Z8 Microcomputer

Technical Manual

April 1983

Z8 Microcomputer Technical Manual

Zilog

Table Of Contents

Table Of Contents (Continued)

Chapter 7. External Interface (Z8681, Z8682)

Chapter 8. Reset and Clock

Chapter 9. I/O Ports

Table Of Contents (Continued)

Table Of Contents (Continued)

List of Illustrations

Table Of Contents (Continued)

List of Tables

Chapter 1
Z8 Family Overview

1.1 INTRODUCTION

This chapter provides an overview of the architecture and features of the Z8 Family of products, with particular emphasis on those features that set this microcomputer apart from earlier microcomputers. Detailed information about the architecture, address spaces and modes, instruction set, external interface, timing, input/output operations, and interrupts can be found in subsequent chapters of this manual.

1.2 FEATURES

The Z8 microcomputer introduces a new level of sophistication to single-chip architecture. Compared to earlier single-chip microcomputers, the Z8 offers faster execution; more efficient use of memory; more sophisticated interrupt, input/output and bit-manipulation capabilities; and easier system expansion.

Z8 products offer the standard on-chip functions of earlier microcomputers, including:

- 2K or 4K bytes of ROM
- 144 8-bit registers
- 32 lines of programmable I/O
- Clock oscillator
- Arithmetic logic unit
- Parallel and serial ports

Beyond these basic features, the Z8 Family offers such advanced characteristics as:

- Two counter/timers
- Six vectored interrupts
- UART for serial I/O communication
- Stack functions
- Power-down option
- TTL compatibility
- Optimized instruction set
- BASIC/Debug interpreter

All members of the Z8 Family are variations of the basic Z8 microcomputer, the Z8601/11. The Z8 Family includes a development device (Z8612), a ROMless device (Z8681/82), BASIC/Debug Interpreter (Z8671), a Protopack emulator (Z8603/13), as well as the basic microcomputer. These products offer all the parts and development tools necessary for systems development (both hardware and software prototyping), field trials (pre-production) and full production. For prototyping and preproduction, or where code flexibility is important, the Z8603/13 Protopack, 2K and 4K EPROM-based parts are the most appropriate. The ROM-based Z8601/11 microcomputers are used in high-volume production applications after the software has been perfected. For ROMless applications, two versions of the Z8 microcomputer are available: the 40-pin Z8681/82 and the 64-pin Z8612. In addition, there is a military version of the Z8611 4K ROM device, available in both 40-pin ceramic and 44-pin leadless chip carrier packages.

The Z8671 MCU is a complete microcomputer preprogrammed with a BASIC/Debug Interpreter. This device, operating with both external ROM or RAM and on-chip memory registers, is suitable for most industrial control applications, or whenever fast and efficient program development is necessary.

The Z8 microcomputer is well-suited for dedicated control applications in real-time mode. Since speed is a key consideration in such applications, the Z8 Family is available in both 8 and 12 MHz versions, supported by either of two development modules: the Development Module ($\overline{DM}$) or the Z-SCAN 8. The Z-SCAN module provides (ICE) in-circuit emulation capability.

1.2.1 Instruction Set

The Z8 instruction set, consisting of 43 basic instructions, is optimized for high-code density and reduced execution time. The 47 instruction types and six addressing modes--together with the ability to operate on bits, 4-bit words, BCD digits, 8-bit bytes, and 16-bit words--make for a code-efficient, flexible microcomputer.

1.2.2 Architecture

Z8 architecture offers more flexibility and performance than previous A/B accumulator designs. All 128 general-purpose registers, including

dedicated I/O port registers, can be used as accumulators. This eliminates the bottleneck commonly found in A/B devices, particularly in high-speed applications such as disk drives, printers and terminals. In addition, the registers can be used as address pointers for indirect addressing, as index registers or for implementing an on-chip stack. Speed of execution and smooth programming are supported by a "working register area"--short 4-bit register addresses.

Table 1-1 lists the basic characteristics of the members of the Z8 Family. As shown, the major differences between the products are in their physical packaging and the manner in which address space is handled. An overall description for each Z8 type is given in the following sections. Variations within each group are specified where applicable.

Table 1-1. Z8 Family of Products

Product	Part Number	ROM Capacity (Bytes)	Programmable I/O Pins	Dedicated I/O Pins	PCB Footprint	Comments
2K ROM	Z8601	2K	32, 4 ports	8 Power, Control	40 Pin	Masked ROM part, used primarily for high volume production.
2K Protopack	Z8603	0	32, 4 ports	8 Power, Control plus 24 EPROM	40 Pin	Piggyback part used where program flexibility is required (prototyping).
4K ROM	Z8611	4K	32, 4 ports	8 Power, Control	40 Pin	Masked ROM part, used primarily for high volume production.
4K Development part	Z8612	0	32, 4 ports	8 Power, Control plus 24 external memory	64 Pin	ROMless part used primarily in development systems.
4K Protopack	Z8613	0	32, 4 ports	8 Power, Control plus 24 EPROM	40 Pin	Piggyback part used where program flexibility is required (prototyping).
BASIC/ Debug	Z8671	2K	32, 4 ports	8 Power, Control	40 Pin	BASIC/Debug part used in low volume applications.
ROMless	Z8681/82	0	24, 3 ports	8 Power, Control plus 8 external memory	40 Pin	Low cost ROMless production part with reduced I/O. Program memory is external.

1.3 MICROCOMPUTERS (Z8601/Z8611)

The Z8 can be a stand-alone microcomputer with either 2K bytes (Z8601) or 4K bytes (Z8611) of internal ROM, a traditional microprocessor that can manage up to 124K bytes (Z8601) or 120K bytes (Z8611) of external memory, or a parallel processing element in a system with other processors and peripheral controllers linked by a Z-BUS. In all configurations, a large number of device pins are available for I/O. Key features of the Z8601/11 microcomputer include:

- **ROM 2K-byte (Z8601) or 4K-byte (Z8611) Program Memory.** This ROM is mask-programmed during production with user-provided programs.

- **144-byte RAM Register File.** The internal register organization of the Z8 microcomputer centers around a 144-byte file composed of 124 general-purpose registers, 16 status and control registers, and 4 I/O port registers. Either an 8-bit or a 4-bit address mode can be used to access the register file. When the 4-bit mode is used, the register file is divided into 9 groups of 16 working registers each. A Register Pointer uses short-format instructions to quickly access any one of the nine groups. Use of the 4-bit addressing mode decreases access time and improves throughput.

- **Programmable Counter/Timers.** Two 8-bit counter/timer circuits are provided, each driven by its own prescaler. Both the counter/timers and their prescaler circuits are programmable.

