/O instructions are system instructions). The $\overline{MREQ}$ line remains high throughout the transaction.

The B/$\overline{W}$ line can be either high or low, depending upon the I/O instruction being used.

During an I/O read, the Z8000 assumes that incoming data is valid during the middle of the T3 cycle. During an I/O write, the Z8000 provides valid data from the beginning of T2 through T3.

Segment Trap and Interrupt Acknowledge (0100-0111). Because of its priority, an $\overline{NMI}$ is stored internally in the Z8000. However, $\overline{VI}$, $\overline{NVI}$, and the $\overline{SEGT}$ states are not stored. During T3 of a normal instruction cycle, the Z8000 examines the $\overline{NMI}$ latch, and the states of the $\overline{VI}$, $\overline{NVI}$, and $\overline{SEGT}$ lines.

If an interrupt or trap is detected, the Z8000 enters a dummy instruction fetch cycle, as shown in Figure 3-7. The dummy fetch cycle is used to decrement the stack pointer so that the Z8000 can return to the proper location following completion of the interrupt servicing routine. The dummy fetch cycle is seven clock cycles long.

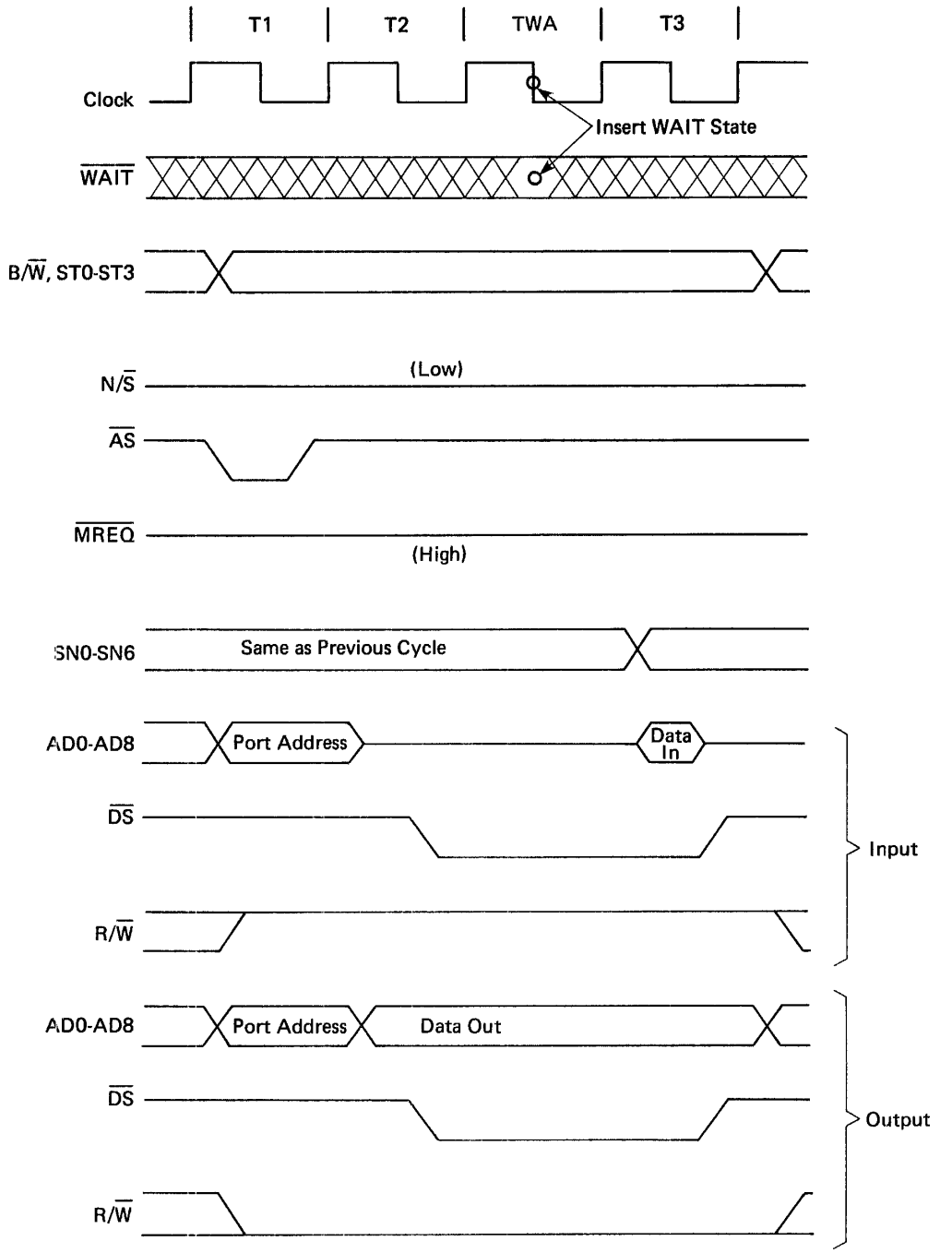


Figure 3-6 I/O transaction timing.

The actual acknowledge cycle begins at the end of the dummy fetch cycle. The beginning of the acknowledge cycle is marked by $\overline{AS}$ making a high-to-low transition. The $\overline{MREQ}$ line remains high throughout the acknowledge cycle, as does the R/$\overline{W}$ line. The B/$\overline{W}$ line goes low during the period of the acknowledge cycle.

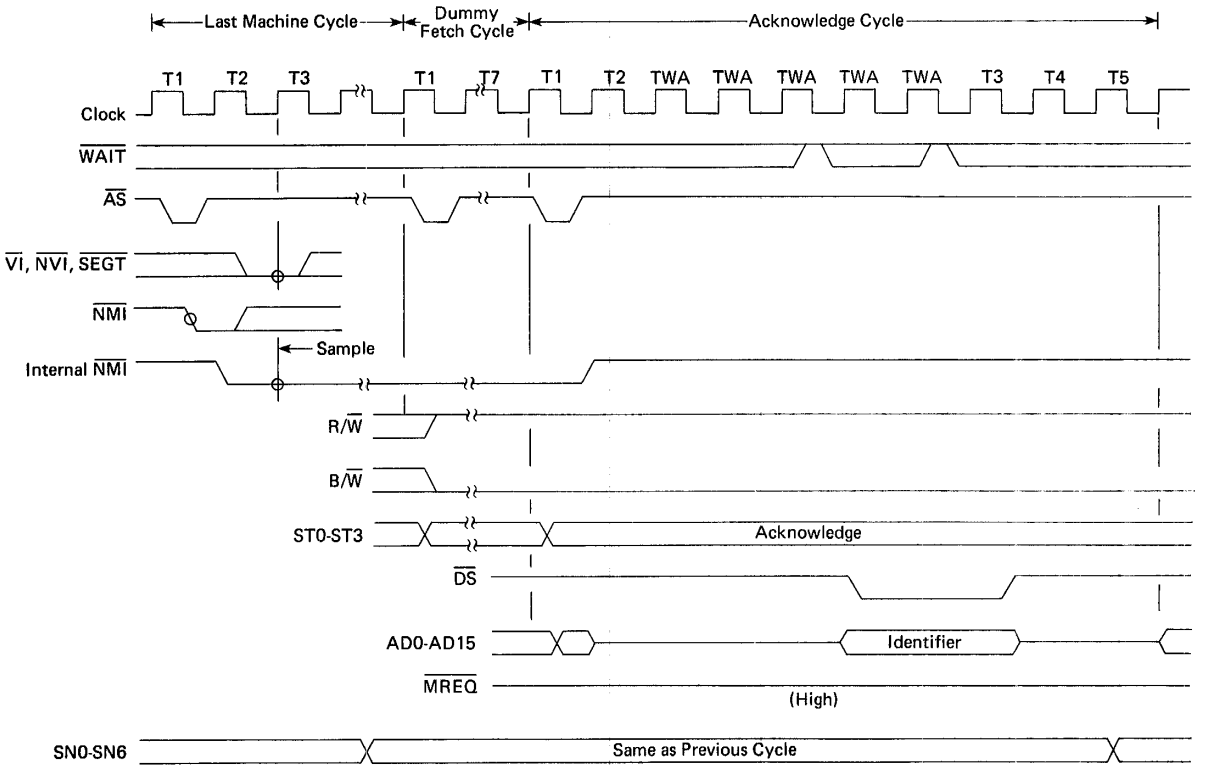


Figure 3-7

During T1, lines AD0-AD15 contain unspecified information. At the beginning of T2, AD0-AD15 go to their high-impedance state, and the $\overline{NMI}$ latch is cleared.

At the end of T2, the Z8000 automatically generates three wait states (Twa). During the third Twa, the Z8000 samples the $\overline{WAIT}$ input line. If $\overline{WAIT}$ is asserted, additional wait states are generated until $\overline{WAIT}$ goes high. After $\overline{WAIT}$ goes high, the $\overline{DS}$ line goes low, and an additional two wait states are generated. So the Z8000 will always generate a minimum of five wait states.

When the $\overline{DS}$ line goes low, the Z8000 will accept a 16-bit identifier word from the interrupting device. The $\overline{DS}$ line

then goes high at the beginning of T3. At the beginning of T3, the interrupting device should remove the identifier from the bus. Cycles T4 and T5 complete the acknowledge cycle.

Following the acknowledge cycle, the Z8000 performs memory transactions to save processor status on the stack.

Bus Request/Bus Acknowledge.

The Z8000 bus request facility (BUSRQ/BUSAK) is typically used in an environment where another device needs access to the Z8000 bus. For example, in direct memory access (DMA) applications, another processor might need to access Z8000 RAM.

The timing for a bus request/acknowledge cycle is shown in Figure 3-8. A bus request may be made during any machine cycle, except while the TSET and TSETB instructions are executing. When BUSRQ goes low, an internal BUSRQ signal is sychnronized with T1. The internal BUSRQ signal is marked, as usual, by AS being asserted.

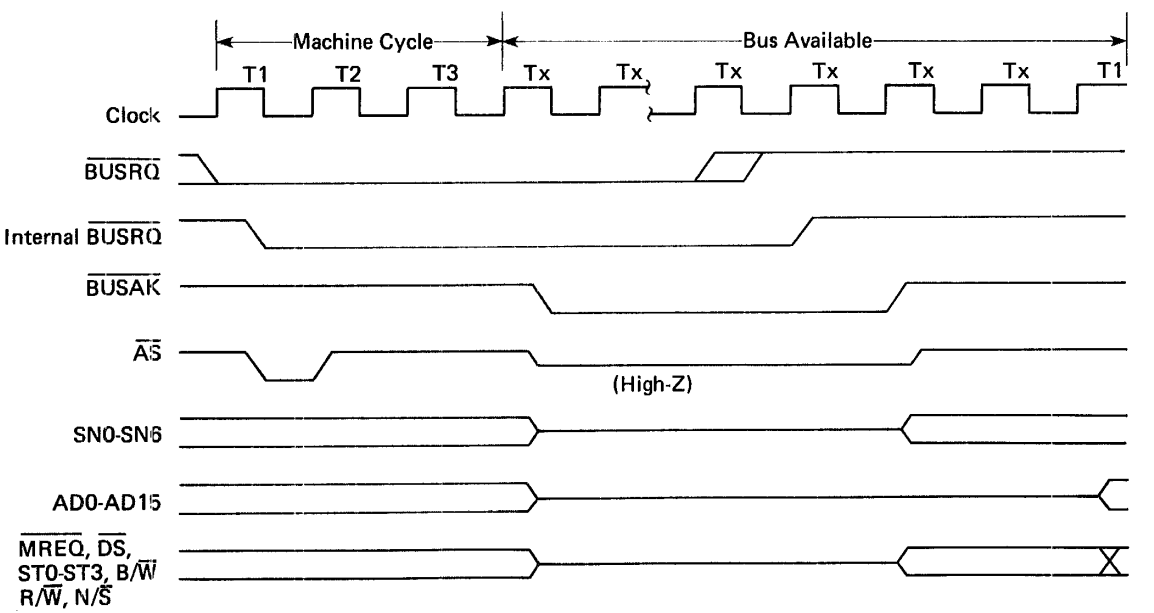


Figure 3-8

When the current machine cycle is completed, the Z8000 issues a BUSAK. When BUSAK goes low, AD0-AD15, AS, DS, MREQ, ST0-ST3, B/W, R/W, SN0-SN6 (Z8001 only), and N/S all go into a high-impedance state.

BUSRQ must remain low until the requesting device has completed all transactions. When BUSRQ goes back high, BUSAK is removed about one clock cycle later.

Single-Cycle Timing

The Z8000 can be single-stepped through a routine by pulling the $\overline{STOP}$ input low. Single-cycle timing is illustrated in Figure 3-9.

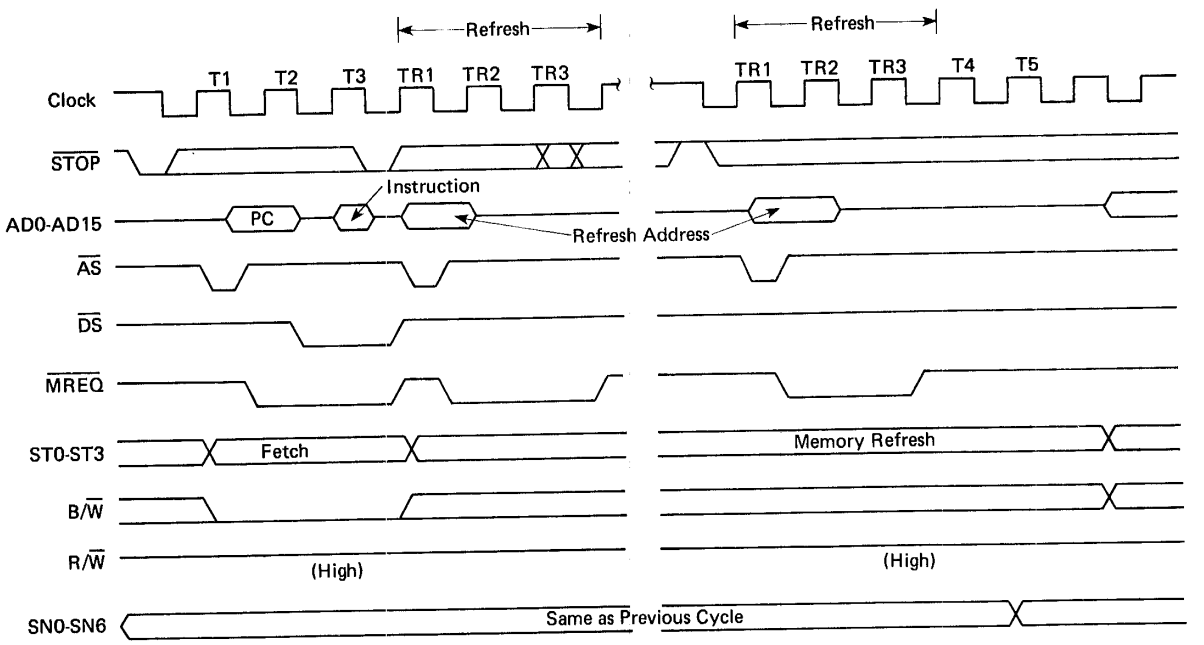


Figure 3-9

The $\overline{STOP}$ input line is sampled on the last high-to-low transition of the clock just prior to a new instruction fetch. If the $\overline{STOP}$ line is low, the Z8000 will begin a memory refresh cycle. The Z8000 again samples the $\overline{STOP}$ line at T3 of the refresh cycle. If the $\overline{STOP}$ line is still low, another refresh cycle is started. The Z8000 will keep doing refresh cycles until the $\overline{STOP}$ line goes high.

When $\overline{STOP}$ goes high, the Z8000 inserts dummy states T4 and T5 to complete the fetch cycle and then resumes execution. The Z8000 will insert refresh cycles while stopped even if refresh is disabled.

Z8000 Characteristics

The following tables describe the ac characteristics of the Z8001/Z8002 and Z8001A/Z8002A. The "A" devices run at a higher clock speed. Included is a composite timing diagram that does **not** illustrate actual timing sequences, but instead

lets you look at the detailed timing relationships between individual signal edges. The composite timing diagram is shown in Figure 3-10, while Z8000 ac characteristics are listed in Tables 3-2 and 3-3. Following the ac characteristics are dc and physical characteristics of the Z8000.

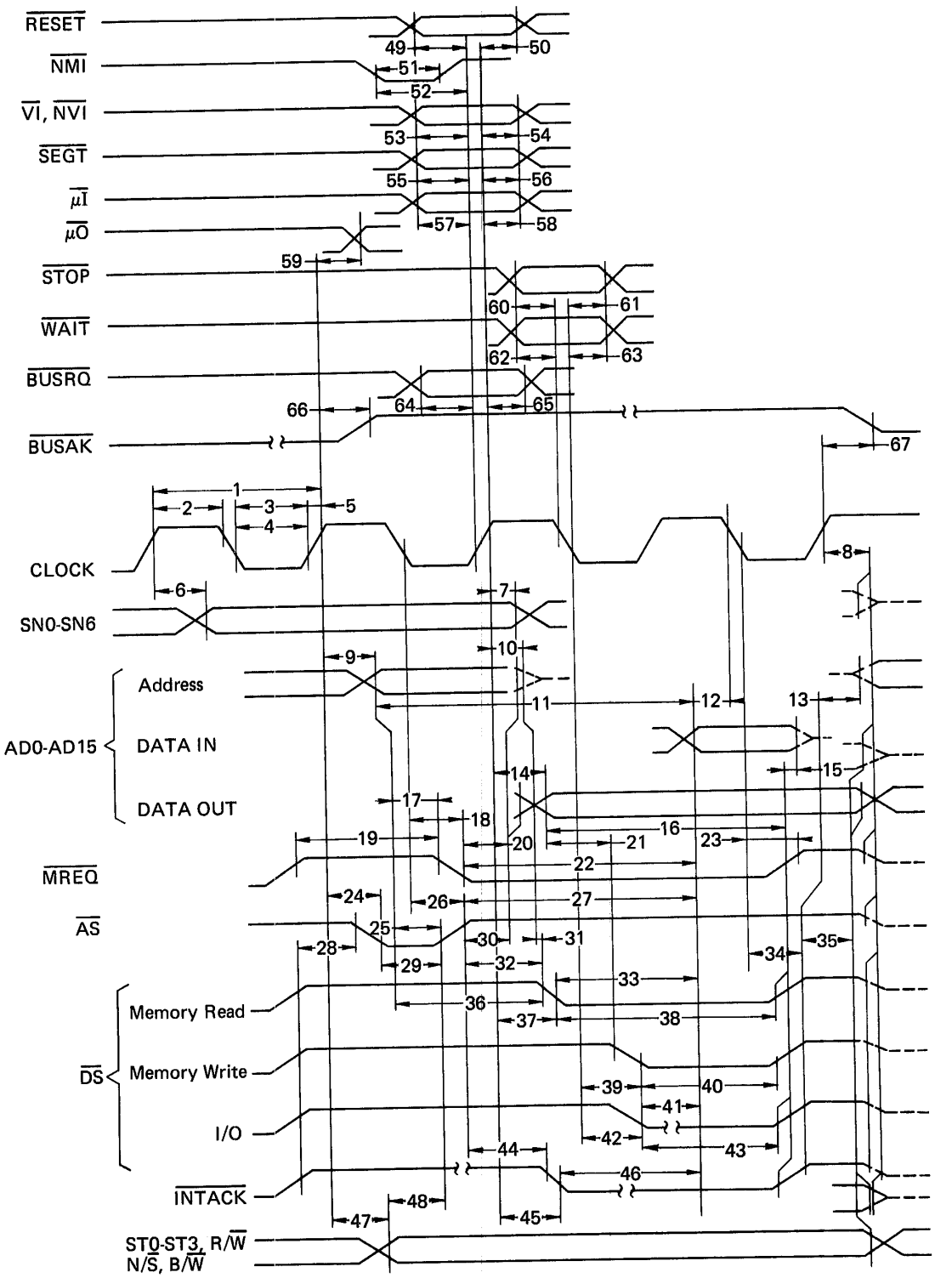

RESET
NMI
VI, NVI
SEGT
μI
μO
STOP
WAIT
BUSRQ
BUSAK
CLOCK
SN0-SN6
AD0-AD15
Address
DATA IN
DATA OUT
MREQ
AS
DS
Memory Read
Memory Write
I/O
INTACK
ST0-ST3, R/W
N/S, B/W
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67