- **UART (Universal Asynchronous Receiver Transmitter).** A full-duplex UART is provided to control serial data communications. One of the on-chip counter/timer circuits provides the required bit rate input to enable the UART to operate at a maximum data transfer rate of 93.75K bits per second at a crystal frequency of 12 MHz.

- **I/O Lines/Ports.** The Z8 microcomputer provides 32 input/output lines, arranged as 4 8-bit ports. Under software control, the I/O ports (Ports 0, 1, 2, 3) can be programmed as input, output, or additional address lines. The I/O ports can also be programmed to provide timing, status signals, interrupt inputs and serial or parallel I/O (with or without handshake).

- **Vectored Interrupts.** The Z8 MPU permits the use of six different interrupts from any of eight different sources. Four Port 3 lines ($P3_0$-$P3_3$), serial input pin ($P3_0$), the serial output pin ($P3_7$) and both counter/timer circuits may be interrupt sources. All interrupts are vectored and are both maskable and prioritized.

- **Oscillator Circuit.** An oscillator circuit that can be driven from an external clock or crystal is provided on the Z8 microcomputer. The oscillator will accept an input frequency of up to 12 MHz on the two input pins provided.

- **Optional Power-Down Feature.** This option permits normal input power to be removed from the chip without affecting the contents of the register file. The power-down function requires an external battery backup system.

Pin functions and descriptions for the Z8601/11 microcomputer can be found in Chapter 6.

1.4 DEVELOPMENT DEVICE (Z8612)

A development device allows users to prototype a system with an actual hardware device and to develop the code that is eventually mask-programmed into the on-chip ROM of the Z8601 or Z8611 microcomputer. Development devices are also useful in applications where production volume does not justify the expense of a ROM system. The Z8612 development device is identical to its equivalent microcomputer, the Z8611, with the following exceptions:

- No internal ROM is provided, so that code is developed in an off-chip memory.

- The normally internal ROM address and data lines are buffered and brought out to external pins to interface with the external memory.

- Control lines are added to interface with external program memory.

- The device package is enlarged in order to accommodate the new control, address, and data lines.

Pin functions and descriptions for the development device can be found in the Appendix.

1.5 PROTOPACK EMULATOR (Z8603/13)

The Protopack emulator devices, Z8603 and Z8613, are ROMless versions of their equivalent microcomputers (Z8601 and Z8611, respectively). The emulators differ from development devices in two ways: they use the same pinout as the microcomputers, and an external ROM or EPROM can be plugged into the top of the package. The emulator package allows for flexibility of application, since it can be used in either prototype or final pc boards, yet still allows for program development.

When the final program is developed, it can be mask-programmed into the Z8601/11 which then replaces the emulator. The emulator is also useful in small volume applications where the cost of mask-programming is prohibitive or where program flexibility is desired.

Physical description for the Protopack emulator is found in the Appendix.

1.6 BASIC/DEBUG INTERPRETER (Z8671)

The Z8671 MCU is a complete microcomputer preprogrammed with a BASIC/Debug interpreter. BASIC/Debug can directly address the Z8671's internal registers and all external memory. It can quickly examine and modify any external memory location or I/O port, and can call machine language subroutines to increase execution speed.

The Z8671 MCU has a combination of software and hardware that is ideal for most industrial control applications. Along with the functions mentioned above, this microcomputer has a self-contained line editor for interactive debugging which further speeds program development. In addition the BASIC/Debug Interpreter allows program execution on power-up or reset, without operator intervention.

Two kinds of memory exist in the Z8671 device: on-chip registers and external ROM or RAM. The BASIC/Debug interpreter is located in the 2K bytes of on-chip ROM. Maximum addressing capability is 62K bytes of external program memory and 62K bytes of data memory. In addition, 32 I/O lines, a 144-byte register file, on-board UART and two counter/timers are provided.

Pin descriptions and functions are the same as those for the Z8601/11 basic microcomputer (Chapter 6).

1.7 ROMLESS MICROCOMPUTER (Z8681/82)

The Z8681 and Z8682 ROMless microcomputers provide virtually all of the functions of the standard Z8 microcomputer without the need to mask-program on-chip ROM. This microcomputer is similar to the Z8601 version except that there is no on-chip program memory. Unlike the ROMless development and Protopack devices the Z8681/82 has no additional address or address control lines nor does it carry a plug-in piggyback memory module. Use of external memory rather than internal ROM enables this Z8 device to be used in low volume applications or where code flexibility is required. The use of Ports 0 and 1 to interface external memory leaves 16 to 24 lines for I/O.

Since Port 1 is dedicated as an 8-bit multiplexed Address/Data bus, and Port 0 lines can be programmed as address bits, the resulting 16-bit addresses can directly address up to 64K bytes of memory for the Z8681 and 62K bytes for the Z8682. (The Z8682 MCU cannot address the lower 2K bytes of memory).

The address capability of the Z8681/82 can be doubled by programming output $P3_4$ of Port 3 as Data Memory ($\overline{DM}$) select signal. The two states of this signal can be used with the 16-bit addresses to identify two separate external address spaces, thus increasing external address space to 128K bytes for the Z8681 and 124K bytes for the Z8682.

Pin functions and descriptions for the Z8681/82 microcomputer can be found in Chapter 7.

1.8 APPLICATIONS

Z8 microcomputers are most often used in high-performance, dedicated applications. Such specialized functions were previously accomplished with TTL logic, TTL logic plus a low-end MCU, or a microprocessor and peripherals. Some typical applications include:

- Disc drive controller
- Printer controller
- Terminals
- Modems
- Industrial controllers
- Key telephones
- Telephone switching systems
- Arcade games and intelligent home games
- Process control
- Intelligent instrumentation
- Automotive mechanisms

Following are brief descriptions for a few Z8 applications.

Printers. Input data (typically transmitted via a terminal or computer) can be sent to the Z8 on either a serial or parallel port. The Z8 then transfers the data into the external RAM buffer via another parallel port, where it can operate on the data before output to the printing mechanism.

Disk. Disk operations are read or write, with input received from either the disk or the computer. Data is transferred to the buffer memory a sector (128, 256, 512, 1024 bytes) at a time via the Z8, operated on as required, and subsequently output to the disk or computer.

Terminal. Input is received from either the keyboard or a computer. The Z8 device must maintain at least an input buffer and often the screen RAM.