Figure 3-10

Table 3-2
Z8001/Z8001A ac Characteristics
("A" timing is given in parentheses)
(All times are given in nanoseconds)

No.	Parameter	Description	Min	Max
1	TcC	Clock Cycle Time	250 (165)	2000 (2000)
2	TwCh	Clock Width (high)	105 (70)	2000 (2000)
3	TwCl	Clock Width (low)	105 (70)	2000 (2000)
4	TfC	Clock Fall Time		20 (10)
5	TrC	Clock Rise Time		20 (10)
6	TdC(SNv)	Clock ↑ to Segment Number Valid (50 pF) (Z8001 only)		130 (110)
7	TdC(SNn)	Clock ↑ to Segment Number Invalid (Z8001 only)	20 (10)	
8	TdC(Bz)	Clock ↑ to Bus Float		65 (55)
9	TdC(A)	Clock ↑ to Address Valid		100 (75)
10	TdC(Az)	Clock ↑ to Address Float		65 (55)
11	TdA(Dl)	Address Valid to Data In Required Valid	455 (305)	
12	TsDI(C)	Data In to Clock ↓ Set-up Time	50 (20)	
13	TdDS(A)	$\overline{DS}$ ↑ to Address Active	80 (40)	
14	TdC(DO)	Clock ↑ to Data Out Valid		100 (75)
15	ThDI(DS)	Data In to $\overline{DS}$ ↑ Hold Time	0 (0)	
16	TdDO(DS)	Data Out Valid to $\overline{DS}$ ↑ Delay	295 (195)	
17	TdA(MR)	Address Valid to $\overline{MREQ}$ ↓ Delay	55 (35)	
18	TdC(MR)	Clock ↓ to $\overline{MREQ}$ ↓ Delay		80 (70)
19	TwMRh	$\overline{MREQ}$ Width (high)	210 (135)	
20	TdMR(A)	$\overline{MREQ}$ ↓ to Address Not Active	70 (35)	

Table 3-2 (cont.)
Z8001/Z8001A ac Characteristics
("A" timing is given in parentheses)
(All times are given in nanoseconds)

No.	Parameter	Description	Min	Max
21	TdDO(DSW)	Data Out Valid to $\overline{DS}$ ↓ (write) Delay	55 (35)	
22	TdMR(DI)	$\overline{MREQ}$ ↓ to Data In Required Valid	350 (225)	
23	TdC(MR)	Clock ↓ to $\overline{MREQ}$ ↑ Delay		80 (60)
24	TdC(ASf)	Clock ↑ to $\overline{AS}$ ↑ Delay		80 (60)
25	TdA(AS)	Address Valid to $\overline{AS}$ ↑ Delay	55 (35)	
26	TdC(ASr)	Clock ↓ to $\overline{AS}$ ↑ Delay		90 (80)
27	TdAS(DI)	$\overline{AS}$ ↑ to Data In Required Valid	340 (215)	
28	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	70 (35)	
29	TwAS	$\overline{AS}$ Width (low)	85 (55)	
30	TdAS(A)	$\overline{AS}$ ↑ to Address Not Active Delay	60 (30)	
31	TdAz(DSR)	Address Fload to $\overline{DS}$ (read) ↓ Delay	0 (0)	
32	TdAS(DSR)	$\overline{AS}$ ↑ to $\overline{DS}$ (read) ↓ Delay	70 (35)	
33	TdDSR(DI)	$\overline{DS}$ (read) ↓ to Data In Required Valid	185 (130)	
34	TdC(DSr)	Clock ↓ to $\overline{DS}$ ↑ Delay		70 (65)
35	TdDS(DO)	$\overline{DS}$ ↑ to Data Out and STATUS not Valid	75 (45)	
36	TdA(DSR)	Address Valid to $\overline{DS}$ (read) ↓ Delay	180 (110)	
37	TdC(DSR)	Clock ↑ to $\overline{DS}$ (read) ↓ Delay		120 (85)
38	TwDSR	$\overline{DS}$ (read) Width (low)	275 (185)	
39	TdC(DSW)	Clock ↓ to $\overline{DS}$ (write) ↓ Delay		95 (80)
40	TwDSW	$\overline{DS}$ (write) Width (low)	185 (110)	

Table 3-2 (cont.)
Z8001/Z8001A ac Characteristics
("A" timing is given in parentheses)
(All times are given in nanoseconds)

No.	Parameter	Description	Min	Max
41	TdDSI(DI)	$\overline{DS}$ (input) ↓ to Data In Required Valid	320 (200)	
42	TdC(DSf)	Clock ↓ to $\overline{DS}$ (I/O) ↓ Delay		120 (100)
43	TwDS	$\overline{DS}$ (I/O) Width (low)	400 (255)	
44	TdAS(DSA)	$\overline{AS}$ ↑ to $\overline{DS}$ (acknowledge) ↓ Delay	1065 (690)	
45	TdC(DSA)	Clock ↑ to $\overline{DS}$ (acknowledge) ↓ Delay		120 (85)
46	TdDSA(DI)	$\overline{DS}$ (acknowledge) ↓ to Data In Rqrd. Delay	435 (295)	
47	TdC(S)	Clock ↑ to Status Valid Delay		110 (85)
48	TdS(AS)	Status Valid to $\overline{AS}$ ↑ Delay	60 (30)	
49	TsR(C)	$\overline{RESET}$ to Clock ↑ Set-up Time	180 (70)	
50	ThR(C)	$\overline{RESET}$ to Clock ↑ Hold Time	0 (0)	
51	TwNMI	$\overline{NMI}$ Width (low)	100 (70)	
52	TsNMI(C)	$\overline{NMI}$ to Clock ↑ Set-up Time	140 (70)	
53	TsVI(C)	$\overline{VI}$,$\overline{NVI}$ to Clock ↑ Set-up Time	110 (50)	
54	ThVI(C)	$\overline{VI}$,$\overline{NVI}$ to Clock ↑ Hold Time	0 (0)	
55	TsSGT(C)	$\overline{SEGT}$ to Clock ↑ Set-up Time (Z8001 only)	70 (55)	
56	ThSGT(C)	$\overline{SEGT}$ to Clock ↑ Hold Time (Z8001 only)	0 (0)	
57	Tsu(C)	$\overline{\mu I}$ to Clock ↑ Set-up Time	180 (110)	
58	Thu(C)	$\overline{\mu I}$ to Clock ↑ Hold Time	0 (0)	
59	TdC(u0)	Clock ↑ to $\overline{\mu O}$ Delay		120 (85)
60	TsSTP(C)	$\overline{STOP}$ to Clock ↓ Set-up Time	140 (70)	

Table 3-2 (cont.)
Z8001/Z8001A ac Characteristics
("A" timing is given in parentheses)
(All times are given in nanoseconds)

No.	Parameter	Description	Min	Max
61	ThSTP(C)	$\overline{STOP}$ to Clock ↓ Hold Time	0 (0)	
62	TsWT(C)	$\overline{WAIT}$ to Clock ↓ Set-up Time	50 (30)	
63	ThWT(C)	$\overline{WAIT}$ to Clock ↓ Hold Time	0 (0)	
64	TsBRQ(C)	$\overline{BUSRQ}$ to Clock ↑ Set-up Time	90 (80)	
65	ThBRQ(C)	$\overline{BUSRQ}$ to Clock ↑ Hold Time	0 (0)	
66	TdC(BAKr)	Clock ↑ to $\overline{BUSAK}$ ↑ Delay		100 (75)
67	TdC(BAIf)	Clock ↑ to $\overline{BUSAK}$ ↓ Delay		100 (75)

Electrical Characteristics

Table 3-3
Z8000 Electrical Characteristics

Parameter	Description	Min	Max	Units
VCH	Clock Input High Voltage	Vcc-0.4	Vcc+0.3	Volts
VCL	Clock Input Low Voltage	-0.3	0.45	Volts
VIH	Input High Voltage	2.0	Vcc+0.3	Volts
VIL	Input Low Voltage	-0.3	0.8	Volts
VOH	Output High Voltage	2.4		Volts
VOL	Output Low Voltage		0.4	Volts
IIL	Input Leakage		+/- 10	A
IOL	Output Leakage		+/- 10	A
Icc	Vcc Supply Current		300	mA

Maximum Ratings

Voltages on all inputs and outputs with respect to GND	= -0.3V to +7.0V
Operating Ambient Temperature	= 0 to +70 C
Storage Temperature	= -65 to +150 C

4

BUILDING A MINIMUM SYSTEM

INTRODUCTION

At this point, we've discussed the Z8000 instruction set and Z8000 timing requirements. In order to pull it all together, let's start to build a Z8000-based computer in block diagram form.

If you're intimately familiar with the Z8000 and are using this book only for reference, you may not be interested in reading this chapter. In it, we apply the Z8000 to as realistic a situation as reasonable, without actually designing a Z8000-based system. Instead, we make a rough specification for our imaginary system and then see what it takes to make the Z8000 fit.

The imaginary system will be built in a "kernal-out" fashion. That is, the minimum requirements (the kernal) for Z8000 operation will be discussed first, and then the kernal will be built upon until we have a system that meets our specifications. Incidentally, the system will be neither specified nor designed with what are commonly called "good" design practices. Structured or "top down" design is absolutely critical when designing a real system. However, for the purposes of this chapter, we'll forgo structured design in favor of clear explanations of how the Z8000 works.

THE SPECIFICATION

For purposes of discussion, say that we have a requirement for a graphics system. That system should support up to four users with four graphics terminals. The system must provide all the normal requirements of a graphics computer, with zoom, rotate, and high resolution calculation abilities. In addition, the system must be able to drive a high-resolution plotter, and must provide mass storage capability with backup.

If we block out the system, it might look like Figure 4-1.

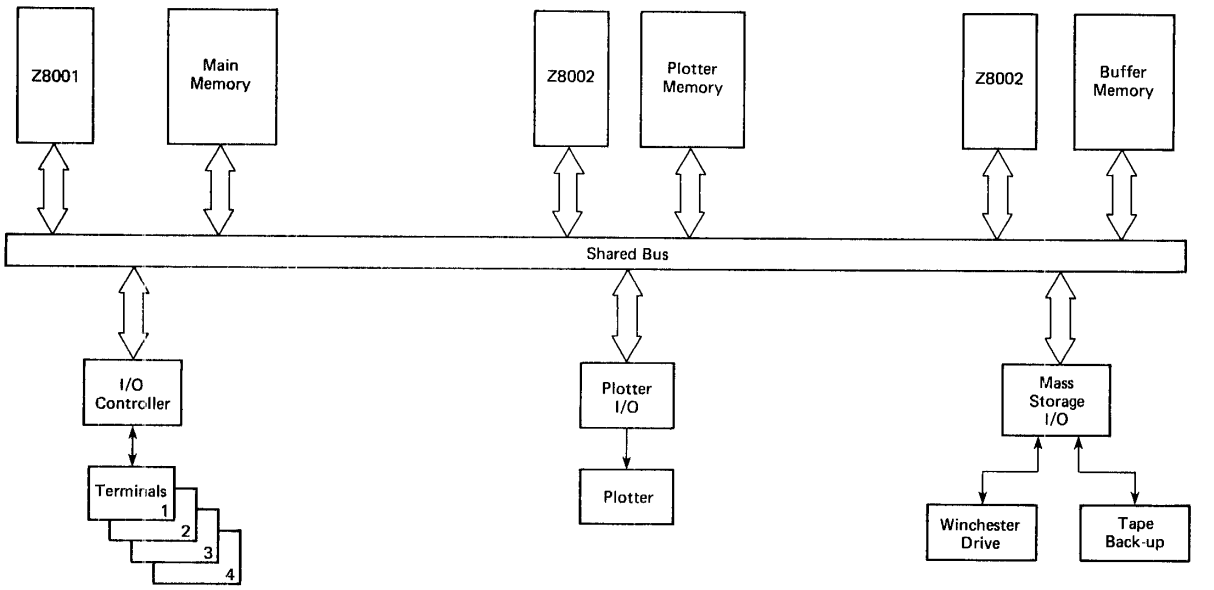


Figure 4-1

Theoretically, a single Z8001 could handle all the tasks

specified. However, performance would improve if the Z8001 were used strictly to handle the graphics computations, and I/O with the four graphics terminals. So we included two Z8002's: one to drive the plotter and another to handle mass storage and a printer. Admittedly, this might not be the most practicle way to build the system, but this design does lend itself to exploring the use of Z8000 devices in a multi-processor environment.

In the block diagram, notice that the Z8001 uses a large amount of read/write memory (RAM). A great deal of RAM is required to support four users simultaneously. Graphics computation and storage require a great deal of memory. Graphics work is memory intensive, to say the least.

Also notice in Figure 4-1 that the three Z8000 devices share a common bus. This feature is implemented to allow direct memory access (DMA) between the various processors. We would like any of the three to be able to access any other processor's memory, with protections built in, of course. We want to be able to send data to the plotter's Z8002 and to and from the mass storage Z8002. We should be able to request that an existing file on disc be dumped to the plotter or to the Z8001 for alteration or updating.

So, let's begin. First, we discuss the power supply for our system and then move on to the system clock, building outward from the kernal.

THE KERNAL

Power

The Z8000 devices themselves require a single +5 Vdc power source, capable of providing 300 mA of current to each. However, the system's memory and I/O logic will undoubtedly require much more than 900 mA and at other voltages. So, let's assume that the system will require +5 Vdc, +12 Vdc, and -12 Vdc, at 60 amps, 10 amps, and 5 amps, respectively. In real practice, these values would be calculated from the known power requirements of various system components.

Clock

Now that the system has a power supply, let's look at clocking requirements. The Z8000 devices require a single-phase TTL level clock. Let's assume that the processors should run at a nominal 4 MHz.

System specifications require that the clock circuit have an output with maximum rise and fall times of less than 20 ns (10 ns for the Z8000A). The clock output should be as

symmetrical as possible, having clock widths, both high and low, of 105 ns (70 ns for the Z8000A) minimum. The specifications also state that the clock must have a high input voltage ranging from +4.6 V to +5.3 V, and a low input voltage ranging from -0.3 V to +0.45 V. Nominally, we want the clock circuit to develop a waveform that looks like the one shown in Figure 4-2.

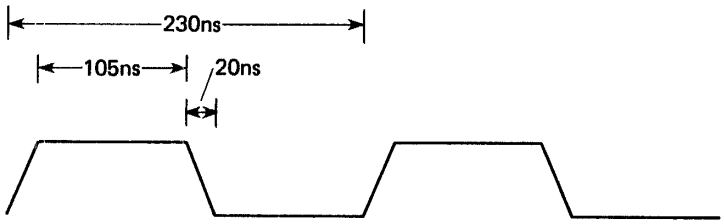


Figure 4-2 Clock timing.

We need a voltage swing on the clock output of 0 V to +5 V, with a rise time of about 5 ns and a high and low time of about 120 ns. From the block diagram level, our clock circuitry would require an 8 MHz crystal controlled oscillator with a rise-time enhancer to both keep the rise times at 5 ns and provide level adjustment to keep the voltage swing within specification. Figure 4-3 illustrates one possible circuit.

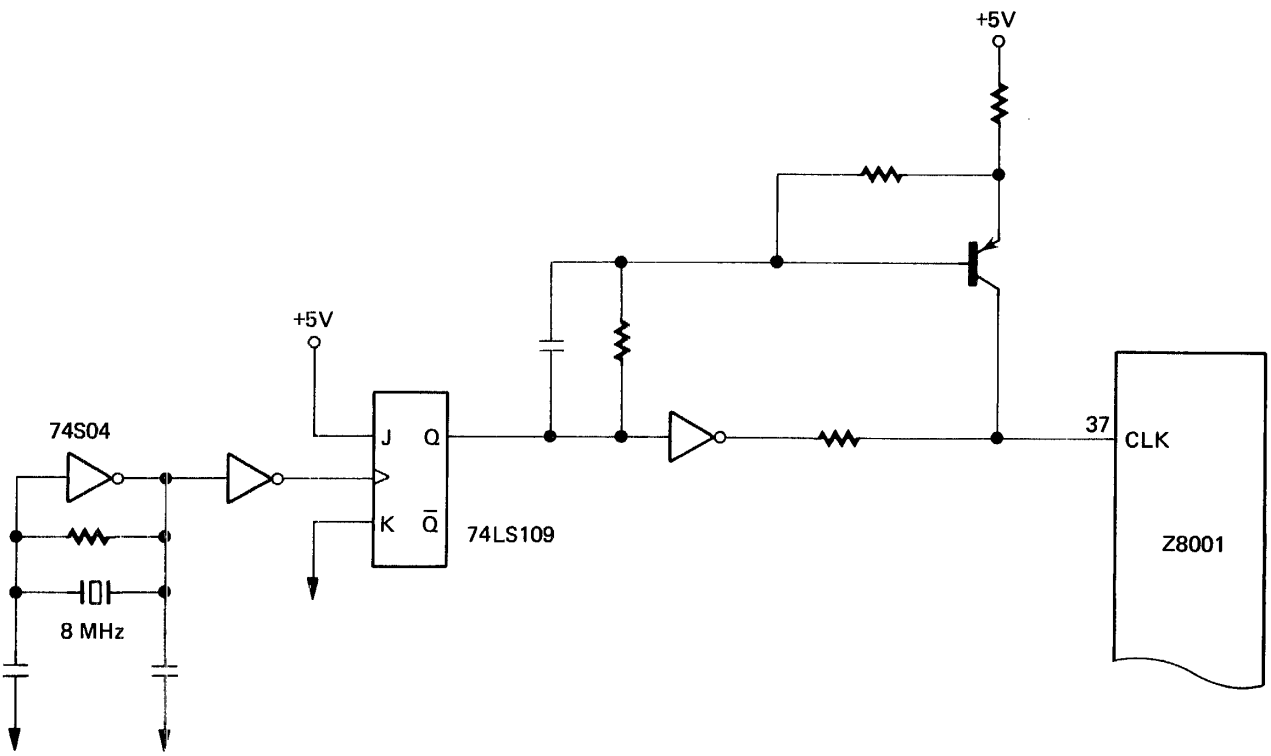


Figure 4-3

In Figure 4-3, the 8 MHz crystal is divided to 4 MHz by the 74LS109. The 4 MHz square wave is then shaped and enhanced by the transistor circuit to provide the proper signal to the Z8000. An additional tap is taken from the 74LS109 to provide a 4 MHz clock to the system by way of the common bus.

Now we have power and a clock. The third part of the required kernal is memory, usually in the form of PROM or ROM, that will allow the system to be powered up (booted) in a sequential fashion. For now, let's skip over the boot-up and look instead at connecting the Z8001 to it main memory, using a Z8010 Memory Management Unit. Connecting the Z8001 to boot-up memory is discussed later in this chapter.

MEMORY MANAGEMENT

Introduction

The Z8001 can, by itself, access 8 M-bytes of memory. With the addition of a Z8010 Memory Managment Unit (MMU), however, that 8 M-bytes of memory is more easily managed. The MMU provides logical-to-physical address translation, memory protection, and many other valuable features.

The Z8010 MMU

As stated before, the Z8010 MMU provides many features to a Z8001-based system:

- Dynamic segment relocation.
- Segment size is individually assignable from 256 bytes to 64 K-bytes
- The MMU can provide protection attributes for system security:

  Read Only
  Execute Only
  System Only
  CPU-Inhibit
  DMA-Inhibit

- The MMU can maintain segment history.

Let's look at each of these features in more detail.

Segment Relocation. In most situations, the user of a system doesn't know or even care where in memory his program is. If a single user is using a system, it doesn't actually matter, because you can assume that he's using all available memory. However, if more that one person is using a system, you need some method of partitioning memory. The Z8010 provides just such a feature. By programming the Z8010 MMU, you can set aside areas in memory for system use, user use, read only, execute only, and so on. The Z8010 can be reprogrammed to change the actual **physical** locations of all these areas whenever necessary. Dynamic segment relocation frees the user from actually specifying where in memory his information is located. The Z8010 takes care of that.

Segment Control. A single Z8010 MMU can control 64 of the Z8001's 128 segments. The MMU can set the size of the 64 segments, ranging from 256 bytes to 64 K-bytes in length. If an access to memory outside the segment boundary occurs, the MMU signals the Z8001. The segment boundary can then be changed if necessary, or the access denied. Either way, access to memory outside a prescribed range is not permitted.

To increase the number of segments under control to 128, all you have to do is add another Z8010 MMU to your system.

Protection Attributes. The Z8010 can assign five protection attributes to each segment. Those attributes are:

1. Read Only. If an attempt is made to write into a memory segment assigned this attribute, a segment trap request is made to the Z8001.

2. Execute Only. This attribute allows the segment to be read only during instruction fetch cycles.

3. System Only. The System Only attribute will let the segment be accessed only while the Z8001 is in system mode. If an access is attempted in normal mode, a segment trap request is made.

4. CPU-Inhibit. This attribute inhibits the processor from accessing the segment, while allowing DMA access only.

5. DMA-Inhibit. The DMA-Inhibit attribute allows only the processor to access the segment, while inhibiting DMA accesses.

Any or all of these segment attributes can be assigned to a particular segment (although you wouldn't want to set all attributes).

Segment History. There are two bits in the attribute field of each segment that indicate that the segment has been written

to or referenced. These bits allow the Z8001 to determine if any segment has been accessed and if that access was a read or write. In this way, a running history of segment activity can be kept.

More information on the detailed operation of the Z8010 MMU is provided later in Chapter 5. Next, we look at how the MMU interfaces to the Z8001.

Z8001-MMU Interface

The Z8010 is interfaced to the Z8001 in the following manner. The MMU is attached to the Z8001 status lines (ST0-ST3), segment lines (SN0-SN6), the $\overline{\text{SEGT}}$ control line, and AD8-AD15. Memory addresses are then generated by the Z8001 and the MMU together. Figure 4-4 illustrates the Z8001/MMU interface.

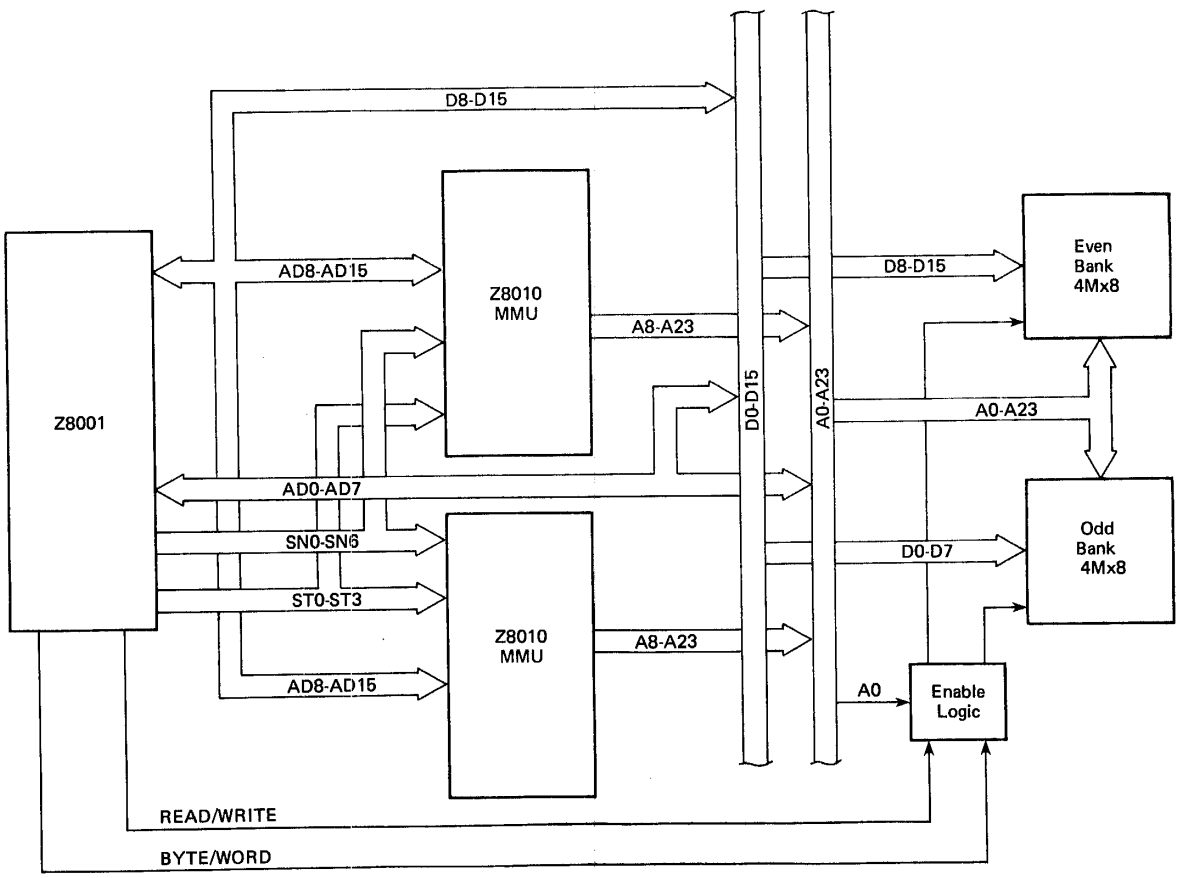


Figure 4-4

As you can see, the MMU provides address lines A8-A23, while the Z8001 provides A0-A7, giving the Z8001 a total of 24 address lines. In operation, the MMU uses the Z8001 segment lines to index an address translation table. The Z8001 segment lines are routed to a 64 x 16-bit base address memory table within the MMU. The base address table is loaded with segment information when the system is initialized. The incoming segment value is used as an index to look up a physical address. The offset value coming from the Z8001 (AD8-AD15) is then added to the indexed value to create a new physical location within memory.

In addition to the base address translation table, the MMU contains a protection table that's also programmed at system initialization. The protection table assigns attributes to each segment in memory. Whenever a segment is addressed by the Z8001, the MMU checks the attribute table against the Z8001 status lines and either allows or prevents memory access to take place.

MEMORY INTERFACE

Introduction

We have now covered the Z8010 MMU and are ready to add memory to our system. In this application, the Z8001 memory will be handled by two Z8010s, while the Z8002 memory will be interfaced directly to the processor.

Since the Z8010s will handle segment management for the Z8001, the system will have an address bus that's 24 bits wide. Therefore, system memory will be build around the 24-bit address bus and 16-bit data bus.

System memory for the Z8001 will include PROM devices to handle boot-up, and 8 M-bytes of RAM for program and data storage. Refer again to Figure 4-4.

RAM

A good choice for system memory might be 2167-type static 16K x 1 RAMs. In order to provide 8 M-bytes of RAM, a high-density, relatively high-speed device like the 2167 is required to keep the number of devices and circuit board "real estate" down to a manageable level. An additional benefit of using the 2167 is its low standby current drain. The power required by the 2167 during standby is less than half of that required while active. And since we can assume that not all our memory will be active at one time, the power savings is substantial.

In the block diagram in Figure 4-4, you will notice that

memory is arranged in an even-bank, odd-bank fashion. The even bank will contain all the even address bytes, while the odd bank will contain all the odd address bytes. This arrangement allows the Z8001 to access either bytes or words. Words are accessed at even addresses only, while bytes can be accessed at even or odd addresses. Selection of the proper bank (even, odd, or both) is made by using address line A0 and the BYTE/WORD control line. The READ/WRITE line, of course, allows memory to be read or written to.

The use of two MMU devices allows all 128 memory segments to be controlled. Each MMU contains a flag called URS (Upper Range Select) that allows each MMU to be assigned a segment range, either 0 through 64 or 65 through 128. When a particular MMU is not being accessed, it latches all its output lines to the high-impedance state.

ROM

The next requirement for our system is boot-up ROM (or PROM). Recall that upon reset, the Z8001 goes to segment 0, locations 0002, 0004, and 0006. So we need ROM at the bottom of memory. The ROM interface is the same as RAM interface, so it could reside at the bottom of memory, replacing a small portion of RAM.

Memory Useage

In the imaginary system, the operating system will be called from disc and loaded into the Z8001 RAM space. With this requirement in mind, the best way to perform this action would be to have the Z8001 operate as the system's "master" processor. When power-up or master restart occurs, the Z8001 should be activated while the two Z8002s remain inactive. The Z8001 would then begin execution in ROM by telling the mass storage Z8002 to load the operating system from disc and DMA it to Z8001 RAM. Once this task is complete, the mass storage Z8002 will again be set inactive until needed. As an alternative, the mass storage Z8002 could be the first processor on-line, immediately loading the Z8001 memory, then turning the bus over to the Z8001. Either way, an orderly power-up or master reset sequence must take place. DMA data transfer is discussed later in this chapter.

Z8002 Memory

Since the Z8002 devices can only access 64 K-bytes of memory, their interface requirements are much simpler. The Z8002s require a straightforward connection between processor and memory, with the only special need being that memory be off

the bus whenever the processor is off the bus, except during DMA transactions.

Memory Organization

Memory can be partitioned by the Z8000 in several ways:

- o system code
- o normal code
- o system data
- o normal data
- o system stack
- o normal stack

Memory is divided into these different areas through the use of the Z8000 status lines (ST0-ST3) and the system/normal (S/$\overline{N}$) line. These control lines can be used as memory enable qualifiers. For example, if you wanted to reserve an area of memory for system stack, you can simply use the S/$\overline{N}$ line and status lines to decode a system memory transaction to the stack. Any other transaction would not enable that particular portion of memory.

Memory Timing

The Z8000 timing requirements for memory are covered in Chapter 3. Refer to that section for actual timing requirements.

DMA Transactions

Since we have a system with three processors and three different memories, this is a good time to talk about DMA transfers and how they're handled by the Z8001 and its Z8010s, and the Z8002s. The most practical way to handle DMA transfers would be to implement a Z8016 DMA controller . The controller will interface to the processor through two control lines: $\overline{BUSRQ}$ and $\overline{BUSAK}$. To gain control of the bus, the controller will drive the Z8000 $\overline{BUSRQ}$ line low. When the Z8000 is ready to relinquish the bus, it drives the $\overline{BUSAK}$ line low.

In the case of our three-processor system, three DMA controllers could be used in a daisy chain configuration. The Z8001 would have the highest priority controller, while the Z8002s would arbitrarily be set to second and third priority.

DMA operations for the Z8002 devices are relatively simple, in that the Z8002 connects to the DMA controller and allows it to process DMA activities. However, the Z8001/Z8010 combination has some special requirements and features that

must be addressed.

Z8001/Z8010 DMA. The Z8001 device will allow DMA transfers between instruction cycles. When the Z8001 is attached to a Z8010 MMU, the MMU handles DMA.

The MMU permits DMA transactions in either system or normal mode. During a transaction, the MMU checks to see if the transfer is in violation of segment attributes. If a violation occurs, the MMU asserts the $\overline{SUP}$ (suppress) line. The device performing the DMA transfer must monitor the $\overline{SUP}$ line, because the MMU does **not** notify the Z8001 of a violation during DMA transfers.

The Z8010 MMU has a control line called $\overline{DMASYNC}$, that's used to synchronize DMA transfers. At the start of a DMA cycle, the requesting device must send the $\overline{DMASYNC}$ line low. Then when the memory address is valid, the requesting device must have the $\overline{DMASYNC}$ line high on the rising edge of the clock. At this time, the MMU checks the address to see if access is permitted. The $\overline{SUP}$ signal will be generated if an access violation occurs.

If the $\overline{DMASYNC}$ line is high for two clock cycles, the MMU assumes that the DMA transfer is over and that the Z8001 again has control of the bus. Therefore, for each memory transfer, the MMU checks the address and either allows or disallows the request.

An Alternative to DMA

In our system, there is an alternative to DMA operations. The Z8000 supports a multi-microprocessor system with multi-micro control lines $\overline{\mu O}$ and $\overline{\mu I}$. The multi-micro capabilities allow the Z8000 devices to share resources such as memory or I/O. It's a relatively simple task to set up a prioritization scheme where three processors might request the same resource. The processors can daisy-chain their requests.

In our system, four control lines must be build into the system bus: $\overline{MMST}$, $\overline{MMRQ}$, $\overline{MMAI}$, and $\overline{MMAO}$. The lines are defined as follows:

$\overline{MMST}$. The $\overline{MMST}$ line signifies that the resource is busy. A requesting processor will look at the state of the $\overline{MMST}$ line before requesting a resource.

$\overline{MMRQ}$. The $\overline{MMRQ}$ line is used as the resource request line. When a processor requires access to a resource, it forces the $\overline{MMRQ}$ line low.

$\overline{MMAI}$. Following a resource request ($\overline{MMRQ}$), the processor asserts the $\overline{MMAI}$ signal. The $\overline{MMAI}$ signal is an exceptance request passed through the daisy chain. If a higher-priority processor has a need for the resource, it blocks the $\overline{MMAI}$. If

the resource is not required, the higher-priority processor passes the request by asserting its $\overline{\text{MMAO}}$ line.

$\overline{\text{MMAO}}$. This line is asserted when a processor in the daisy chain passes a resource request.

The four control lines can be connected to the multiple processors in our system as shown in Figure 4-5.

When a processor requires the use of a shared resource, it inspects the $\overline{\text{MMST}}$ line. If $\overline{\text{MMST}}$ is low, the resource is busy. If $\overline{\text{MMST}}$ is high, the resource is available.

The Z8000 generates a request by forcing the $\overline{\text{MMRQ}}$ line low, thus driving the $\overline{\text{MMST}}$ lines of the other processors low, which indicates that a resource is in use. The low on $\overline{\text{MMRQ}}$ also is fed to the $\overline{\text{MMRQ}}$ line of the highest-priority processor. This allows the highest-priority processor to use the resource, if necessary. If the highest-priority processor doesn't need the resource, it drives its $\overline{\text{MMAO}}$ line low, passing the request on to the next-highest priority processor.

The request is passed on down the chain until it reaches the processor that made the initial request. When the request reaches the initial processor, it holds its $\overline{\text{MMAO}}$ line high, preventing any other requests from being made until it's done.

The Z8000 uses the MREQ, MBIT, MRES, and MSET instructions to handle multi-processor environments. Refer to Chapter 2 for information concerning these instructions.

I/O OPERATIONS

Our imaginary graphics system is connected to four graphics terminals. We can assume that the easiest way to interface the system to the terminals would be through serial I/O ports. A good choice for handling serial I/O would be the Z8030 dual-port Serial Communications Controller. Two of these controllers could handle the system terminals. A third controller could be used by the mass storage Z8002 as a disc drive controller. In any case, the Z8000 devices support a variety of I/O capabilities, reflected in the Z8000 instruction set. Refer to Chapter 2 for specific Z8000 I/O instructions. I/O timing is described in Chapter 3 of this book.

However, for the purposes of this chapter, assume that you would like to interface the Z8000 to 8-bit interface devices such as the Z80 Programmable I/O (PIO).

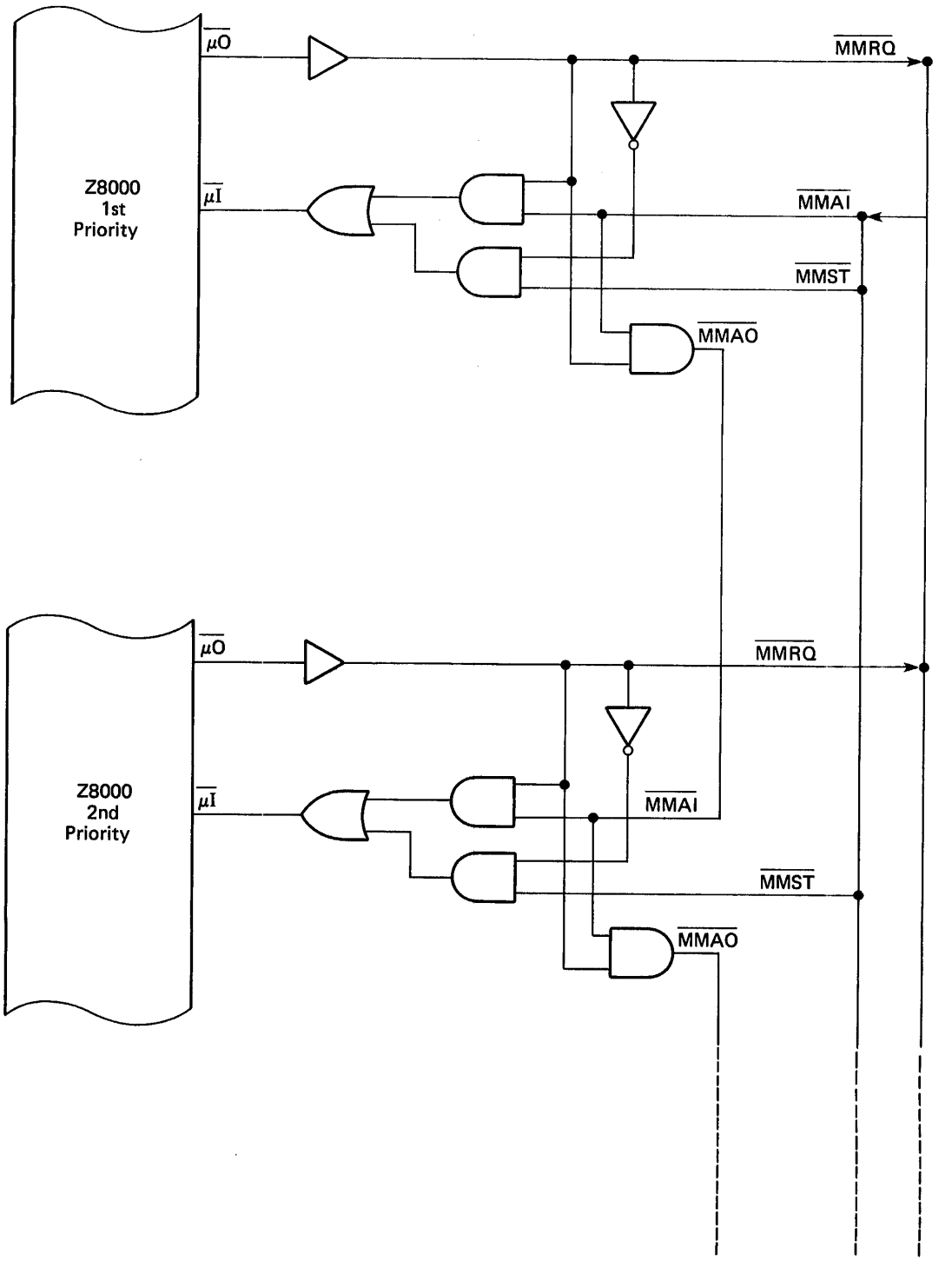

Z8000
1st
Priority
μO
μI
MMRQ
MMAI
MMST
MMAO
Z8000
2nd
Priority
μO
μI
MMRQ
MMAI
MMST
MMAO

Figure 4-5

Z8000-Z80 PIO Interface

Recall that the Z8000 is able to read and write eight or sixteen bits of I/O data at a time, and that the Z8000 has a wide variety of I/O instructions that allow it to perform single byte/word data transfers, or whole block data transfers.

You can use a single Z80 PIO as an 8-bit dual-port I/O interface, or two PIOs can be used as a 16-bit dual-port I/O interface. Figure 4-6 is a block diagram of the Z8001 used with two PIOs to provide two 16-bit interfaces.