Chapter 2
Architectural Overview

2.1 INTRODUCTION

The Z8 is a versatile single-chip microcomputer. Because its multiplexed address/data bus is merged with several I/O-oriented ports, the Z8 can function as either an I/O-intensive or a memory-intensive microcomputer. One key advantage to this organization is that external memory can be addressed while maintaining many of the I/O lines. Figure 2-1 shows the Z8 block diagram.

2.2 ADDRESS SPACES

To provide for both I/O-intensive and memory-intensive applications, the Z8 supports three basic address spaces:

- Program memory (internal and external)
- Data memory (external)
- Register file (internal)

A maximum of 64K bytes of program memory are directly addressable. In the Z8601 and Z8611 microcomputers, internal program memory consists of a mask-programmed ROM. The size of this internal ROM is 2K bytes for the Z8601 and 4K bytes for the Z8611. In one member of the Z8 family, the Z8681, all of the program memory is externally addressable.

Data memory space is always external to the Z8 microcomputer and is 62K bytes in size for the Z8601 and Z8682, and 60K and 64K bytes in size respectively for the Z8611 and Z8681.

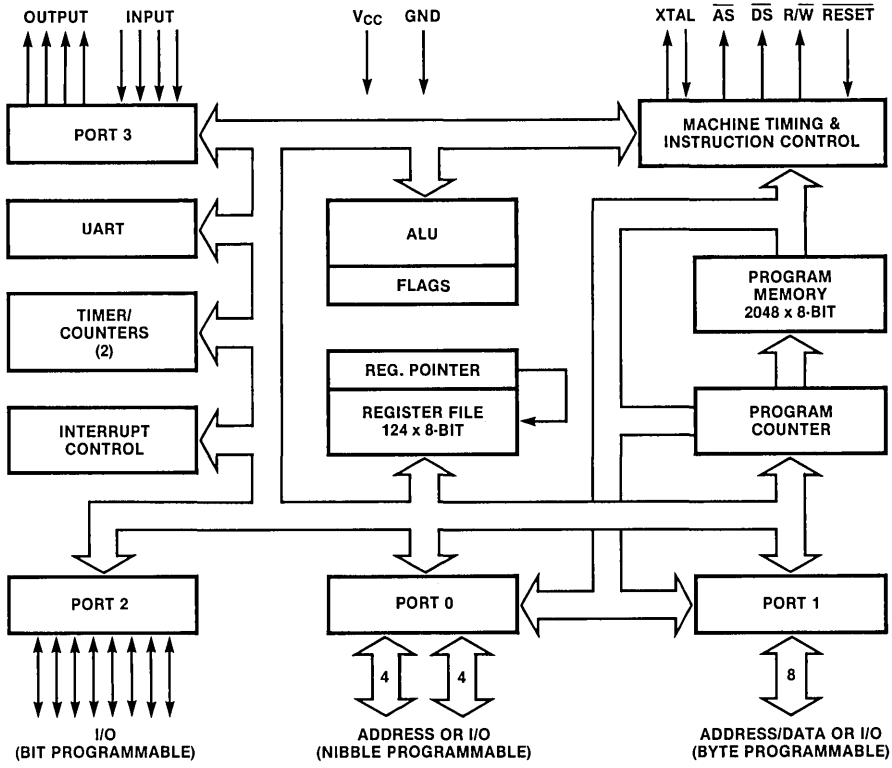


Figure 2-1. Z8 Block Diagram

2.3 REGISTER FILE

The Z8's register-oriented architecture centers around an internal register file composed of 124 general-purpose registers, 16 CPU and peripheral control registers, and 4 I/O port registers. All registers are eight bits. Any general-purpose register can be used as an accumulator, an address pointer, or an index, data, or stack register.

2.3.1 Register Pointer

A Register Pointer logically divides the register file into 9 working register groups of 16 registers each, which allows for fast context switching and shorter instruction formats.

2.3.2 Instruction Set

The Z8 CPU has an instruction set designed for the large register file. The instruction set provides a full complement of 8-bit arithmetic and logical operations. BCD operations are supported using a decimal adjustment of binary values, and 16-bit quantities for addresses and counters can be incremented and decremented. Bit manipulation and Rotate and Shift instructions complete the data manipulation capabilities of the Z8 system. No special I/O instructions are necessary since the I/O is mapped into the register file.

2.3.3 Data Types

The Z8 CPU supports operations on bits, BCD digits, bytes, and 2-byte words.

Bits in the register file can be tested, set, cleared, and complemented. Bits within a byte are numbered from 0 to 7 with bit 0 being the least significant (right-most) bit (Figure 2-2).

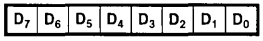


Figure 2-2. Bits in Register

Manipulation of BCD digits packed two-to-a-byte is accomplished by a Decimal Adjust instruction and a Swap instruction. Decimal Adjust is used after a binary addition or subtraction on BCD digits.

Logical, Shift, Rotate and Load instructions operate on bytes in the register file. Bytes in data memory are only affected by Load instructions.

Sixteen-bit arithmetic instructions (Increment Word and Decrement Word) operate on words in the register file.

2.3.4 Addressing Modes

The addressing modes of the Z8 CPU are:

- Register
- Indirect Register
- Immediate
- Direct Address
- Indexed (with a short 8-bit displacement)
- Program Counter Relative

Register, Indirect Register, and Immediate addressing modes are available for Load, Arithmetic, Logical, Shift, Rotate, and Stack instructions. Conditional Jumps use both Direct Address and Program Counter Relative, while Jump and Call instructions use Direct Address and Indirect Register addressing modes.

2.4 I/O OPERATIONS

The Z8 has 32 pins dedicated to input and output. These lines are grouped into four ports of eight lines each. Ports can be programmed as input, output, or bidirectional. Under software control, the ports provide timing, status signals, address outputs, and serial or parallel I/O with or without handshake. Multiprocessor system configurations are also supported.

2.4.1 Timers

To unburden the program from real-time problems such as serial data communications and counting/timing, the Z8 contains an on-chip universal asynchronous receiver/transmitter (UART) and two counter/timers with a large number of user-selectable modes. One on-chip timer provides the bit rate input to the UART during communications.

2.4.2 Interrupts

I/O operations can be interrupt-driven or polled. The Z8 supports six vectored interrupts that can be masked and prioritized.

2.5 OSCILLATOR

The Z8 offers an on-chip oscillator and an optional power-down mechanism that can be used to maintain the contents of the register file with a low-power battery.

2.6 PROTOPACK

The Z8 Protopack allows the user to prototype system hardware and develop software that is eventually to be mask-programmed into the on-chip ROM of the 2K byte (Z8601) or the 4K byte (Z8611) version of the Z8.

Chapter 3
Address Spaces

3.1 INTRODUCTION

Three address spaces are available in the Z8 microcomputer:

- The CPU Register File contains addresses for all general-purpose, peripheral, control, and I/O port registers.
- The CPU Program Memory contains addresses for all memory locations having executable code and/or data.
- The CPU Data Memory contains addresses for all memory locations that hold data only.

These address spaces are described in detail in the following sections.

3.2 CPU REGISTER FILE

The register file totals 256 consecutive bytes, of which 144 have been implemented. (Unused register space is reserved for future expansion.) The register file consists of 4 I/O ports (R0-R3), 124 general-purpose registers (R4-R127), 9 peripheral registers (R240-R248), and 7 control registers (R249-R255). Figure 3-1 shows the layout of the register file, including register names, locations, and identifiers.

Registers can be accessed as either 8- or 16-bit registers using Direct, Indirect, or Indexed addressing. All 144 registers can be referenced or modified by any instruction that accesses an 8-bit register, without the need for special instructions. Registers accessed as 16-bits are treated as even-odd register pairs (there are 72 valid pairs). In this case, the data's MSB is stored in the even-numbered register, while the LSB goes into the next higher odd-numbered register (Figure 3-2).

DEC		HEX	IDENTIFIERS
255	STACK POINTER (BITS 7-0)	FF	SPL
254	STACK POINTER (BITS 15-8)	FE	SPH
253	REGISTER POINTER	FD	RP
252	PROGRAM CONTROL FLAGS	FC	FLAGS
251	INTERRUPT MASK REGISTER	FB	IMR
250	INTERRUPT REQUEST REGISTER	FA	IRQ
249	INTERRUPT PRIORITY REGISTER	F9	IPR
248	PORTS 0-1 MODE	F8	P01M
247	PORT 3 MODE	F7	P3M
246	PORT 2 MODE	F6	P2M
245	T0 PRESCALER	F5	PRE0
244	TIMER/COUNTER 0	F4	T0
243	T1 PRESCALER	F3	PRE1
242	TIMER/COUNTER 1	F2	T1
241	TIMER MODE	F1	TMR
240	SERIAL I/O	F0	SIO
	NOT IMPLEMENTED		
127	GENERAL-PURPOSE REGISTERS	7F	
4		04	
3	PORT 3	03	P3
2	PORT 2	02	P2
1	PORT 1	01	P1
0	PORT 0	00	P0

Figure 3-1. Register File

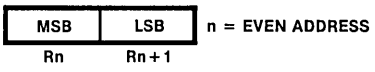


Figure 3-2. 16-Bit Register Addressing

By using logical instructions and a mask, individual bits within registers can be accessed for bit set, bit clear, bit complement, or bit test operations. For example, the instruction AND R, MASK performs a bit clear operation.

When instructions are executed, registers are read when defined as sources and written when defined as destinations. All general-purpose registers function as accumulators, address pointers, index registers, stack areas, or scratchpad memory.

Z8 instructions can access 8-bit registers and register pairs (16-bit) using either 4-bit or 8-bit address fields. With 4-bit addressing, the register file is logically divided into 9 groups of 16 working registers as shown in Figure 3-3. A Register Pointer (one of the control registers) contains the base address of the active working register group.

When accessing one of the working registers, the 4-bit address is concatenated with the upper four bits of the Register Pointer, thus forming an 8-bit address. Figure 3-4 illustrates this operation. Since working registers are typically specified by short format instructions, there are fewer bytes of code needed, which reduces execution time. In addition, when processing interrupts or changing tasks, the Register Pointer speeds context switching. A special Set Register Pointer (SRP) instruction sets the contents of the Register Pointer.

3.2.1 Error Conditions

Registers must be correctly used because certain conditions produce inconsistent results and should be avoided:

- Registers R243 and R245-R249 are write-only registers. If an attempt is made to read these registers, %FF is returned (% is a prefix that indicates hexadecimal notation).
- When register R253 (Register Pointer) is read, all 0s are returned in the least significant four bits.

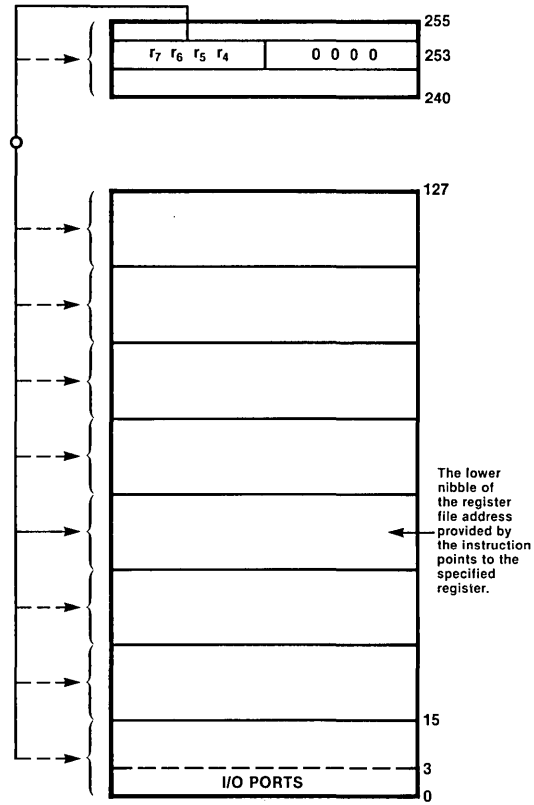


Figure 3-3. Working Register Groups

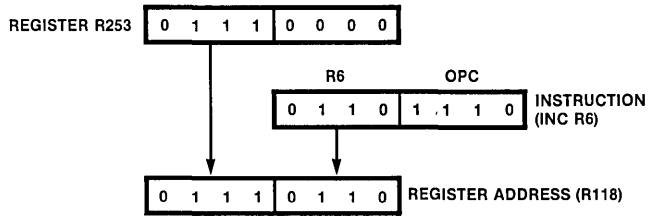


Figure 3-4. Working Register Addressing

- When registers R0 and R1 (Ports 0 and 1) are defined as address outputs, they will return 1s in each address bit location when read.

- Writing to bits which are defined as address output, timer output, serial output, or handshake output will have no effect.

- Instruction DJNZ uses a general register as a counter. Only registers R4-R127 can be used with this instruction.