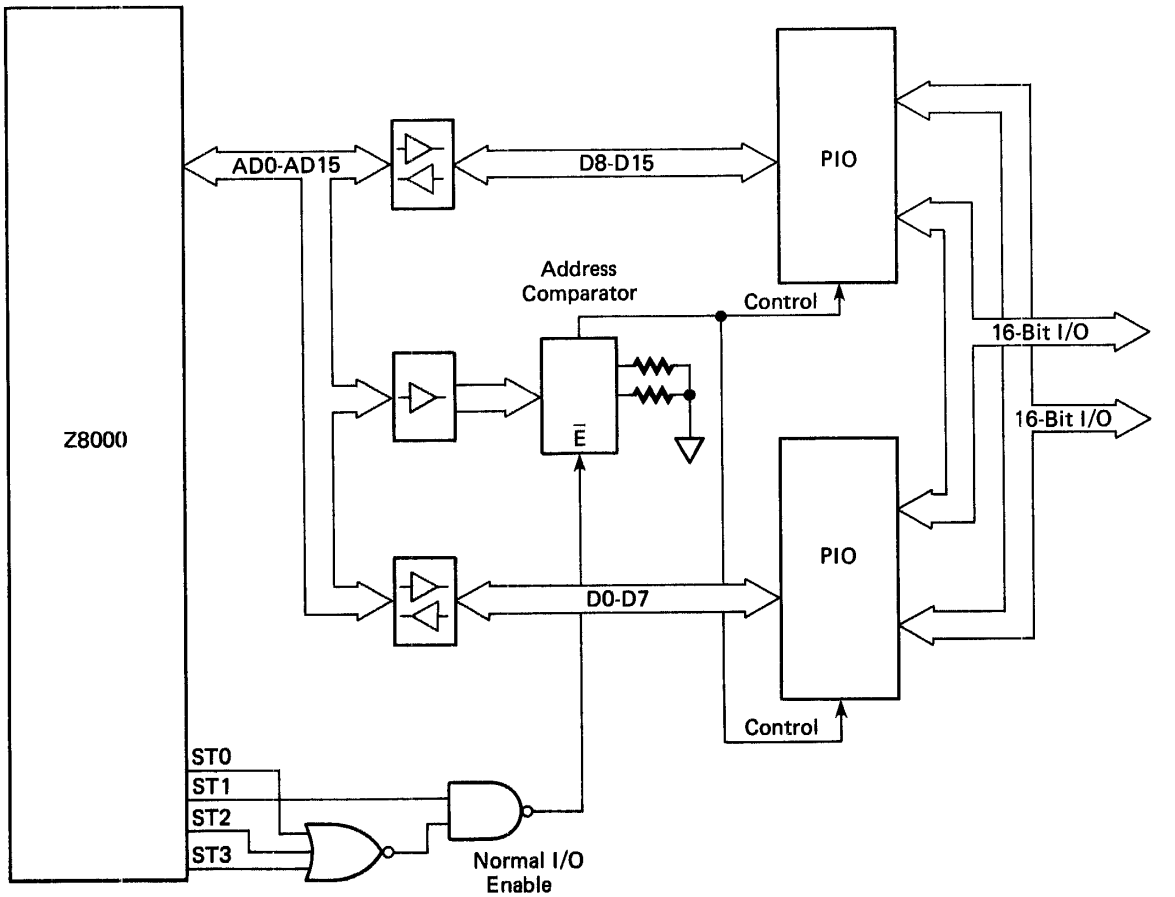


Figure 4-6

The Z8000 status lines (ST0-ST3) are fed through a simple NOR-NAND arrangement to provide a normal I/O enable line. Directional control for the line transceivers would be provided with logic tied to the Z8000's $\overline{DS}$ (data strobe) and R/$\overline{W}$ control lines. The Z8000 $\overline{AS}$ (address strobe) line would latch the address to the address comparator.

Although this is an overly simplified block diagram (the Z80 PIO has six control lines that must be accessed), it

provides an insight into the capability of the Z8000 to interface to 8-bit devices. The Z80 PIO is described fully in the next chapter.

SUMMARY

A more complete block diagram of the graphics system used in this chapter is shown in Figure 4-7. It's beyond the scope of this book to detail the operating system requirements for such a piece of equipment. However, as you can see, the Z8000 devices are flexible enough to support almost any kind of operation you might require.

In the next chapter, most of the devices mentioned in this chapter are described more fully. Included are the details of interfacing these devices to the Z8000.

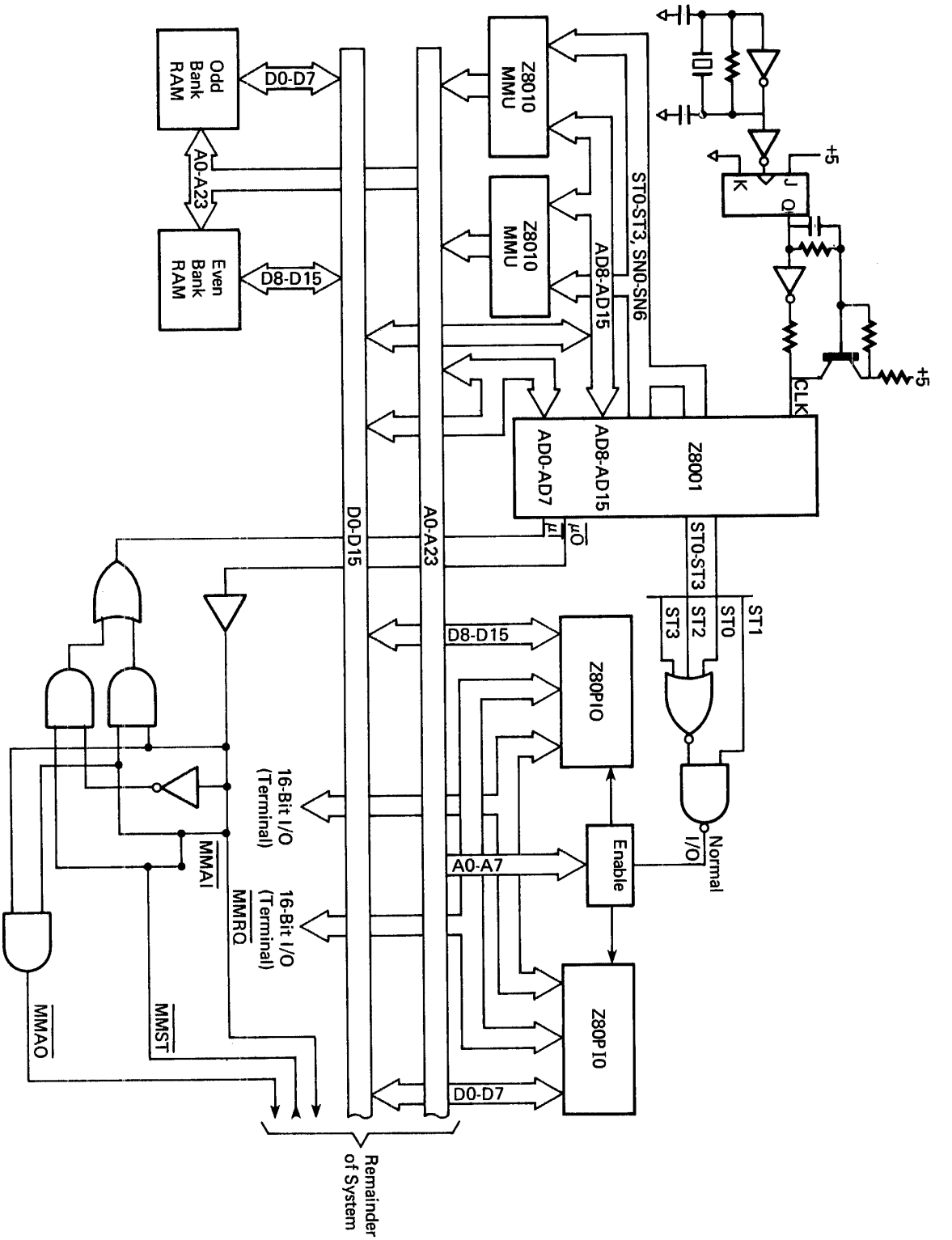

Odd Bank RAM
D0-D7
A0-A23
Even Bank RAM
D8-D15
Z8010 MMU
Z8010 MMU
ST0-ST3, SN0-SN6
AD8-AD15
+5
J
K
Q
CLK
AD0-AD7
AD8-AD15
Z8001
D0-D15
A0-A23
μI
μO
ST0-ST3
ST1
ST0
ST2
ST3
D8-D15
Z80PIO
16-Bit I/O (Terminal)
16-Bit I/O (Terminal)
MMAI
MMRQ
A0-A7
Enable
I/O
Normal
MMAO
MMST
Z80PIO
D0-D7
Remainder of System

Figure 4-7

5

Z8000 SUPPORT DEVICES

INTRODUCTION

In this chapter, we'll look at some of the support devices available for the Z8000, and how they interface to the microprocessor. The following devices will be described:

- Z8010 Memory Management Unit (MMU)
- Z8030 Serial Communications Controller (SCC)
- Z8036 Counter/Timer and Parallel I/O Unit (CIO)
- Intel 2167 16Kx1 Static RAM
- Z80 Parallel I/O Device

Much of the information about new Z8000 support devices is preliminary in nature at this writing. However, as much useful detail is included as possible.

In reading the information concerning the Z80 PIO, keep in mind that the PIO was designed long before the advent of the Z8000. However, Zilog managed to provide enough Z8000 to Z80 interface points to allow easy interface to Z80 peripherals. There are a few points to be made about

Z8000-to-Z80 interface. One is control lines.

The Z80 generates separate read ($\overline{RD}$) and write ($\overline{WR}$) control lines, while the Z8000 uses one read/write line (R/$\overline{W}$). A simple interface may be constructed to modify Z8000 output for Z80 peripheral devices. Figure 5-1 illustrates just such an interface.

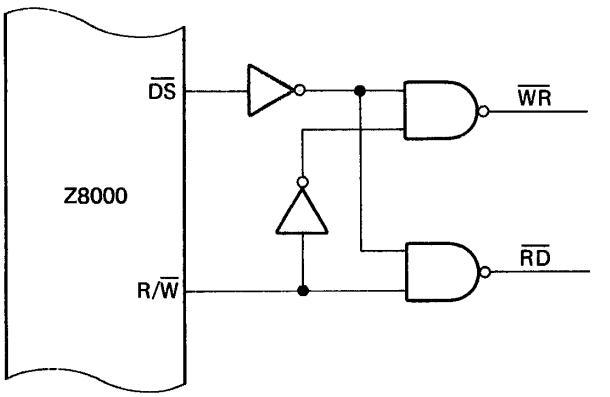


Figure 5-1 Z80 $\overline{WR}$-$\overline{RD}$ generation.

Most of the remainder of Z80 peripheral control line requirements can be constructed from the Z8000 status lines (ST0-ST3). Every major operation executed by the Z8000 is reflected by the state of the control lines.

Of course, peripheral devices designed for the Z8000 don't require any special interface considerations.

Z8010 MEMORY MANAGEMENT UNIT

Introduction

The Z8010 Memory Management Unit (MMU) provides two primary functions for the Z8001 microprocessor: it allows transparent relocation and sizing of memory segments, and it assigns protection attributes to those segments.

The Z8010 translates logical segment addresses provided by the Z8001 into physical memory addresses. Each segment can be sized from 256 bytes to 64 K-bytes, and each segment is assigned attributes that are checked each time that segment of memory is accessed.

The Z8010 also supports a multi-MMU environment, in that up to eight MMUs can be interfaced to a single Z8001. This feature allows the Z8001 to control what could be termed "virtual" memory. The control and status reporting capabilities of the MMU are abundant enough to provide complete control over massive amounts of memory, beyond the Z8001's normal 8 M-bytes. Multiple MMUs can also provide even greater control over the 8 M-bytes possible, by assigning separate translation tables to each task.

The following text describes the internal architecture of the Z8010, how the Z8010 is programmed, and how the Z8010 interfaces to the Z8001.

Z8010 Architecture

There are three tables maintained within the Z8010: a segment base address table, a segment size table, and a segment attribute table. The tables are shown in Figure 5-2.

Size Limit	Access Restrictions	Base Address
8 Bits	8 Bits	16 Bits
	•	
	•	
	•	

Figure 5-2

Segment Base Address Table. Recall from Chapter 4 that the Z8001 provides a segment number and AD8-AD15 as inputs to the Z8010. The 7-bit segment number is fed into a segment base address table within the Z8010. The table contains values that are programmed into the Z8010 at initialization. The incoming segment value acts as an index pointer to one of the 64 segment base values.

The value pointed to by the Z8001 segment number is then added to another value created from AD8-AD15. The output of the 16-bit adder is then used to generate bits 8 through 23 of

a physical address. Bits 0 through 7 are provided directly by the Z8001.

Figure 5-3 illustrates the flow of addresses through the Z8010.

Segment Limit Table. Another table programmed at initialization is the segment limit table. This table is used to assign sizes to each segment, ranging from 256 bytes to 64 K-bytes in 256 byte blocks. The segment limit table is 8 bits wide, and each segment is assigned a limit. Then each time a segment is accessed, the segment limit is checked to see if the address is out of bounds. If segment length is violated, the Z8010 sets a bit in a status register.

Segment Attributes Table. Each segment can be assigned attributes that indicate how that segment may be used. If an attempt is made to violate an attribute, the Z8010 issues a trap request to the Z8001 and asserts the SUP (suppress) control line. The SUP line is monitored by memory logic and prevents any access to the address currently on the address bus. The following segment attributes are available:

Read-Only. A segment assigned the read-only attribute is write protected. Any time an attempt is made to write to a read-only segment, the Z8010 issues a trap request to the Z8001 and asserts the SUP (suppress) control line.

Execute-Only. This attribute is used to protect executable code. The only time a segment with the execute-only attribute may be accessed is during an instruction fetch. If an attempt is made to access a segment with this attribute and the Z8001 is **not** in a fetch cycle, a trap request is issued, and the SUP line is asserted.

System-Only. The system-only attribute allows a segment to be accessed only if the Z8001 is operating in the system mode.

CPU Inhibit. The CPU inhibit attribute allows only DMA accesses to the segment. The segment with this attribute cannot be accessed by the Z8001.

DMA Inhibit. This attribute prevents DMA accesses to the segment. Only the Z8001 can access a segment with the DMA inhibit attribute.

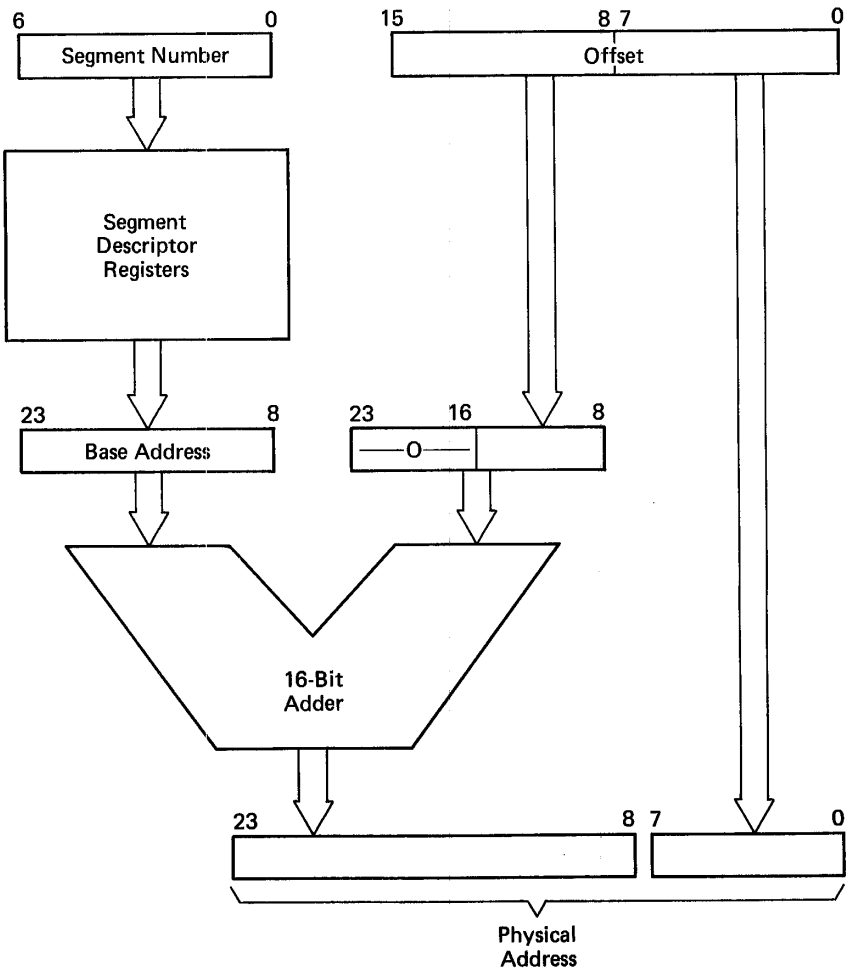


Figure 5-3 Address flow through the Z8010.

Programming the Z8010

The Z8010 is controlled by Z8001 "special I/O" instructions. There are 22 special I/O instructions that can be executed by the Z8001 while in system mode. The MMU monitors the Z8001 status lines to detect when a special I/O instruction is taking place. During a special I/O machine cycle, the MMU goes into command mode and accepts the command over AD8-AD15. If data is to be written or read, the transfer is made during the next part of the special I/O cycle.

The Z8010 MMU is programmed by way of three register sets: Control, Segment Descriptor, and Status. Each of these registers is described next.

Control Registers. The Z8010 control registers provide a means to control MMU operation. There are three 8-bit registers: the mode register, the segment address register (SAR), and the descriptor register.

Mode Register. The Mode register is used in a multi-MMU environment to selectively enable a single MMU. The Mode register contains six fields, each of which control MMU operation. The six fields are illustrated in Figure 5-4.

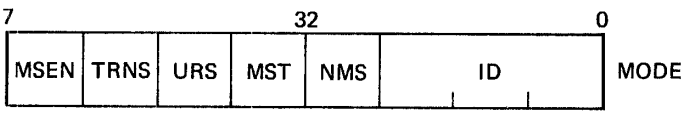


Figure 5-4

ID The ID field is used to assign the MMU a 3-bit identification number. The ID field is used by the Z8001 during a segment trap acknowledge sequence to identify which MMU generated the trap.

NMS (Normal Mode Select). The NMS bit is used to indicate when the MMU will translate addresses. If the NMS bit is set high, the MMU will translate only when the Z8001 is in normal mode. If the NMS bit is reset low, the MMU will translate only when the Z8001 is in system mode. When the NMS bit and the state of the N/$\overline{S}$ line don't agree, the MMU keeps its output lines in a high-impendance state. The action of the NMS bit is determined by the state of the MST bit in this register, which is described next.

MST (Multiple Segment Table). The MST bit is set to indicate that there is more than one MMU being driven by the Z8001. When this bit is set, The N/$\overline{S}$ line is used to determine whether the MMU contains the right segment table. See the NMS bit.

URS (Upper Range Select). The URS bit determines whether the MMU contains the descriptors for segments 0 - 64 or for segments 65 - 128. The most significant bit of the segment number (SN6) must match the state of the URS bit for the MMU to operate. Otherwise, the MMU outputs are kept in their high-impedance state.

TRNS (Translate). This bit allows the MMU to be used in a "transparent" mode. That is, if the TRNS bit is reset, the addresses presented by the Z8001 are routed directly to the outputs of the MMU, without translation or segment checking. When in the transparent mode, the most significant byte of the output is the segment number, and the most significant bit of the segment number is 0. If the TRNS bit is set, the MMU operates normally.

MSEN (Master Enable). The MSEN bit is the master enable bit for the MMU. Simply put, if MSEN is set, the MMU works. If MSEN is reset, the MMU is disabled, and its outputs remain in the high-impedance state.

Segment Address Register (SAR). The Segment Address Register (SAR) is a 6-bit read/write register that points to one of the 64 segment descriptors. The bit pattern of the SAR is shown in Figure 5-5.

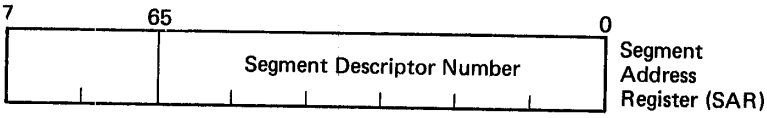


Figure 5-5

The SAR is used to select a particular segment descriptor register for reading or writing. Control commands used to access segment descriptor registers implicitly use the SAR to point to a particular register.

An additional capability of the SAR is its auto-incrementing function. You can read or write single segment descriptors one at a time, or you can access the descriptors in block fashion, using the Z8001's block I/O transfer capabilities. The commands used to access segment descriptor registers are given later in this chapter.

Descriptor Selection Register (DSR). The Descriptor Selection Register is a 2-bit register used to select on of the three segment descriptor fields. The bit pattern for the DSR is shown in Figure 5-6. Table 5-1 lists the binary values and their corresponding descriptor fields.