3.3 CPU CONTROL AND PERIPHERAL REGISTERS

The Z8 control registers govern the operation of the CPU. Any instruction that references the register file can access these control registers. Available control registers are:

- Interrupt Priority register (IPR)
- Interrupt Mask register (IMR)
- Interrupt Request register (IRQ)
- Program Control flags (FLAGS)
- Register Pointer (RP)
- Stack Pointer - high-byte (SPH)
- Stack Pointer - low-byte (SPL)

The Z8 uses a 16-bit Program Counter (PC) to determine the sequence of current program instructions. The PC is not an addressable register.

Peripheral registers are used to transfer data, configure the operating mode, and control the operation of the on-chip peripherals. Any instruction that references the register file can access peripheral registers. The peripheral registers are:

- Serial I/O (SIO)
- Timer Mode (TMR)
- Timer/Counter 0 (T0)
- T0 Prescaler (PRE0)
- Timer/Counter 1 (T1)
- T1 Prescaler (PRE1)
- Port 0-1 Mode (P01M)
- Port 2 Mode (P2M)
- Port 3 Mode (P3M)

In addition, the four port registers (P0-P3) are considered to be peripheral registers.

The functions and applications of control and peripheral registers are described in subsequent sections of this manual.

3.4 CPU PROGRAM MEMORY

The Z8 can access 64K bytes of program memory with the 16-bit Program Counter. In the Z8601, the lower 2K bytes of the program memory address space are internal ROM, while in the Z8611 the lower 4K bytes are internal ROM. In the Z8682 the lower 2K bytes are not accessible.

To access program memory outside the on-board ROM space, Port 0 and Port 1 can be configured as a memory interface. For example, Port 1 as a multiplexed Address/Data port (AD_0-AD_7) provides Address lines A_0-A_7 and Data lines D_0-D_7. Port 0 can be configured for an additional four or eight address lines (A_8-A_{11} or A_8-A_{15}). This memory interface is supported by the control lines $\overline{AS}$ (Address Strobe), $\overline{DS}$ (Data Strobe) and R/$\overline{W}$ (Read/Write).

In the ROMless Z8681 version, Port 1 is automatically a multiplexed Address/Data port. Port 0 must be configured for additional address lines as needed.

The first 12 bytes of program memory are reserved for the interrupt vectors. Addresses 0-11 contain six 16-bit vectors that correspond to the six available interrupts. Figure 3-5 illustrates the order of 16-bit data stored in program memory.

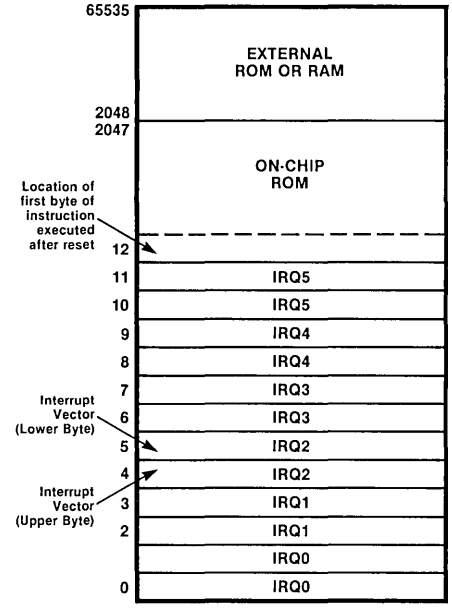


Figure 3-5a. Z8601 Program Memory Map

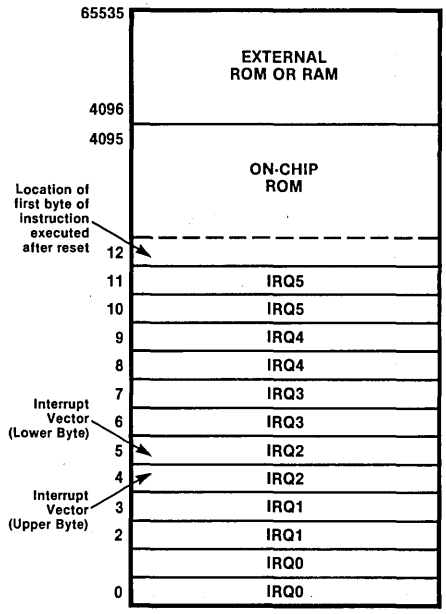


Figure 3-5b. Z8611 Program Memory Map

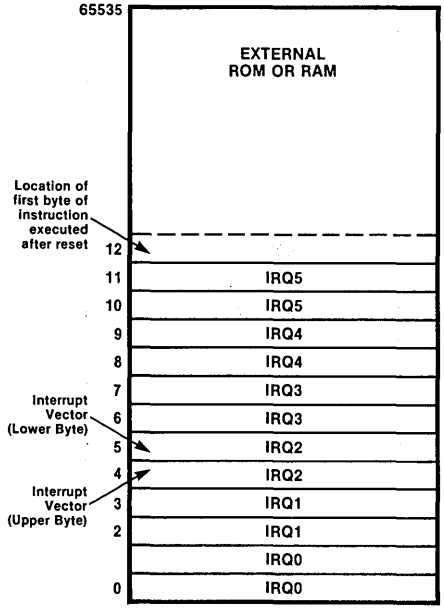


Figure 3-5c. Z8681 Program Memory Map

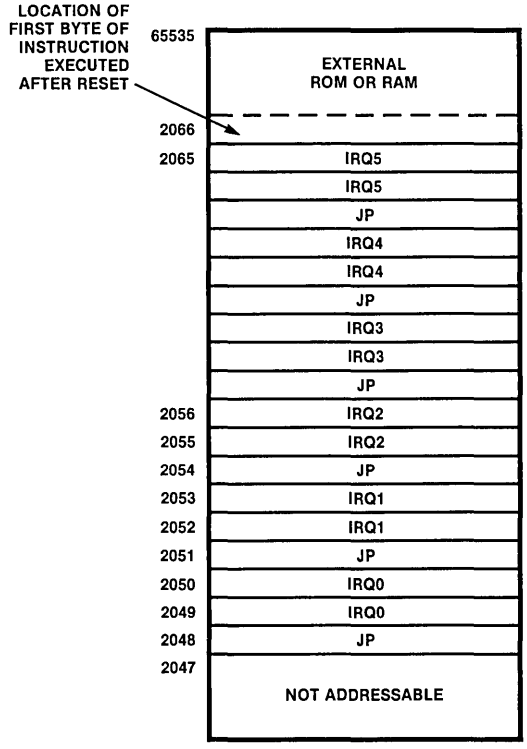


Figure 3-5d. Z8682 Program Memory Map

When an interrupt occurs, the address stored in the interrupt's vector location points to a service routine. This routine assumes program control.

The first 2K bytes of program memory are not addressable in the Z8682 ROMless version. Beginning at address 2048 the first 18 bytes contain interrupt vectors which are Jump Direct instructions. When an interrupt occurs, the Z8682 executes the corresponding Jump to interrupt.

The first address available for a user program is location 12. This address is loaded into the Program Counter after a hardware reset.

The first address available for a user program in the Z8682 is location 2066 (Hexadecimal %812). This address is loaded into the Program Counter after a hardware reset.

3.5 CPU DATA MEMORY

Up to 64K bytes of external data memory can be accessed in the Z8 microcomputer. As shown in Figure 3-6, the origin, and hence, the actual size of data memory is device-dependent. The origin of data memory is the same as the starting address of external program memory.

Like external program memory, external data memory Address/Data lines are provided by Port 1 for 8-bit addresses, and by Ports 0 and 1 for 12-bit and 16-bit addresses.

External data memory can be included with or separated from the external program memory addressing space. When data memory is separated from program memory, the Data Memory output ($\overline{DM}$) is used to select between data and program memories.

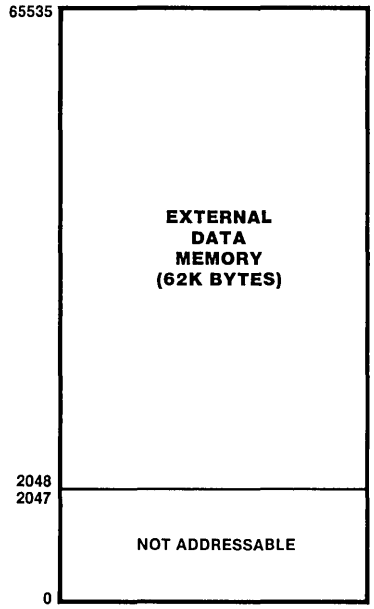


Figure 3-6a. Z8601 or Z8682 Data Memory Map

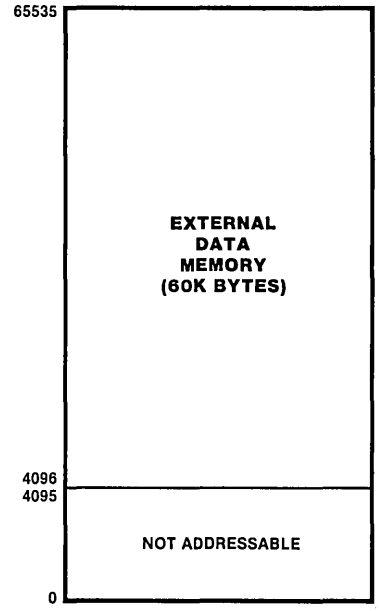


Figure 3-6b. Z8611 Data Memory Map

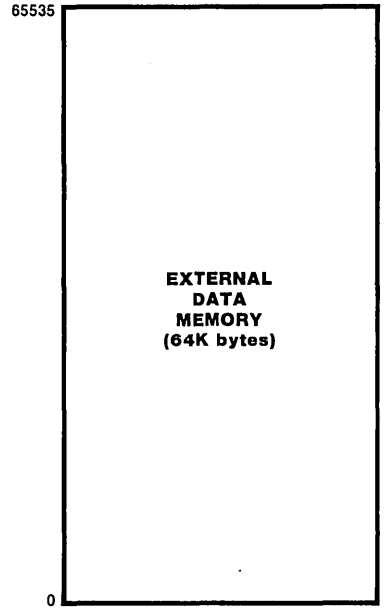


Figure 3-6c. Z8681 Data Memory Map

3.6 CPU STACKS

Stack operations can occur in either the register file or data memory. Under software control, Port 0 and 1 Mode register (R258) selects stack location.

The register pair R254 and R255 forms the 16-bit Stack Pointer (SP) which is used for all stack operations. The stack address is stored with the MSB in R254 and LSB in R255 (Figure 3-7).

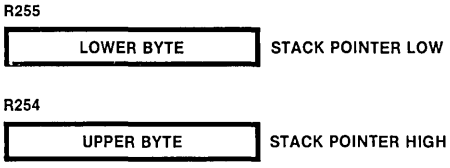


Figure 3-7. Stack Pointer

The stack address is decremented prior to a Push operation and incremented after a Pop operation. The stack address always points to the data stored on the top-of-stack. The Z8 stack is a return stack for Call instructions and interrupts as well as a data stack. During a Call instruction, the contents of the PC are saved on the stack. The PC is restored during a Return instruction. Interrupts cause the contents of the PC and Flag register to be saved on the stack. The IRET instruction restores them (Figure 3-8).

When the Z8 is configured for an internal stack (i.e., using the register file), register R255 serves as the Stack Pointer. The value in R254 is ignored and can be used as a general-purpose register. However, an overflow or underflow can occur when stack address is incremented or decremented during normal stack operations.

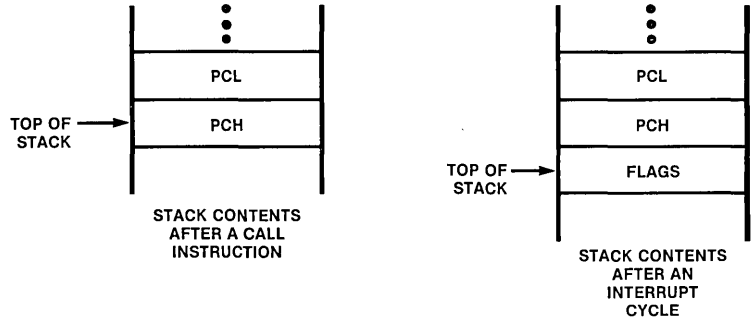


Figure 3-8. Stack Operations

Chapter 4
Address Modes

4.1 INTRODUCTION

The Z8 microcomputer provides six addressing modes:

- Register (R)
- Indirect Register (IR)
- Indexed (X)
- Direct (D)
- Relative (RA)
- Immediate (IM)

With the exception of immediate data and condition codes, all operands are expressed as register file, program memory, or data memory addresses. Registers are accessed using 8-bit addresses in the range 0-127 and 240-255.

Working registers are accessed using 4-bit addresses in the range 0-15. The address of the register being accessed is formed by the concatenation of the upper four bits in the Register Pointer (R253) with the 4-bit working register address supplied by the instruction.

Registers can be used in pairs to designate 16-bit values or memory addresses. A register pair must be specified as an even-numbered address in the range 0, 2,...., 14.

Addressing modes are instruction-specific. Section 5.4 discusses each addressing mode as it corresponds to particular instructions.

In the following definitions, the use of "register" also implies register pair, working register, or working register pair.