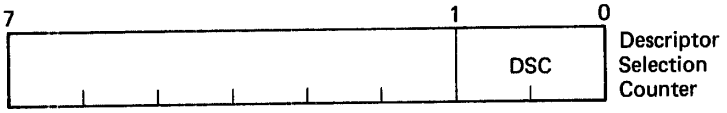


Figure 5-6

Table 5-1
DSR Bit Definitions

Bit 1	Bit 0	Descriptor
0	0	High-order Byte of Base Address
0	1	Low-order Byte of Base Address
1	0	Segment Limit Field
1	1	Attribute Field

Control Register Commands. The control registers are accessed by three MMU commands, as shown in Table 5-2.

Table 5-2
Control Register Instructions

Opcode	Instruction
00	Read/Write the Segment Mode Register
01	Read/Write the Segment Address Register (SAR)
20	Read/Write the Descriptor Selection Register

Segment Descriptor Registers

There are 64 segment descriptor registers in the Z8010. Each register contains a 16-bit base address field, an 8-bit limit (size) field, and an 8-bit attribute field, as shown in Figure 5-7.

Figure 5-7 Atttribute fields.

The base address field is divided into a high-order and low-order byte. The Descriptor register is loaded a single byte at a time when programmed, with the high-order byte loaded first.

The limit field contains a value describing the size of that particular segment. The value sizes the segment in terms of 256-byte blocks.

The attribute field determines the attributes assigned a segment. There are eight attribute flags that must be programmed:

Read-Only (bit 0). The segment is write-protected.

System Only (bit 1). The segment can only be accessed while the Z8001 is in system mode.

CPU-Inhibit (bit 2). The segment can be accessed only during a DMA transfer.

Execute-Only (bit 3). The segment can be accessed only during an instruction fetch.

DMA-Inhibit (bit 4). The segment is protected against DMA access.

Direction and Warning (bit 5). This bit is used to warn of potential segment overflow. When this flag is set, the Z8010 assumes that memory is accessed in decending order. During each access, the Z8010 checks to see if the lowest 256-byte block is being acceessed. If the last block is being accessed, the Z8010 generates a segment trap to notify the Z8001.

The Direction and Warning bit is particularly useful in stack management, in that the system can be warned before stack space is used up.

Changed (bit 6). The Changed flag is set whenever the segment is written to. This bit can be used to maintain segment history.

Referenced (bit 7). This flag is set whenever the segment is accessed, either through a read or write. Like the Changed

flag, the Referenced flag can be used to maintain segment history.

Programming the Segment Descriptor Registers

There are 10 commands available to program the Segment Descriptor registers, as listed in Table 5-3.

Table 5-3
Segment Descriptor Instructions

OPCODE	INSTRUCTION
08	Read/Write the Base Address Field
09	Read/Write the Limit Field
0A	Read/Write the Attribute Field
0B	Read/Write All Descriptor Fields
0C	Read/Write the Base Field, Increment SAR
0D	Read/Write the Limit Field, Increment SAR
0E	Read/Write the Attribute Field, Increment SAR
0F	Read/Write All Descriptor Fields, Increment SAR
15	Set All CPU-Inhibit Flags
16	Set All DMA-Inhibit Flags

The Z8001's block I/O transfer instructions can be used to load all 64 segment descriptor registers. The Segment Address Register (SAR) within the Z8010 automatically increments to the next field each time the registers are accessed. See the description of the SAR given previously.

As you can see in Table 5-3, you have the option of programming an individual descriptor field or all descriptor fields. You also have the option of programming the fields in block fashion, using the SAR.

Status Registers

The Z8010 MMU utilizes six 8-bit registers to report its status. Each register is described in the following.

Violation Type Register. The Violation Type Register describes the type of error that caused a segmentation trap to be generated. This register, along with the Violation Segment Number and Violation Offset registers, provides complete information as to the cause of a trap. Figure 5-8 illustrates the contents of the Violation Type register.

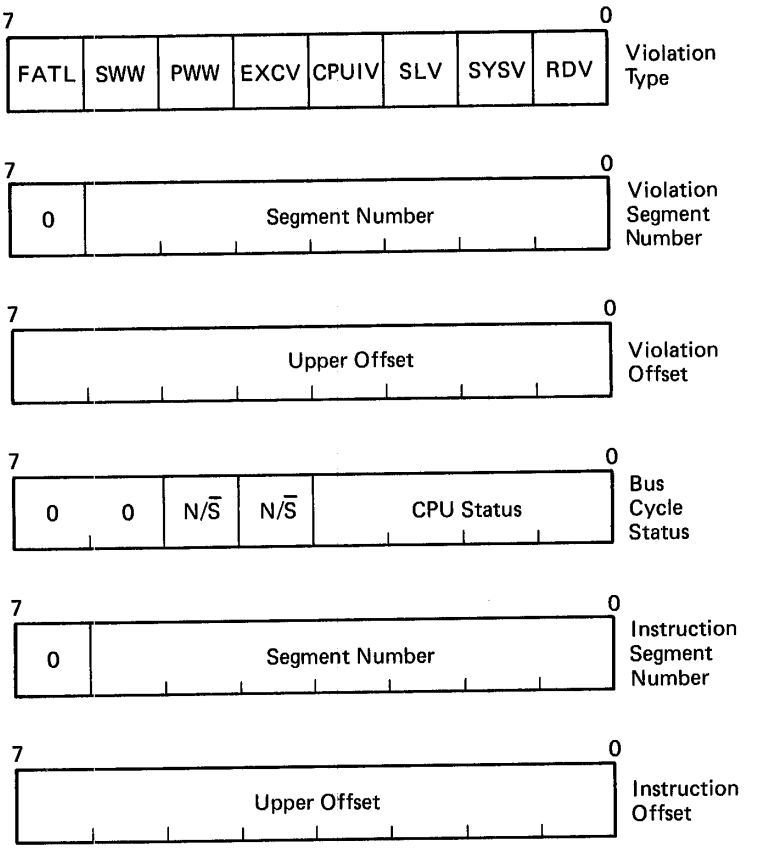


Figure 5-8

Read-Only Violation (RDV). This bit is set when the Z8001 attempts to write to a read-only protected segment.

System Violation (SYSV). The SYSV bit is set when the Z8001 attempts to access a segment in normal mode, and the segment has the system only attribute.

CPU-Inhibit Violation (CPUIV). This bit indicates that the CPU attempted to access a segment reserved for DMA operations.

Execute-Only Violation (EXCV). This bit is set when the Z8001 attempts to access a segment in something other than an instruction fetch cycle.

Segment Length Violation (SLV). The SLV bit is set when an offset falls outside of the memory range specified by the limit field.

Primary Write Warning (PWW). If the DIRW flag is set, the PWW bit is set when an access is made to the lowest 256-byte block of memory. This bit is useful in actively sizing stack memory.

Secondary Write Warning (SWW). The SWW bit is used when the Z8001 pushes data into the last 256 bytes of the system stack. This bit is set only if the EXCV, CPUIV, SLV, SYSV, RDV, or PWW bits are set. Once the SWW bit is set, subsequent write warnings do not generate a Segment Trap.

Fatal Condition (FATL). This bit is set if a memory access error has occurred in the trap processing routine. This bit is also set if any other bit in the register is set, and a violation is detected or a write warning occurs in normal mode. Once set, no segment traps are issued; however, memory suppress (SUP) signals are generated until the FATL bit has been reset.

Violation Segment Number Register. When a violation occurs, this register contains the number of the segment that caused the violation.

Violation Offset Register. The Violation Offset register contains the offset value of the segment that caused the violation.

Bus Cycle Status Register. The Bus Cycle Status register records the bus cycle status at the time of the address violation.

Instruction Segment Number Register. This register contains the segment number of the last valid instruction fetched before the violation occurred.

Instruction Offset Register. This register contains the high-order offset value of the last valid instruction fetched before the violation occurred.

Reading the Status Registers

The MMU status registers are read with the instructions listed in Table 5-4.

Table 5-4
Status Register Commands

Opcode	Instruction
02	Read Violation Type Register
03	Read Violation Segment Number Register
04	Read Violation Offset Register
05	Read Bus Status Register
06	Read Instruction Segment Number Register
07	Read Instruction Offset Register
11	Reset Violation Type Register
13	Reset SWW Bit in Violation Type Register
14	Reset FATL Bit in Violation Type Register

Z8010 Pinout and Timing

The remainder of this discussion of the Z8010 MMU involves the device pinout and timing relationships between the Z8001 and MMU and between the MMU and memory. First, the MMU pinout will be described.

Z8010 Pinout. The Z8010 MMU is a 48 pin device that connects between the Z8001 and memory. Figure 5-9 illustrates the MMU pins, and the text following describes each signal.

Inputs.

AD8-AD15 These are the address/data inputs from the Z8001. AD8-AD15 are used to transfer commands, data, and logical addresses between the Z8001 and MMU.

SN0-SN6 The segment lines are fed from the Z8001 to the MMU. SN0-SN5 are used to address one of the 64 segment tables in the MMU, while SN6 is used to enable a particular MMU in a dual-MMU environment.

ST0-ST3 The Z8001 status lines provide status information to the MMU. In particular, the status lines indicate when the Z8001 is issuing a special I/O command.

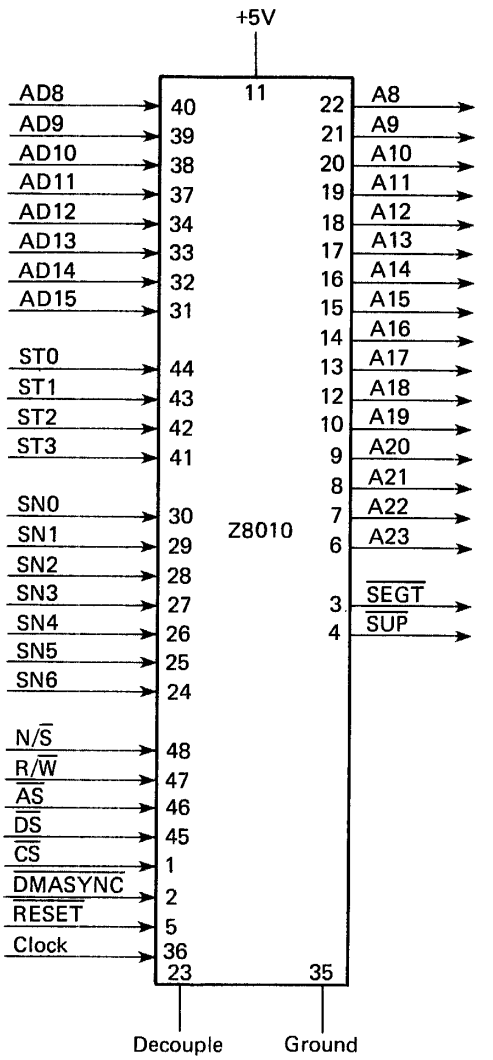


Figure 5-9

$\overline{AS}$ The address strobe ($\overline{AS}$) is used by the Z8001 to indicate a stable address is being presented.

$\overline{DS}$ The data strobe ($\overline{DS}$) line is used by the Z8001 to indicate that a data presented is stable.

N/$\overline{S}$ The normal/system mode line from the Z8001 is used to indicate whether the device is in normal or system mode. The MMU uses this line to check segment attributes and to switch between MMUs in a multi-MMU environment.

R/$\overline{W}$ The read/write line is issued from the Z8001 to

indicate whether it's reading or writing. The MMU checks this line for segment attribute violation.

CLK — The clock input is used by both the Z8001 and the MMU.

$\overline{CS}$ — This is the chip select input to the MMU. The $\overline{CS}$ line is useful in multi-MMU environments and indicates that the Z8001 is issuing a command. The $\overline{CS}$ line is not used during normal address translation.

$\overline{DMASYNC}$ — The $\overline{DMASYNC}$ line, when low, indicates that a DMA access is occuring. A high on this line indicates that the segment number issued by the Z8001 is valid. The $\overline{DMASYNC}$ line will always be high during Z8001 transactions.

$\overline{RESET}$ — A low on this line resets the MMU.

Outputs.

A8-A23 — The physical addresses generated by the MMU are output on these pins.

$\overline{SEGT}$ — This is the segment trap request issued to the Z8001. The MMU will assert $\overline{SEGT}$ whenever a segment violation or a write warning occurs.

$\overline{SUP}$ — The suppress line is asserted when any access violation occurs, except write warnings. $\overline{SUP}$ is usually used to prevent access to memory.

DECOUPLE — This line is an output from the MMU negative substrate-bias generator and serves no purpose as an active output.

MMU Timing

The Z8010 MMU communicates with the Z8001 through special I/O commands and the segment trap. Therefore, MMU timing is presented here in the following order: MMU command cycle, Segment Trap timing, memory read/write timing, and reset.

MMU Command Cycle. Figure 5-10 illustrates the command cycle timing from the Z8001 viewpoint. During a command cycle, commands are placed on AD8-AD15 by the Z8001 during T1. At the same time, Z8001 status information is available on ST0-ST3. The $\overline{CS}$ (chip select) line is asserted about midway through the T1 cycle.

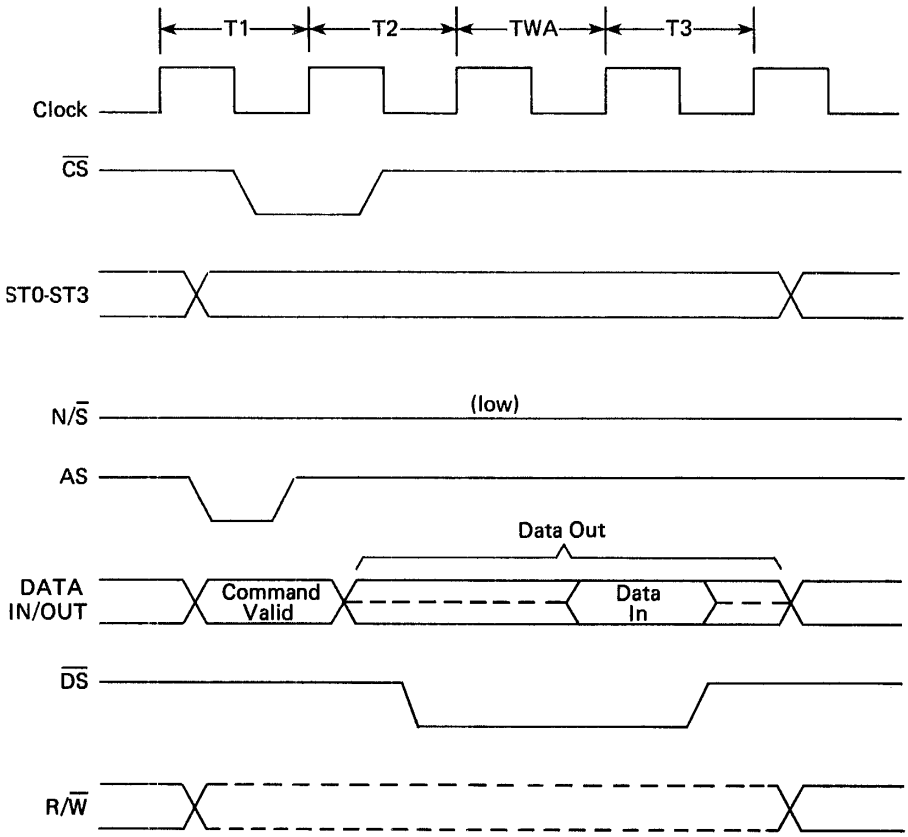


Figure 5-10

Data sent to the MMU following a command must be valid from the last part of the T2 cycle and into the first half of the T3 cycle.

Data read from the MMU by the Z8001 is valid from the end of TWA through the first half of T3.

MMU Segment Trap Timing. The MMU $\overline{\text{SEGT}}$ line is driven low whenever the MMU detects a segment violation or a write into the lowest block of a segment that has the DIRW bit set. Trap timing is illustrated in Figure 5-11. The $\overline{\text{SEGT}}$ line remains low until a trap acknowledge occurs.

During the acknowledge cycle, the MMU identifies itself by driving one of the address/data lines high. The particular line used is dependent upon the identifier in the Mode register.

If violations occur while the MMU is waiting for acknowledgement, the MMU continues to detect them and handle them appropriately.

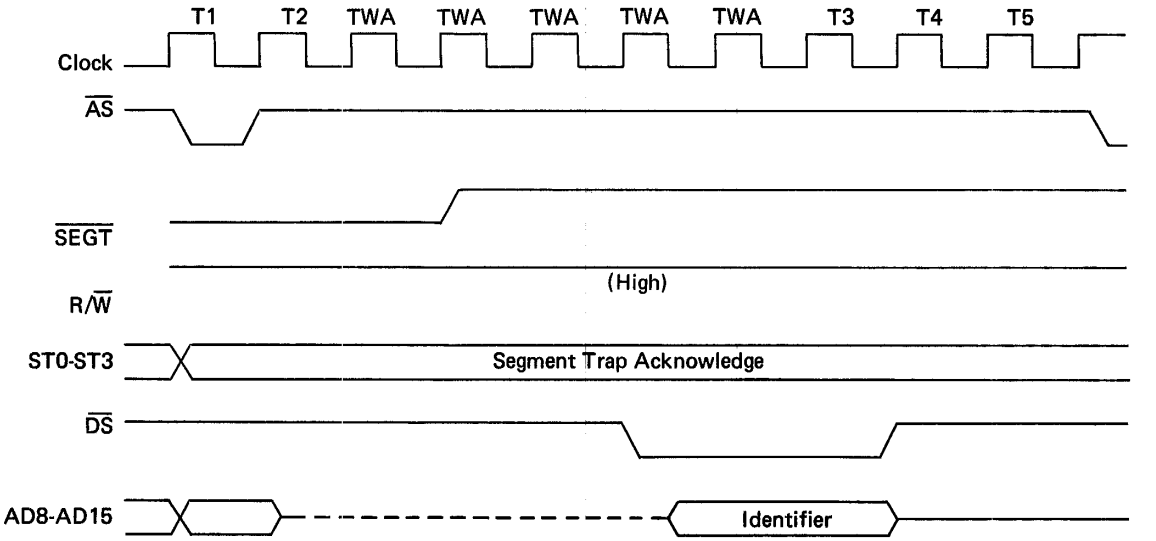


Figure 5-11

Memory Read/Write Timing. Memory timing is illustrated in Figure 5-12. During a memory read, the logical memory address is issued from the Z8001 early in T1. The output to memory on address lines A8-A23 is valid from the beginning of T2 through T3. The Z8001 will read data during T3.

During a memory write, the memory address is presented by the Z8001 early in T1, while the physical address (A8-A23) is issued by the MMU at the beginning of T2. Shortly after the physical address is valid, the Z8001 will write the data out.

If a segment violation occurs, the MMU asserts the $\overline{\text{SEGT}}$ line in T2. $\overline{\text{SEGT}}$ remains low until the trap acknowledge cycle begins. The $\overline{\text{SUP}}$ line is also asserted and terminates during T3.

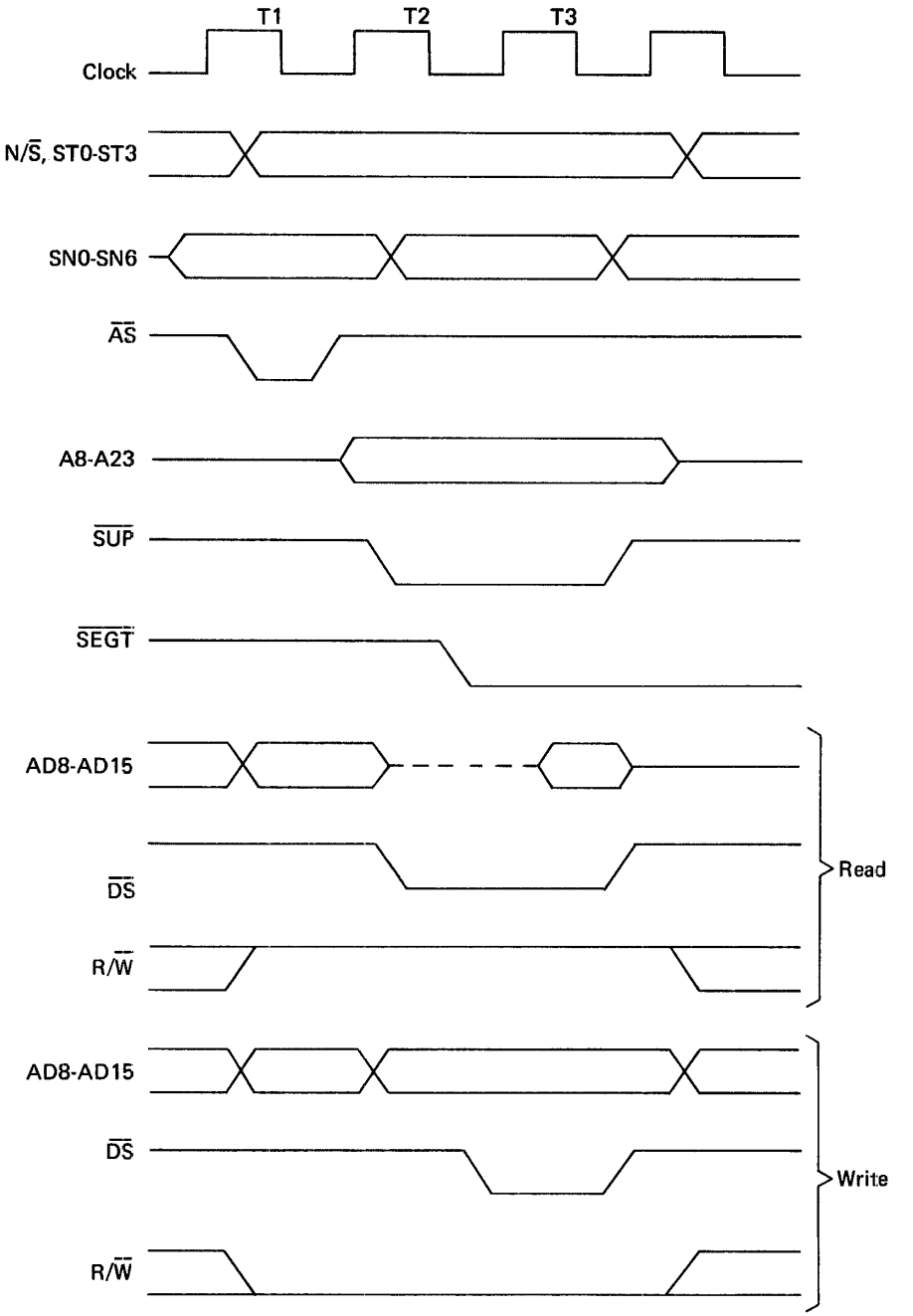

T1
T2
T3
Clock
N/S, ST0-ST3
SN0-SN6
AS
A8-A23
SUP
SEGT
AD8-AD15
DS
R/W
Read
AD8-AD15
DS
R/W
Write

Figure 5-12

Reset. If the $\overline{\text{RESET}}$ line is asserted, the following events take place:

- The Mode and Violation Type registers are cleared, as is the Descriptor Selection Counter.
- If the $\overline{\text{CS}}$ input is low, the Master Enable bit in the Mode register is set. All other register contents are undefined.
- Following reset, AD8-AD15 and A8-A23 are set to their high-impendance state. The $\overline{\text{SUP}}$ and $\overline{\text{SEGT}}$ open-drain outputs are not driven.
- Following the reset, the Master Enable flag in the Mode register must be set to enable operation.

Summary

A composite timing diagram showing the exact relationships of MMU signals is shown in Figure 5-13. Table 5-5 lists the ac characteristics of the Z8010 MMU. Following Table 5-5, are the dc characteristics of the MMU.

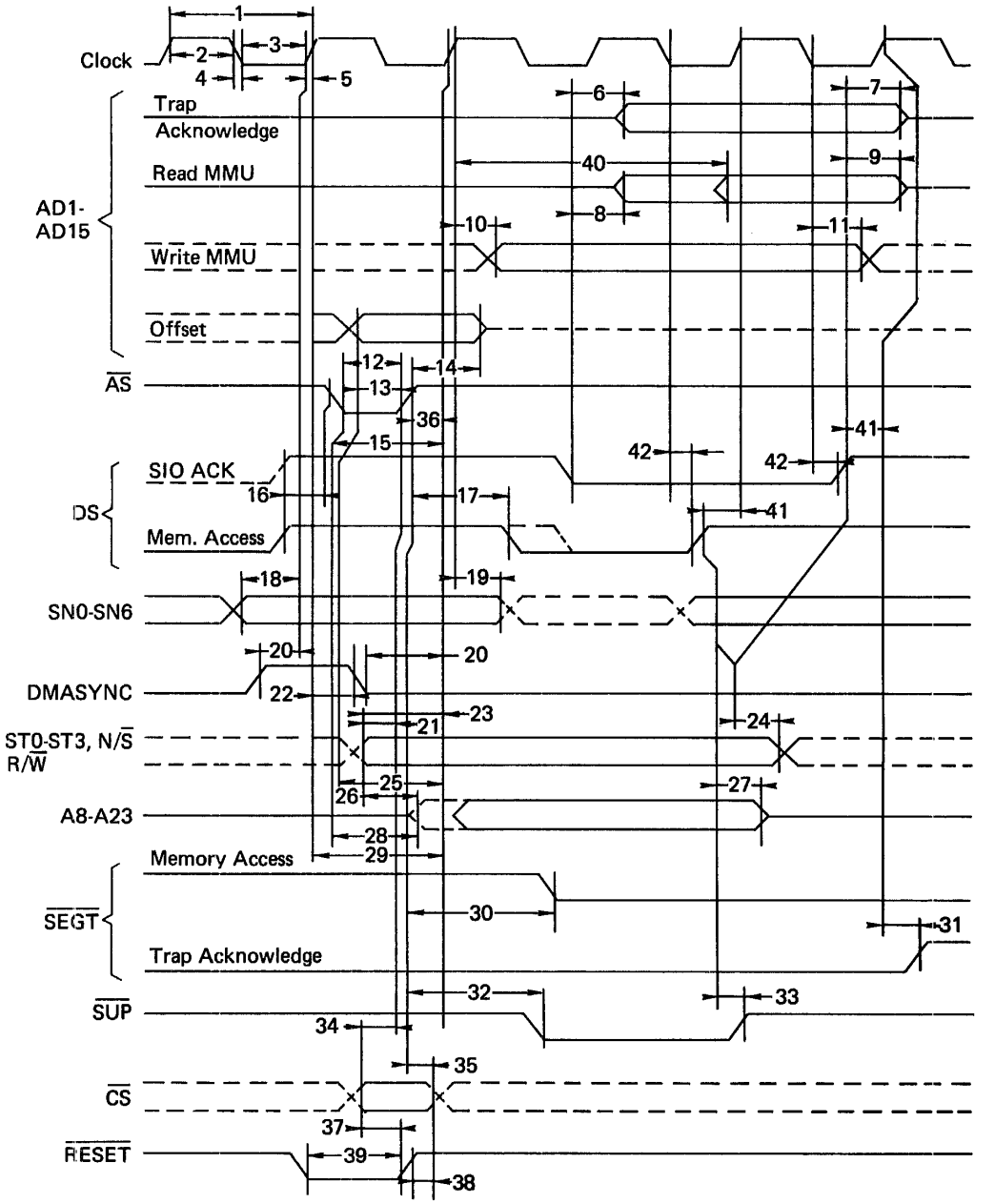

Clock
Trap Acknowledge
Read MMU
AD1-AD15
Write MMU
Offset
AS
SIO ACK
DS
Mem. Access
SN0-SN6
DMASYNC
ST0-ST3, N/S
R/W
A8-A23
Memory Access
SEGT
Trap Acknowledge
SUP
CS
RESET

Figure 5-13

Table 5-5
Z8010 MMU ac Timing Characteristics
(all times in nanoseconds)

No.	Parameter	Description	Min	Max
1	TcC	Clock Cycle Time	250	
2	TwCh	Clock Width (high)	105	
3	TwCl	Clock Width (low)	105	
4	TfC	Clock Fall Time		25
5	TrC	Clock Rise Time		25
6	TdDSA(RDv)	$\overline{DS}$ ↓ (acknowledge) to Read Data Valid Delay see note 1		100
7	TdDSA(RDf)	$\overline{DS}$ ↑ (acknowledge) to Read Data Float Delay see note 1	20	
8	TdDSR(RDv)	$\overline{DS}$ ↓ (read) to $\overline{AD}$ Output Driven Delay see note 1		100
9	TdDSR(RDf)	$\overline{DS}$ ↑ (read) to Read Data Float Delay see note 1	20	
10	TdC(WDv)	CLK ↑ to Write Data Valid Delay		160
11	ThC(WDn)	CLK ↓ to Write Data Not Valid Hold Time	30	
12	TwAS	Address Strobe Width	60	
13	TsOFF(AS)	Offset Valid to AS Setup Time	60	
14	ThAS(OFFn)	$\overline{AS}$ ↑ to Offset Not Valid Delay Hold Time	60	
15	TdAS(C)	$\overline{AS}$ ↓ to CLK ↑ Delay	110	
16	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	50	
17	TdAS(DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	50	
18	TsSN(C)	SN Data Valid to CLK ↑ Setup Time	120	
19	ThC(SNn)	CLK ↑ to SN Data Not Valid Hold Time	0	
20	TdDMAS(C)	DMASYNC Valid to CLK ↑ Delay	120	
21	TdSTNR(AS)	ST0-ST3, N/$\overline{S}$, R/$\overline{W}$ Valid to $\overline{AS}$ Delay	60	
22	TdC(DMA)	CLK ↑ to $\overline{DMASYNC}$ ↓ Delay	140	
23	TdST(C)	ST0-ST3 Valid to CLK ↓ Delay	140	
24	TdDS(STn)	$\overline{DS}$ ↑ to Status Not Valid Delay	0	
25	TdOFF(Av)	Offset Valid to Address Output Valid Delay see notes 1, 3, and 5		175
26	TdST(Ad)	Status Valid to Address Ouput Driven Delay see notes 1, 3, and 4		155
27	TdDS(Af)	$\overline{DS}$ ↑ to Address Output Float Delay see note 1	30	
28	TdAS(Ad)	$\overline{AS}$ ↓ to Address Output Driven Delay see notes 1 and 3		145
29	TdC(Av)	CLK ↑ to Address Output Valid Delay see notes 1 and 3		255
30	TdAS(SEGT)	$\overline{AS}$ ↑ to $\overline{SEGT}$ ↓ Delay see notes 1 and 2		160

Table 5-5 (cont.)
Z8010 MMU ac Timing Characteristics
(all times in nanoseconds)

No.	Parameter	Description	Min	Max
31	TdC(SEGT)	CLK ↑ to $\overline{SEGT}$ ↑ Delay see notes 1 and 2		300
32	TdAS(SUP)	$\overline{AS}$ ↑ to $\overline{SUP}$ ↓ Delay see notes 1 and 2		150
33	TdDS(SUP)	$\overline{DS}$ ↑ to $\overline{SUP}$ ↑ Delay see notes 1 and 2	30	155
34	TdCS(AS)	Chip Select Input Valid to $\overline{AS}$ ↑ Setup Time	10	
35	ThAS(CSn)	$\overline{As}$ ↑ to Chip Select Input Not Valid Hold Time	80	
36	TdAS(C)	$\overline{AS}$ ↑ CLK ↑ Delay	0	
37	TsCS(RST)	Chip Select Input Valid to $\overline{RESET}$ ↑ Setup Time	150	
38	ThRST(CSn)	$\overline{RESET}$ ↑ to $\overline{CS}$ Input Not Valid Hold Time	0	
39	TwRST	$\overline{RESET}$ Width (low)	2TcC	
40	TdC(RDv)	CLK ↑ to Read Data Valid Delay see note 1		460
41	TdDS(C)	$\overline{DS}$ ↑ to $\overline{CLK}$ ↑ Delay	30	
42	TdC(DS)	CLK ↓ to $\overline{DS}$ ↑ Delay	0	110

Notes:
1. 50 pF load.
2. 2K pull-up
3. These times apply to Z8010-3 version
4. Clock ↑ to Address Valid for Z8001 is specified at 100 ns with 100 pF load. This delay is reduced to 80 ns with a 50 pF load.
5. Clock ↑ to Status Valid for Z8001 is specified at 110 ns with 100 pF load. This delay is reduced to 100 ns with a 50 pF load.

dc Characteristics

Table 5-6
Z8010 dc Characteristics

Parameter	Description	Minimum	Maximum	Units
VCH	Clock Input High Voltage	Vcc -0.4	Vcc +0.3	Volts
VCL	Clock Input Low Voltage	-0.3	0.45	Volts
VIH	Input High Voltage	2.0	Vcc +0.3	Volts
VIL	Input Low Voltage	-0.3	0.8	Volts
VOH	Output High Voltage	2.4		Volts
VOL	Output Low Voltage		0.4	Volts
IIL	Input Leakage		+/-10	uA
IOL	Output Leakage		+/-10	uA
ICC	Vcc Supply Current		300	mA

Maximum Ratings

Voltages on all inputs and outputs with respect to GND	-0.3 to +7.0 V
Ambient Temperature under bias	0 to 70 C
Storage Temperature	-65 to +150 C

Z8030 SERIAL COMMUNICATIONS CONTROLLER

Introduction

The Z8030 Serial Communications Controller (SCC) is a dual-channel communications device that provides complete modem control. Each channel is independent of the other to the extent of having separate clocks, baud-rate generators, and I/O control and data handling logic.

Z8030 SCC Architecture

A block diagram of the SCC is shown in Figure 5-14. As you can see, the communications interfaces are distinct and separate, joined only by a common internal bus. Each channel is controlled by its own control registers, which are programmed by the Z8000 device.

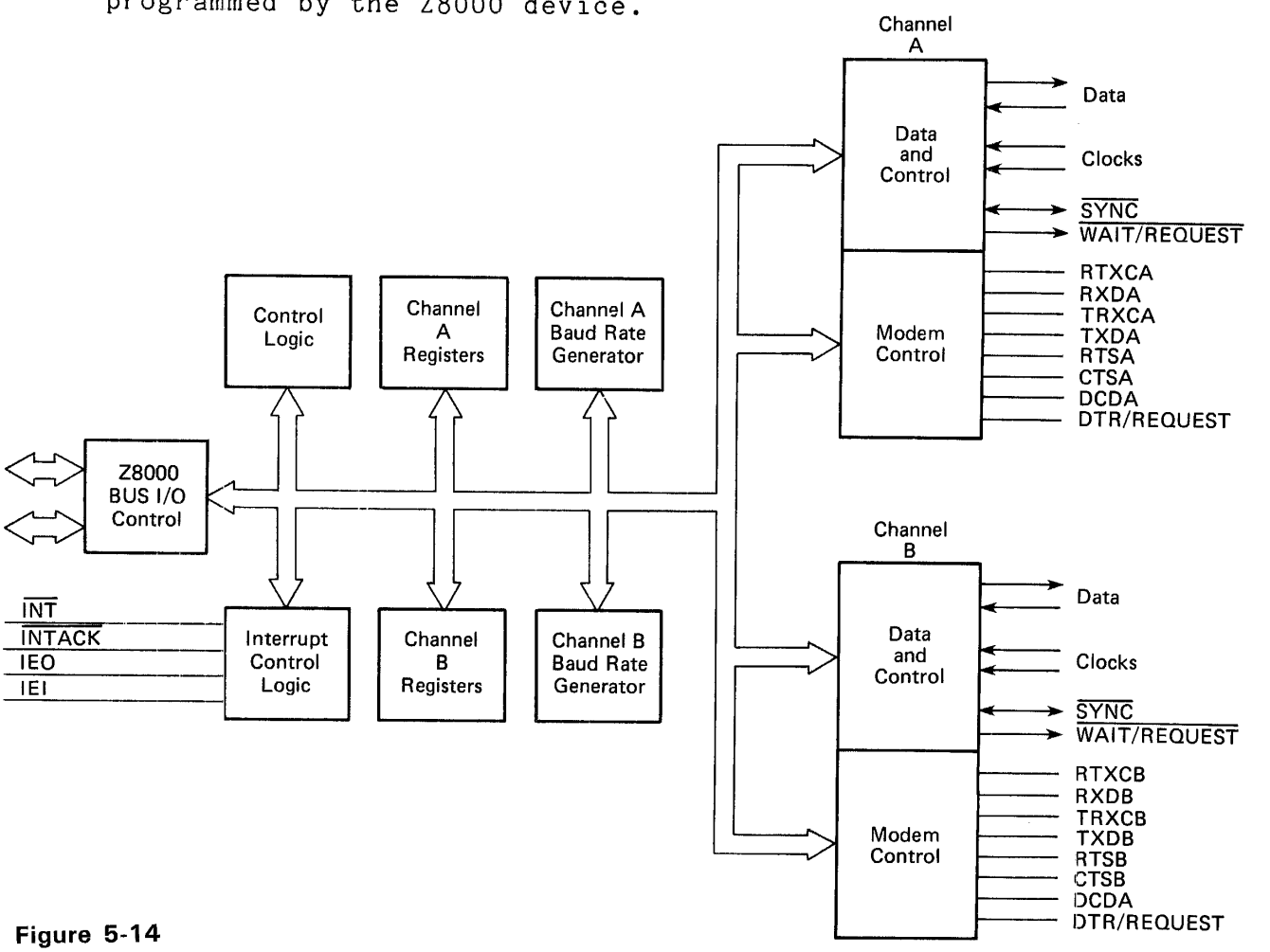


Figure 5-14

The Z8030 interfaces to the Z8000 through AD0-AD8 and five control lines. Each channel also provides interrupt logic to request service from the Z8000.

The baud rate generators are crystal controlled and contain phase-locked loops for clock stability. The channels are capable of communicating from 0 to 1 M-bits/second under program control.

The I/O channel protocol is selected under program control, as are the communications modes. The Z8030 performs cyclic-redundancy checks (CRC) on incoming data and generates CRC information for outgoing data.

Communications Protocol

The Z8030 SCC supports a number of communications protocols, both in hardware and programming. Table 5-7 lists the possible protocol variables.

Table 5-7
Z8030 Protocol

Characteristic	Capabilities
No. of Data Bits	5, 6, 7 or 8
No. Stop Bits/Char.	1, 1 1/2, or 2
Parity Checking	Programmable
Break Detection/ Generation	Programmable
Overrun Detection	Programmable
Framing Error Detection	Programmable

The Z8030 has the capability of communicating synchronously or asynchronously within a number of established protocols, such as:

- IBM Bisync
- IBM SDLC (Sychronous Data Link Control)
- HDLC

The Z8030 can also provide NRZ, NRZI, or FM data coding.

Programming the Z8030

Each channel in the Z8030 contains eight control registers, two sync-character registers, and four status registers. Each baud-rate generator contains two registers that hold time constant values for determining buad rates. The interrupt logic for each channel contains a register to hold the device's interrupt vector and three status registers: vector with status, vector without status, and interrupt pending status.

Z8030 Interface

The Z8030 pinout is shown in Figure 5-15. The following text describes each pin's function.

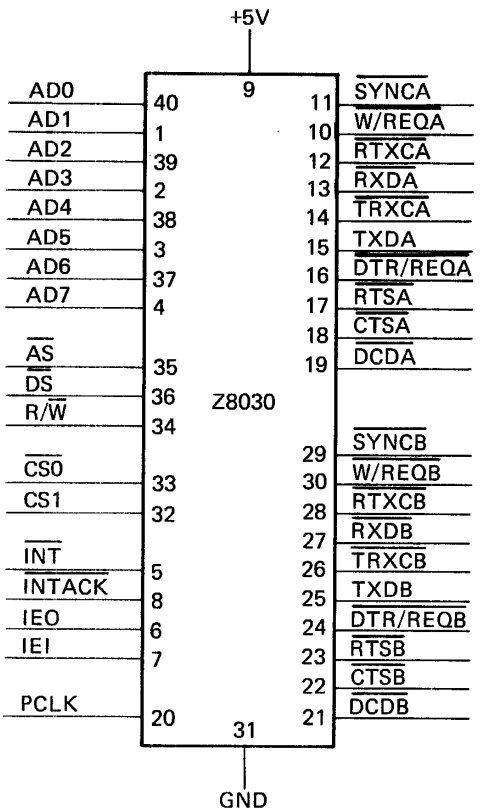


Figure 5-15

Z8000/Z8030.

AD0-AD7 Data information is passed between the two devices through these pins.

$\overline{AS}$ The address strobe ($\overline{AS}$) line is output from the Z8000 to indicate that address information on AD0-AD7 is valid.