4.2 REGISTER ADDRESSING (R)

In the Register addressing mode, the operand value is the contents of the specified register or register pair (Figures 4-1 and 4-2).

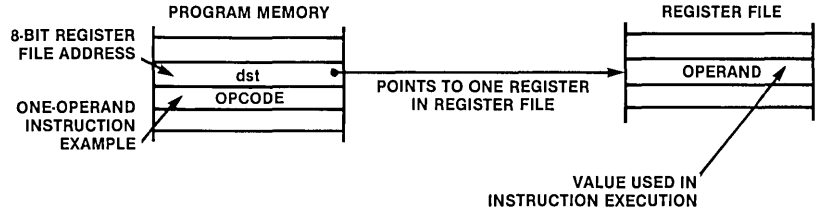


Figure 4-1. Register Addressing

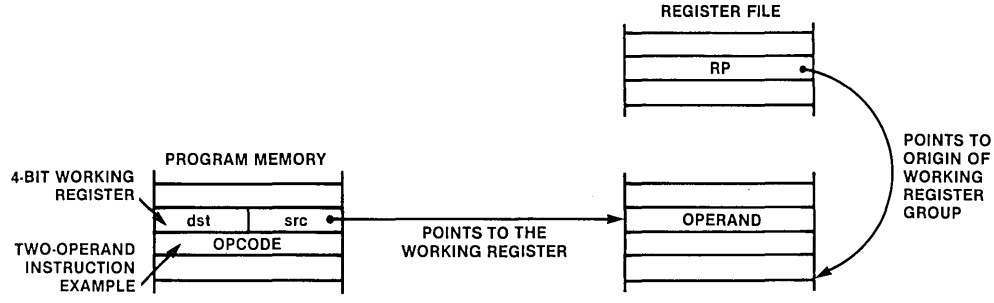


Figure 4-2. Working-Register Addressing

4.3 INDIRECT REGISTER ADDRESSING (IR)

In the Indirect Register addressing mode, the contents of the specified register is the address of the operand (Figures 4-3 and 4-4).

Depending upon the instruction selected, the address points to a register, program memory, or an external data memory location.

When accessing program memory or external data memory, register pairs or working register pairs are used to hold the 16-bit addresses.

4.4 INDEXED ADDRESSING (X)

The Indexed addressing mode is used only by the Load (LD) instruction. An indexed address consists of a register address offset by the contents of a designated working register (the Index). This offset is added to the register address to obtain the address of the operand. Figure 4-5 illustrates this addressing convention.

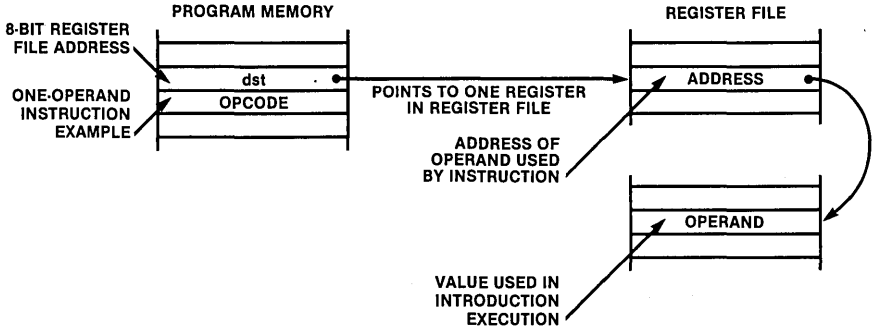


Figure 4-3. Indirect Register Addressing to Register File

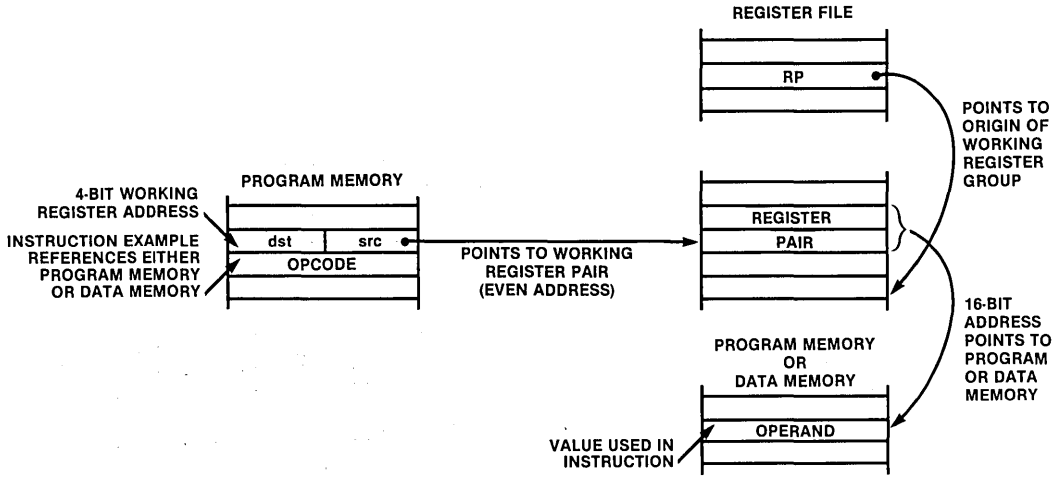


Figure 4-4. Indirect Register Addressing to Program or Data Memory

4.5 DIRECT ADDRESSING (DA)

The Direct addressing mode, as shown in Figure 4-6, specifies the address of the next instruction to be executed. Only the Conditional Jump (JP) and Call (CALL) instructions use this addressing mode.

4.6 RELATIVE ADDRESSING (RA)

In the Relative addressing mode, illustrated in Figure 4-7, the instruction specifies a two's-complement signed displacement in the range of -128 to +127. This is added to the contents of the PC to obtain the address of the next instruction to be executed. The PC (prior to the add) consists of the address of the instruction following the Jump Relative (JR) or Decrement and Jump if Nonzero (DJNZ) instruction. JR and DJNZ are the only instructions that use this addressing mode.