$\overline{DS}$ The data strobe ($\overline{DS}$) line is output from the Z8000 to indicate that data on AD0-AD7 is valid.

R/$\overline{W}$ The read/write control line indicates whether the Z8000 is reading information from or writing information to the Z8030.

$\overline{CS0}$ The $\overline{CS0}$ input is one of two chip select lines input to the Z8030. The chip select inputs are used by the Z8000 to select between multiple SCCs.

CS1 The CS1 input is another chip select line and is used in conjunction with $\overline{CS0}$.

$\overline{INT}$ The $\overline{INT}$ output requests interrupt service from the Z8000 device.

$\overline{INTACK}$ This line is asserted by the Z8000 to acknowledge an interrupt request.

IEI The IEI (interrupt enable input) line is used in a daisy-chain interrupt structure, where interrupt priority is determined by physical proximity to the Z8000. The IEI line is an input recieved from the next higher-priority device on the bus. If the IEI line is not asserted, the Z8030 cannot generate an interrupt request ($\overline{INT}$). If the IEI line is asserted, the Z8030 can generate an interrupt.

IEO The IEO (interrupt enable output) is the other part of the Z8030 daisy-chained interrupt structure. The IEO output of one device is tied to the IEI input of the next lower priority device on the bus. If, for example, the Z8030 generated an interrupt, it would force its IEO line low, disabling any other devices with lower priority from generating an interrupt.

Z8030 I/O Channels

Since both Z8030 channels are alike, the following text describes only the channel A I/O lines.

TxDA/RxDA These two lines transmit and recieve serial data.

TRxCA/RTxCA These lines are used for remote clock input, using synchronous communications protocol. The TRxCA input is used to clock transmitted data,

	while the RTxCA input is used to clock received data.
SYNCx	The SYNCx input is a synchronizing input/output line.
W/REQ	The W/REQ (wait/request)line.
DTR/REQ	The DTR/REQ (data-terminal-ready/request) line.
RTSx	The RTSx output is the ready-to-send output.
CTSx	The CTSx input is a clear-to-send input.
DCDx	The DCDx input.

Z8036 COUNTER/TIMER - PARALLEL I/O UNIT

Introduction

The Z8036 Counter/Timer and Parallel I/O Unit (CIO) provides counter/timer functions, as well as three parallel I/O ports. The Z8036 CIO contains three independent 16-bit counters, two 8-bit I/O ports, and one 4-bit I/O port.

Either of the 8-bit ports can be configured as handshaking ports. The handshaking modes available are: IEEE-488, interlocked, strobed, and pulsed.

Counter/Timers

Two of the three 16-bit counters can be programmably joined to create a single 32-bit counter, or they can be used independently. Each counter can send its output to four I/O lines for direct control applications. Each counter has a programmable output duty cycle, can operate in single-cycle mode or continuously, and can be programmed to be retriggered or non-retriggerable.

The counter/timers are composed of a 16-bit down-counter, a 16-bit time constant register, a 16-bit current count register, an 8-bit control register, and an 8-bit status register.

Time Constant Register. The Time Constant register is used to contain the value loaded into the counter. A time constant value is loaded into this register by the Z8000. The value is then transfered to the down-counter as a initial value.

Current Count Register. The Current Count register contains the current value contained in the down-counter. The Current Count register may be read by the Z8000 to determine the contents of the down-counter.

Control Register. The Control register is used to program the counter/timers.

Status Register. The Status register contains counter/timer status information.

Because the Z8036 can perform as either a counter or a timer, each counter/timer circuit has access to up to four I/O lines. The lines serve as inputs to control the counter/timers and as outputs for the counter/timers. The I/O lines are organized as follows:

- Counter Input--used as the input when the counter/timer is running in counter mode.
- Gate Input
- Trigger Input--if the counter is running in retriggerable mode, this input serves as the trigger input.
- Counter/Timer Output--this is the output line for the counter/timer. The counter/timer duty cycle can be programmed to provide these outputs: pulse, one-shot, or square wave.

The counter/timers can also serve as handshake controllers, providing timing information to peripherals.

Parallel I/O Ports

The Z8036 I/O ports are very flexible. Ports A and B can act as 8-bit parallel input, output, or bidirectional I/O ports. In addition, the direction of each bit can be programmed to be input, output, or bidirectional. Port C has the same capability, but is limited to four bits.

Port B serves as the I/O port for two of the counter/timers. If the third counter/timer requires an I/O port, Port C serves. However, any unused lines can still be used as general purpose I/O lines.

Each of the ports contains pattern recognition logic. This allows the Z8036 to generate an interrupt if a specific pattern is detected. Pattern recognition lets the Z8036 to be configured as an interrupt priority controller.

Programming the Z8036 Ports

Each of the three I/O ports contains thirteen registers. The registers perform the following functions:

Input Register. This register accepts incoming data from the peripheral.

Ouput Register. The Output register holds outgoing data.

Buffer Register. The Buffer register serves as a short-term storage medium for the Input and Output registers.

Mode Specification Register. This register defines the mode of operation for the port: i.e., 8-bit output only, 8-bit input only, etc.

Handshake Specification Register. This register defines the type of handshake to be used by the port, if any.

Pattern Mask Register The Pattern Mask register contains the bit pattern to be compared against incoming data.

Pattern Polarity Register. The Pattern Polarity register determines the polarity of the pattern checking logic.

Pattern Transition Register. This register holds the transition pattern for pattern checking logic.

Data Path Polarity Register. This register contains the polarity (positive or negative logic) for each bit path.

Data Direction Register. This register contains the direction of data flow (input, output, or bidirectional) for each bit path.

Special I/O Control Register.

Interrupt Vector Register. The Interrupt Vector register contains the interrupt vector of the Z8036.

Primary Control and Status Register. After all the other registers are initialized, the Primary Control and Status register is usually the only register that need be accessed. Primary control and monitoring of port activities are provided by this register.

Z8036 Interface

The pinout for the Z8036 is provided in Figure 5-16. The following text describes each line.

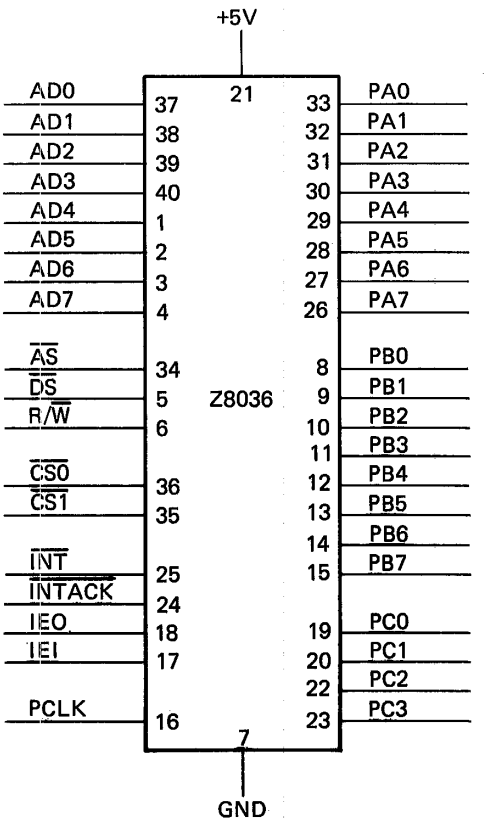


Figure 5-16

Input.

AD0-AD7. These are the communication lines between the Z8000 and the Z8036 CIO.

$\overline{AS}$. The $\overline{AS}$ (address strobe) line is issued by the Z8000 to indicate that it is presenting a stable address on AD0-AD7.

$\overline{DS}$. The $\overline{DS}$ (data strobe) line indicates that stable data is available on AD0-AD7.

R/$\overline{W}$. This line indicates whether the Z8000 is writing to or reading from the Z8036.

$\overline{CS0}$/CS1. These are the chip select inputs to the Z8036. They are used in a multi-CIO environment to determine which device is being accessed.

$\overline{INT}$/$\overline{INTACK}$/IEI/IEO. These control lines provide interrupt handling for the Z8036. See Chapter 4 for more detail about interrupts.

Outputs.

PA0-PA7. These are the lines for Port A. The direction of data flow can be selected for each line individually.

PB0-PB7. These are the I/O lines for Port B. In addition to serving as a normal I/O port, Port B can be configured to provide I/O capability for two counter/timers.

PC0-PC3. This is the Z8036's 4-bit port. Port C can be configured as a normal I/O port, provide handshaking services for another port, or act as the I/O interface for the third counter/timer.

2167 16K x 1 STATIC RAM

Introduction

The 2167 is a product of the Intel Corporation and is a 16,384 x 1 static RAM device. The 2167 comes in a 20-pin package, which allows high-density memories to be built on relatively small circuit boards.

The 2167 is fast enough to run with the Z8000 at maximum speed. An additional plus of the 2167 is its power-down feature, which causes the device to draw less current when not selected.

2167 Pinout

The 2167 has a relatively simple pinout structure, as illustrated in Figure 5-17. The following defines each 2167 pin.

A0-A13. These are the address select pins for the 2167.

Dout. This is the single data output pin.

Din. This is the data input pin.

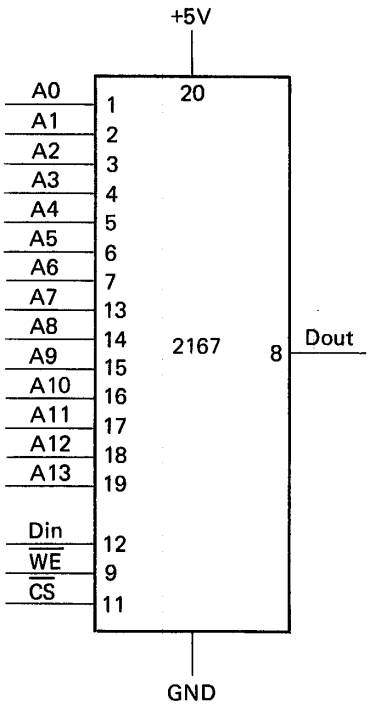


Figure 5-17

$\overline{\text{WE}}$. The state of the $\overline{\text{WE}}$ (write enable) pin determines whether the device is to be written to or read from.

$\overline{\text{CS}}$. The $\overline{\text{CS}}$ (chip select) pin is used to select a particular device for access. The $\overline{\text{CS}}$ pin also controls the 2167's power-down feature. Within one cycle time after being deselected (the $\overline{\text{CS}}$ line goes high), the 2167 will reduce its power requirements by more than half.

ac Characteristics

Tables 5-8 and 5-9 list the ac characteristics of the 2167. Figures 5-18 and 5-19 illustrate the device's timing requirements.

Table 5-8
2167-55 ac Characteristics
Read Cycle
(all times are in nanoseconds)

Symbol	Definition	Min	Max
tRC	Read Cycle Timing (1)	55	
tAA	Address Access Time		55
tACS	Chip Select Access Time		55
tOH	Output Hold from Access Change	5	
tLZ	Chip Select to Output Low Z (2,3)	10	
tHZ	Chip Deselect to Output High Z (2,3)	0	30
tPU	Chip Select to Power-up Time	40	
tPD	Chip Deselect to Power-down Time		55

Notes:
1. All Read Cycle timings are referenced from the last valid address to the first transitioning address.
2. At any given temperature and voltage condition, tHZ maximum is less than tLZ minimum both for a given device and from device to device.
3. Transition is measured +/-500 mV from steady-state voltage with loading specified by manufacturer.
4. WE is high for Read Cycles.
5. Device is continuously selected. CS = VIL
6. Address valid prior to or coincident with CS transition low.

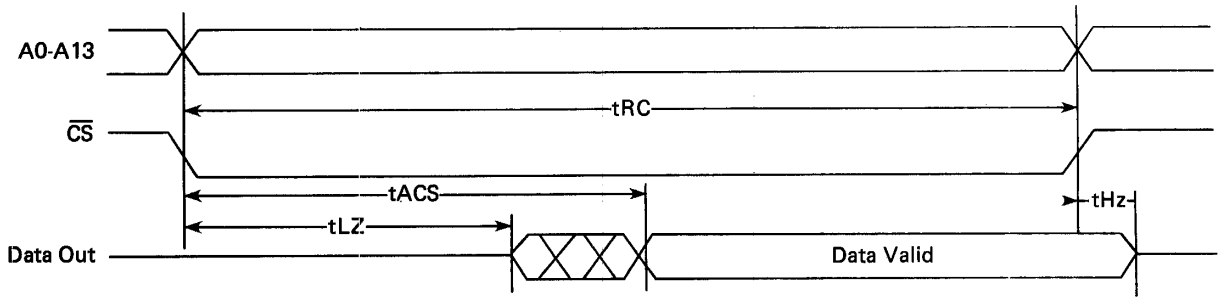


Figure 5-19 Write timing cycle.

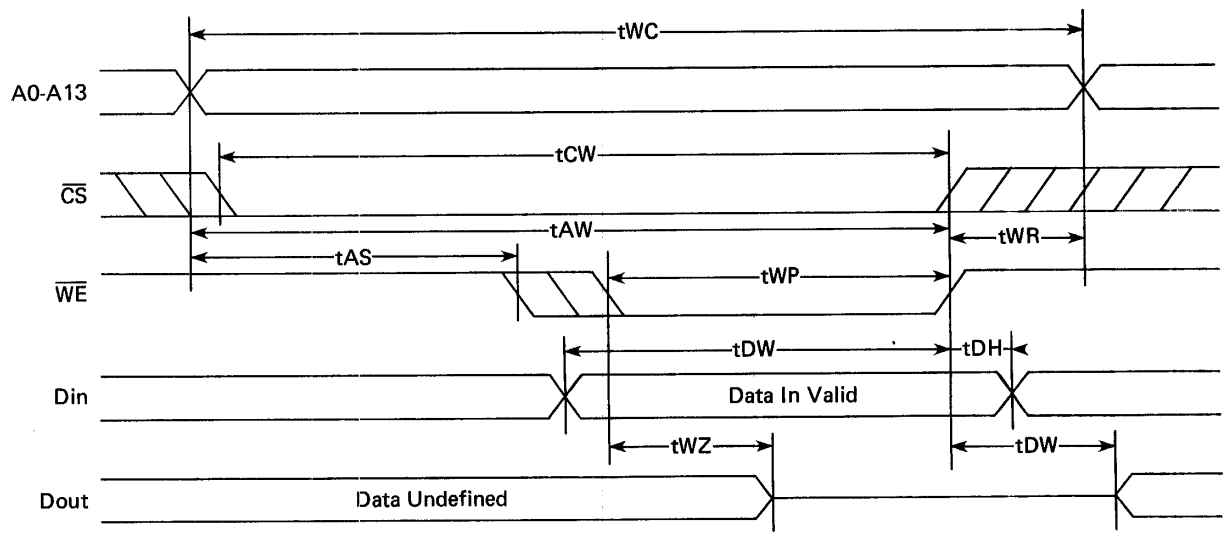


Figure 5-18 Read timing cycle.

Table 5-9
2167-55 ac Characteristics
Write Cycle
(all times are in nanoseconds)

Symbol	Definition	Min	Max
tWC	Write Cycle Time (2)	55	
tCW	Chip Select to End of Write	55	
tAW	Address Valid to End of Write	55	
tAS	Address Setup Time	0	
tWP	Write Pulse Width	35	
tWR	Write Recovery Time	0	
tDW	Data Valid to End of Write	25	
tDH	Data Hold Time	0	
tWZ	Write Enabled to Output in High Z (3)	0	25
tOW	Output Active from End of Write	0	

Notes: 1. If CS goes high simultaneously with WE high, the output remains in a high impedance state.
2. All Write Cycle timings are referenced from the last valid address to the first transition address.
3. Transition is measured +/-500 mV from steady-state voltage with specific loading specified by manufacturer.
4. CS or WE must be high during address transitions.

dc Characteristics

The dc characteristics for the 2167 are listed in Table 5-10.

Table 5-10
2167-55 dc Characteristics

Symbol	Definition	Min	Typ	Max	Units
ILI	Input Load Current (all inputs)		0.01	10	uA
ILO	Output Leakage Current		0.1	50	uA
ICC	Operating Current		90	120	mA
ISB	Standby Current		25	40	mA
IPO	Peak Power-on Current (1)		30	70	mA
VIL	Input Low Voltage	-3.0		0.8	V
VIH	Input High Voltage	2.0		6	V
VOL	Output Low Voltage			0.4	V
VOH	Output High Voltage	2.4			V

Notes: 1. A pull-up resistor to VCC on the CS input is required to keep the device deselected. Otherwise, power-on current approaches ICC activity.

Z80 PARALLEL I/O (PIO)

Introduction

The Z80 Parallel I/O device was developed by Zilog to support their 8-bit Z80 microprocessor. However, the structure of the Z80 PIO remains completely compatible with the Z8000.

The Z80 PIO provides two 8-bit parallel I/O ports. Each port can be specified as input, output, or bidirectional. In addition, each port may be configured as a control port, providing single-line control over data direction flow. In this way, the Z80 PIO is very similar to the Z8036 Counter/Timer device described earlier in this chapter.

Interrupt handling is fairly straightforward, in that the Z80 PIO is easily interfaced to the Z8000 interrupt structure. The Z80 PIO makes use of IEI and IEO pins to interface to a daisy-chain interrupt scheme.

Programming the Z80 PIO

The PIO operates in four modes: output with handshaking, input

with handshaking, bidirectional I/O with handshaking, and control mode. The modes are selected by writing a command byte to the PIO while selecting, with control lines, which port you want to configure. The command byte is illustrated in Figure 5-20. The four modes are selected by setting the most significant two bits of the command byte.

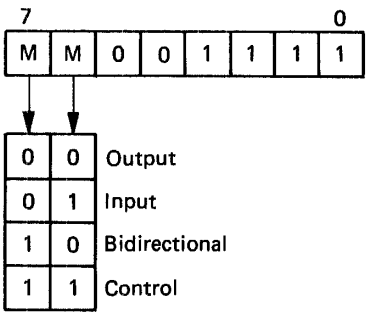


Figure 5-20

Each of these modes is described next.

Mode 0, 1, and 2. These modes are rather simple, in that the action of the port is exactly as described by the mode selection. Mode 0 forces to the port to transmit data only, mode 1 forces the port to receive data only, and mode 2 allows bidirectional data transfer.

Mode 3. Mode 3 is somewhat more complex than the other modes. If a port is configured for mode 3 operation, a second control byte must be issued to the port. The second byte is a pin direction mask, telling the PIO which pins are to transmit or recieve. The byte is strictly a binary mask, with each bit corresponding to a port I/O line. If, for example, bit 0 were set to 1, line 0 of the port would be **input** only. If bit 0 were set to 0, line 0 would be **output** only.

Two further bytes may be written to a port configured for mode 3 operation. You can specify what condition would cause the port to issue an interrupt to the Z8000. The interrupt control byte is shown in Figure 5-21. The high-order nybble of the byte selects how the PIO handles an interrupt situation.

Note that bit 4 is used to tell the PIO that an interrupt bit mask will follow. The bit mask, when used in combination with the other bits of the interrupt control byte, acts as a pattern matching device. In the bit mask byte, a 0 selects a pin for matching, while a 1 deselects a pin.

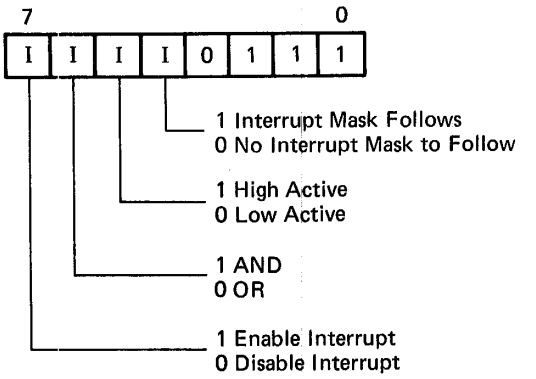


Figure 5-21

Bit 5 of the interrupt control byte sets the polarity for the pattern. A 1 in this position indicates positive logic (1 = true), while a 0 indicates negative logic (0 = true).

Bit 6 of the interrupt control byte lets you AND or OR the selected pins to generate an interrupt.

Bit 7 of the control byte enables and disables the PIO's ability to generate an interrupt.

Interrupts

As stated earlier, the Z80 PIO fits in well with the Z8000 interrupt scheme. Each parallel port can be assigned an interrupt identifier, by writing the byte shown in Figure 5-22. When an interrupt acknowledge is received from the Z8000, the PIO will place its identifier on D0-D7.

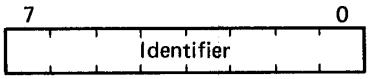


Figure 5-22

Z80 PIO Pinout and Timing

The Z80 PIO is a 40-pin device. Figure 5-23 illustrates the PIO pinout, and the text following describes each pin.

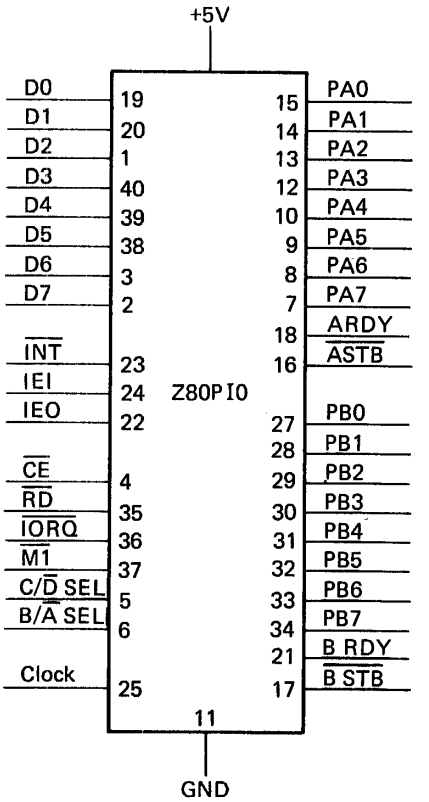


Figure 5-23

Z80 PIO Pinout.

D0-D7. These are the data pins that connect to the Z8000's AD0-AD7 lines.

$\overline{CE}$. The $\overline{CE}$ line is the chip select line for the PIO. Aside from selecting a chip, the Z8000 must select a port with the B/$\overline{A}$ SEL line.

B/$\overline{A}$ SEL. This control line selects which port is being accessed, Port A or Port B.

C/$\overline{D}$ SEL. The C/$\overline{D}$ SEL line indicates whether the processor is writing a data byte or a control byte to the PIO.

A RDY. This output is an indicator that Port A is ready for a transaction.

B RDY. The B RDY output, like the A RDY output, indicates that Port B is ready for a transaction.

$\overline{A\ STB}$. The $\overline{A\ STB}$ input is used as a handshaking mechanism to indicate that a peripheral device did indeed receive or send data. The $\overline{A\ STB}$ line is asserted by the peripheral upon receipt or transmission of data.

$\overline{B\ STB}$. This control line, like $\overline{A\ STB}$, is used as a handshaking mechanism to indicate receipt or transmission of data.

A0-A7. Port A I/O lines.

B0-B7. Port B I/O lines.

$\overline{M1}$. This control input is a hold-over from the Z80. The $\overline{M1}$ line is asserted by the Z80 to indicated the first cycle in a machine cycle. $\overline{M1}$ can be generated from the Z8000 status lines, and $\overline{DS}$ or $\overline{AS}$.

$\overline{IORQ}$. Like $\overline{M1}$, the $\overline{IORQ}$ line can be generated from the Z8000 status lines, and $\overline{DS}$ or $\overline{AS}$. $\overline{IORQ}$ serves as an I/O request line for the Z80.

$\overline{RD}$. The $\overline{RD}$ line, when low, indicates a read operation, and while high, indicates a write operation. This line can be tied directly to the Z8000 R/$\overline{W}$ output.

$\overline{INT}$. This is the interrupt request line.

IEI-IEO. These two lines are used in the daisy-chain interrupt scheme as described previously.

Z80 PIO Timing.

Mode 0 Timing. A timing diagram for mode 0 operation is shown in Figure 5-24. The PIO generates a psuedo-write pulse by ANDing its $\overline{RD}$, $\overline{CE}$, C/$\overline{D}$, and $\overline{IORQ}$ lines. The write pulse (labeled $\overline{WR}$ in Figure 5-24) goes low when all four inputs are asserted. At the high-to-low transition of $\overline{WR}$, data is strobed off D0-D7 and into the port output register.

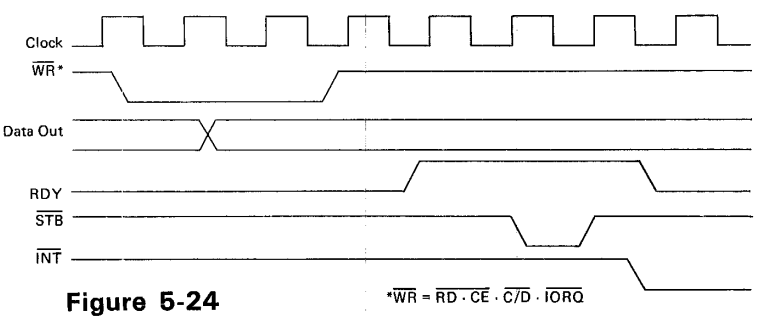


Figure 5-24

On the next high-to-low clock transition after $\overline{WR}$ returns high, the PIO asserts the ready (A RDY or B RDY) output line.

RDY stays high until the port strobe line ($\overline{\text{A STB}}$ or $\overline{\text{B STB}}$) is asserted by the peripheral indicating a receipt of the data. When the STB input is asserted, the PIO generates an interrupt request. The next time the clock makes a high-to-low transition, RDY goes low, ending the cycle.

Mode 1 Timing. A timing diagram for mode 1 operation is shown in Figure 5-25. In this mode, the information transfer cycle is started when the peripheral device asserts the strobe ($\overline{\text{A STB}}$ or $\overline{\text{B STB}}$) line, indicating that it's ready to send data to the PIO. When STB goes low, the PIO transfers data from its input pins (either A0-A7 or B0-B7) to the port input register. On the rising edge of the STB, the PIO generates an interrupt request.

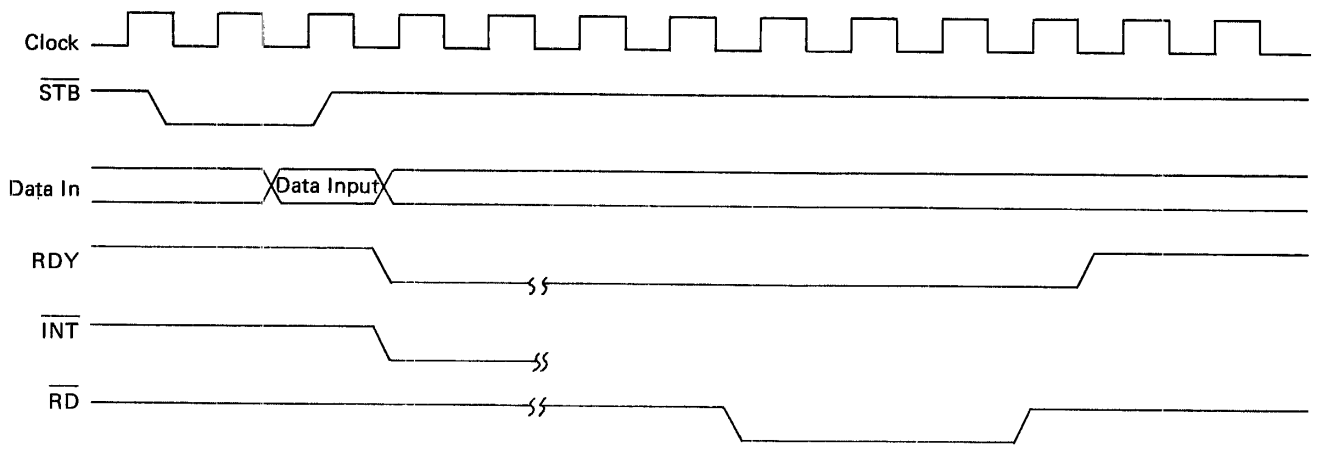


Figure 5-25

On the next falling edge of the clock, the PIO forces the port RDY line low, indicating that data has been received. When the Z8000 reads the data, the RDY line goes back high. Note that the Z8000 must not transmit data to the PIO while the RDY line is low, otherwise the received data will be lost.

Mode 2 Timing. Both Figures 5-24 and 5-25 represent the timing for a port in mode 2. A useful way to control bidirectional transfer would be to assign Port A as the data port, and to configure Port B as a mode 3 control port. However, it's beyond the scope of this book to explore this possiblity.

Mode 3 Timing. Since mode 3 timing is entirely dependent upon the way the port is programmed, it would be unreasonable to attempt to provide timing information.

APPENDIX

Introduction

A list of the Z8000 instructions are given in Table A-1. The instruction mnemonics are listed in alphabetic order, and provide the number of clock cycles used, the number of bytes per instruction, a brief description of the instruction, and the page number in Chapter 2 describing the instruction.

Table A-1
Z8000 Instruction Set

Mnemonic	Cycles	Bytes	Description
ADC	5	2	ADD words with carry 113
ADCB	5	2	ADD bytes with carry 114
ADD	9-13	4/6	ADD word to register 115
ADDB	9-13	4/6	ADD byte to register 117
ADDL	15-19	4/6	ADD long word to register 120
AND	9-13	4/6	AND word with register 189
ANDB	9-13	4/6	AND byte with register 191
BIT	10-14	4/6	Test bit in word 219
BITB	10-14	4/6	Test bit in byte 221
CALL	12-21	4/6	CALL subroutine 65
CALR	10-15	2	CALL subroutine relative 67
CLR	11-15	4/6	Clear word 122
CLRB	11-15	4/6	Clear byte 124
CLRL	5-18	2/4	Clear long word 126
COM	15-19	4/6	Complement word 128
COMB	15-19	4/6	Complement byte 130
COMFLG	7	2	Complement flag 61
CP	14-18	6/8	Compare register with word 193
CPB	14-18	6/8	Compare register with byte 195
CPL	8-19	2/6	Compare register with long word 198
CPSD	25	4	Compare memory word strings, autodecrement 232
CPSDB	25	4	Compare memory byte strings, autodecrement 233
CPSDR	11 + 14n	4	Compare memory word strings, autodecrement and repeat 235
CPSDRB	11 + 14n	4	Compare memory byte strings, autodecrement and repeat 236
CPSI	25	4	Compare memory word strings, autoincrement 238
CPSIB	25	4	Compare memory byte strings, autoincrement 239
CPSIR	11 + 14n	4	Compare memory word strings, autoincrement and repeat 241
CPSIRB	11 + 14n	4	Compare memory byte strings, autoincrement and repeat 242
DAB	5	2	Decimal adjust byte 132
DBJNZ	11	2	Decrement byte register and jump on non-zero result 68

Table A-1 (cont.)
Z8000 Instruction Set

Mnemonic	Cycles	Bytes	Description
DEC	4-17	2/6	Decrement word 134
DECB	4-17	2/6	Decrement byte 136
DI	7	2	Disable interrupt 42
DIV	107-112	2/6	Divide register pair by source word 138
DIVL	744-749	2/6	Divide register quad by source long word 140
DJNZ	11	2	Divide word register and jump on non-zero result 69
EI	7	2	Enable interrupt 43
EX	6-19	2/6	Exchange source and destination words 78
EXB	6-19	2/6	Exchange source and destination bytes 80
EXTS	11	2	Extend word sign 143
EXTSB	11	2	Extend byte sign 144
EXTSL	11	2	Extend long word sign 145
HALT	8 + 3n	2	Halt 44
IN	10-12	2/4	Input port word to register 258
INB	10-12	2/4	Input port byte to register 259
INC	4-17	2/6	Increment word 146
INCB	4-17	2/6	Increment byte 148
IND	21	4	Input port word to memory, autodecrement 260
INDB	21	4	Input port byte to memory, autodecrement 261
INDR	11 + 10n	4	Input port word to memory, autodecrement and repeat 263
INDRB	11 + 10n	4	Input port byte to memory, autodecrement and repeat 264
INI	21	4	Input port word to memory, autoincrement 266
INIB	21	4	Input port byte to memory, autoincrement 267
INIR	11 + 10n	4	Input port word to memory, autoincrement and repeat 268
INIRB	11 + 10n	4	Input port byte to memory, autoincrement and repeat 269
IRET	13-16	2	Return from interrupt 71
JP	7-15	2/6	Jump conditional 72
JR	6	2	Jump conditional relative 74

Table A-1 (cont.)
Z8000 Instruction Set

Mnemonic	Cycles	Bytes	Description
LD	11-18	4/6	Load immediate word into memory 82
LDA LDAR	12-15	4/6	Load address into register 86
LDAR	15	4	Load relative address into register 88
LDB	11-18	4/6	Load immediate byte into memory 84
LDCTL	7	2	Load control register to/from register 45
LDCTLB	7	2	Load flag byte to/from register 46
LDD	20	4	Load memory word to memory, autodecrement 89
LDDB	20	4	Load memory byte to memory, autodecrement 90
LDDR	11 + 9n	4	Load memory word to memory, autodecrement and repeat 91
LDDRB	11 + 9n	4	Load memory byte to memory, autodecrement and repeat 92
LDI	20	4	Load memory word to memory, autoincrement 93
LDIB	20	4	Load memory byte to memory, autoincrement 94
LDIR	11 + 9n	4	Load memory word to memory, autoincrement and repeat 95
LDIRB	11 + 9n	4	Load memory byte to memory, autoincrement and repeat 96
LDK	5	2	Load immediate digit into register 97
LDL LDRL	5-17	2/6	Load long word to/from register 98
LDM	11 + 3n	4/8	Load multiple registers to/from memory 101
LDPS	12-23	2/6	Load program status 103
LDR	14	4	Load relative word into/from register 105
LDRB	14	4	Load relative byte into/from register 107
LDRL	17	4	Load relative long word into/from register 109
MBIT	7	2	Multi-microprocessor test 47
MREQ	12 + 7n	2	Multi-microprocessor request 48
MRES	5	2	Multi-microprocessor reset 50
MSET	5	2	Multi-microprocessor set 51
MULT	70-75	2/6	Multiply register with word 150
MULTL	282 + 7n	2/6	Multiply register with long word 152
NEG	7-19	2/6	Negate word 154
NEGB	7-19	2/6	Negate byte 156

Table A-1 (cont.)
Z8000 Instruction Set

Mnemonic	Cycles	Bytes	Description
NOP	7	2	No operation 52
OR	4-13	2/6	OR word with register 200
ORB	4-13	2/6	OR byte with register 202
OTDR	11 + 10n	4	Output memory word to I/O port, autodecrement and repeat 271
OTDRB	11 + 10n	4	Output memory byte to I/O port, autodecrement and repeat 272
OTIR	11 + 10n	4	Output memory word to I/O port, autoincrement and repeat 274
OTIRB	11 + 10n	4	Output memory byte to I/O port, autoincrement and repeat 275
OUT	10-12	2/4	Output register word to I/O port 277
OUTB	10-12	2/4	Output register byte to I/O port 278
OUTD	21	4	Output memory word to I/O port, autodecrement 280
OUTDB	21	4	Output memory byte to I/O port, autodecrement 281
OUTI	21	4	Output memory word to I/O port, autoincrement 282
OUTIB	21	4	Output memory byte to I/O port, autoincrement 283
POP	8-19	2/6	Pop word from stack 53
POPL	12-26	2/6	Pop long word from stack 55
PUSH	9-17	2/6	Push word onto stack 57
PUSHL	12-24	2/6	Push long word onto stack 59
RES	4-17	2/6	Reset bit in word 223
RESB	4-17	2/6	Reset bit in byte 225
RESFLG	7	2	Reset flag 62
RET	10,13 T 7,7 F	2	Return conditional from subroutine 75
RL	6-7	.2	Rotate word left 158
RLB	6-7	2	Rotate byte left 159
RLC	6-7	2	Rotate word left through carry 160
RLCB	6-7	2	Rotate byte left through carry 161
RLDB	9	2	Rotate byte digit left 162
RR	9	2	Rotate word right 164
RRB	6-7	2	Rotate byte right 165
RRC	6-7	2	Rotate word right through carry 166
RRCB	6-7	2	Rotate byte right through carry 167
RRDB	9	2	Rotate byte digit right 168

Table A-1 (cont.)
Z8000 Instruction Set

Mnemonic	Cycles	Bytes	Description
SBC	5	2	Subtract word with carry 170
SBCB	5	2	Subtract byte with carry 171
SC	33,39	2	System call 76
SDA	15 + 3n	4	Shift word arithmetic 178
SDAB	15 + 3n	4	Shift byte arithmetic 179
SDAL	15 + 3n	4	Shift long word arithmetic 180
SDL	15 + 3n	4	Shift word logical 204
SDLB	15 + 3n	4	Shift byte logical 205
SDLL	15 + 3n	4	Shift long word logical 207
SET	4-17	2/6	Set bit in word 227
SETB	4-17	2/6	Set bit in byte 229
SETFLG	7	2	Set flag 63
SIN	12	2	Special input I/O word to register 284
SINB	12	2	Special input I/O byte to register 285
SIND	21	4	Special input I/O word to memory, autodecrement 286
SINDB	21	4	Special input I/O byte to memory, autodecrement 287
SINDR	11 + 10n	4	Special input I/O word to memory, autodecrement and repeat 288
SINDRB	11 + 10n	4	Special input I/O byte to memory, autodecrement and repeat 289
SINI	21	4	Special input I/O word to memory, autoincrement 291
SINIB	21	4	Special input I/O byte to memory, autoincrement 292
SINIR	11 +10n	4	Special input I/O word to memory, autoincrement and repeat 293
SINIRB	11 + 10n	4	Special input I/O byte to memory, autoincrement and repeat 294
SLA	13 + 3n	4	Shift word arithmetic left 182
SLAB	13 + 3n	4	Shift byte arithmetic left 183
SLAL	13 + 3n	4	Shift long word arithmetic left 184
SLL	13 + 3n	4	Shift word logical left 208
SLLB	13 + 3n	4	Shift byte logical left 209
SLLL	13 + 3n	4	Shift long word logical left 210

Table A-1 (cont.)
Z8000 Instruction Set

Mnemonic	Cycles	Bytes	Description
SOTDR	11 + 10n	4	Special output memory word to port, autodecrement and repeat 296
SOTDRB	11 + 10n	4	Special output memory byte to port, autodecrement and repeat 297
SOTIR	11 + 10n	4	Special output memory word to port, autoincrement and repeat 299
SOTIRB	11 + 10n	4	Special output memory byte to port, autoincrement and repeat 300
SOUT	12	4	Special output register word to port 302
SOUTB	12	4	Special output register byte to port 303
SOUTD	21	4	Special output memory word to port, autodecrement 304
SOUTDB	21	4	Special output memory byte to port, autodecrement 305
SOUTI	21	4	Special output memory word to port, autoincrement 306
SOUTIB	21	4	Special output memory byte to port, autoincrement 307
SRA	13 + 3n	4	Shift word arithmetic right 185
SRAB	13 + 3n	4	Shift byte arithmetic right 186
SRAL	13 + 3n	4	Shift long word arithmetic right 187
SRL	13 + 3n	4	Shift word logical right 211
SRLB	13 + 3n	4	Shift byte logical right 212
SRLL	13 + 3n	4	Shift long word logical right 213
SUB	4-13	2/6	Subtract word from register 172
SUBB	4-13	2/6	Subtract byte from register 174
SUBL	8-19	2/6	Subtract long word from register 176
TRDB	25	4	Translate byte, autodecrement 244
TRDRB	11 + 14n	4	Translate byte, autodecrement, and repeat 245
TRIB	25	4	Translate byte, autoincrement 247
TRIRB	11 + 14n	4	Translate byte, autoincrement and repeat 248
TRTDB	25	4	Translate and test byte, autodecrement 250
TRTDRB	11 + 14n	4	Translate and test byte, autodecrement and repeat 251
TRTIB	25	4	Translate and test byte, autoincrement 253
TRTIRB	11 + 14n	4	Translate and test byte, autoincrement and repeat 254

Table A-1 (cont.)
Z8000 Instruction Set

Mnemonic	Cycles	Bytes	Description
XOR	4-13	2/6	Exclusive OR word with register 214
XORB	4-13	2/6	Exclusive OR byte with register 216
XCTL	14	4	Internal EPU operation 309
XLDM	11 + 3n	4	Load CPU from EPU 312, 313, 314, 316
XLDCTL	14	4	Load FCW from EPU 310, 311