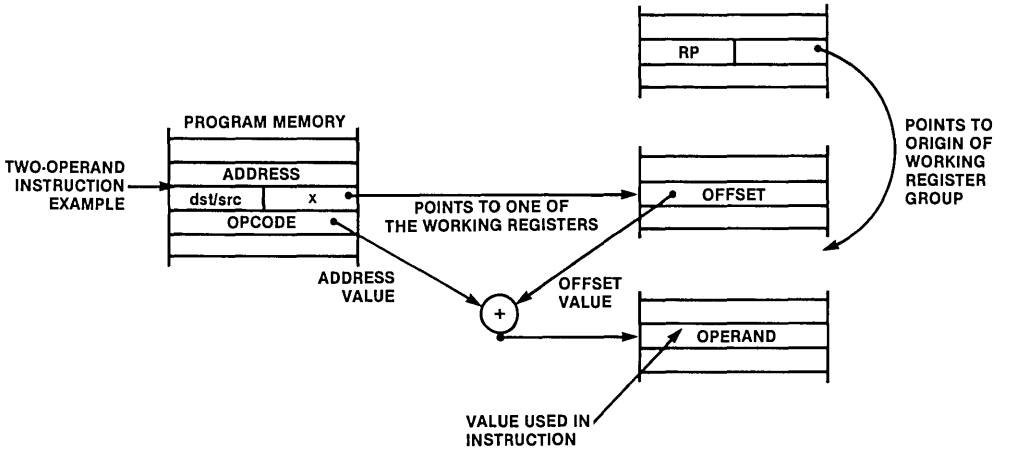


Figure 4-5. Indexed Addressing

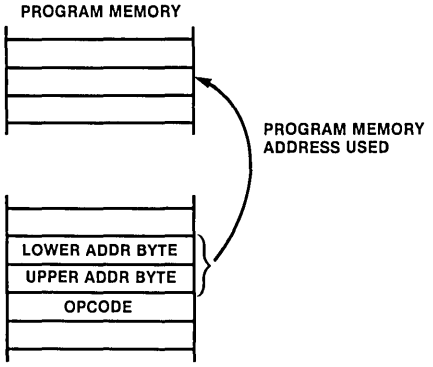


Figure 4-6. Direct Addressing

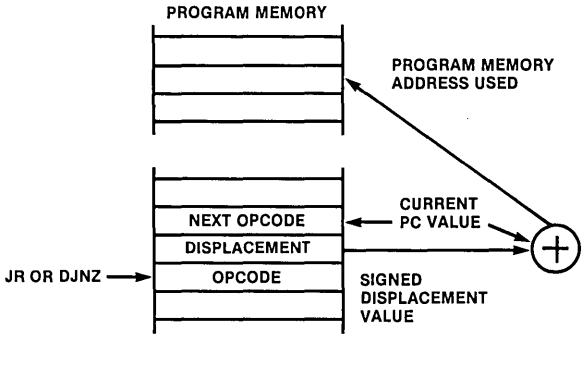


Figure 4-7. Relative Addressing

4.7 IMMEDIATE DATA ADDRESSING (IM)

Immediate data is considered an "addressing mode" for the purposes of this discussion. It is the only addressing mode that does not indicate a register or memory address as the source operand; the operand value used by the instruction is the value supplied in the operand field itself. Because an immediate operand is part of the instruction, it is always located in the program memory address space.

Figure 4-8. Immediate Data Addressing

Chapter 5
Instruction Set

5.1 FUNCTIONAL SUMMARY

Z8 instructions can be divided functionally into the following eight groups:

- Load
- Arithmetic
- Logical
- Program Control
- Bit Manipulation
- Block Transfer
- Rotate and Shift
- CPU Control

The following summary shows the instructions belonging to each group and the number of operands required for each. The source operand is "src", "dst" is the destination operand, and "cc" is a condition code.

Load Instructions

Mnemonic	Operands	Instruction
CLR	dst	Clear
LD	dst,src	Load
LDC	dst,src	Load Constant
LDE	dst,src	Load External
POP	dst	Pop
PUSH	src	Push

Arithmetic Instructions

Mnemonic	Operands	Instruction
ADC	dst,src	Add With Carry
ADD	dst,src	Add
CP	dst,src	Compare
DA	dst	Decimal Adjust
DEC	dst	Decrement
DECW	dst	Decrement Word
INC	dst	Increment
INCW	dst	Increment Word
SBC	dst,src	Subtract With Carry
SUB	dst,src	Subtract

Logical Instructions

Mnemonic	Operands	Instruction
AND	dst,src	Logical And
COM	dst	Complement
OR	dst,src	Logical Or
XOR	dst,src	Logical Exclusive Or

Program-Control Instructions

Mnemonic	Operands	Instruction
CALL	dst	Call Procedure
DJNZ	r,dst	Decrement and Jump NonO
IRET		Interrupt Return
JP	cc,dst	Jump
JR	cc,dst	Jump Relative
RET		Return

Bit-Manipulation Instructions

Mnemonic	Operands	Instruction
TCM	dst,src	Test Complement Under Mask
TM	dst,src	Test Under Mask
AND	dst,src	Bit Clear
OR	dst,src	Bit Set
XOR	dst,src	Bit Complement

Block-Transfer Instructions

Mnemonic	Operands	Instruction
LDCI	dst,src	Load Constant Auto-increment
LDEI	dst,src	Load External Auto-increment

Rotate and Shift Instructions

Mnemonic	Operands	Instruction
RL	dst	Rotate Left
RLC	dst	Rotate Left Through Carry
RR	dst	Rotate Right
RRC	dst	Rotate Right Through Carry
SRA	dst	Shift Right Arithmetic
SWAP	dst	Swap Nibbles

CPU Control Instructions

Mnemonic	Operand	Instruction
CCF		Complement Carry Flag
DI		Disable Interrupts
EI		Enable Interrupts
NOP		No Operation
RCF		Reset Carry Flag
SCF		Set Carry Flag
SRP	src	Set Register Pointer

5.2 PROCESSOR FLAGS

The Flag register (R252) informs the user about the current status of the Z8. The flags and their bit positions in the Flag register are shown in Figure 5-1.

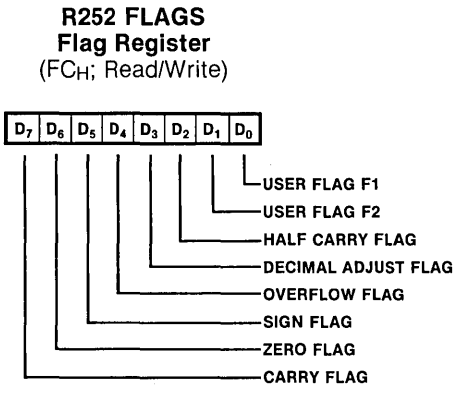


Figure 5-1. Flag Register

The Z8 Flag register contains six bits of status information which are set or cleared by CPU operations. Four of the bits (C, V, Z and S) can be tested for use with conditional Jump instructions. Two flags (H, D) cannot be tested and are used for BCD arithmetic.

The two remaining bits in the Flag register (F1, F2) are available to the user, but they must be set or cleared by instruction and are not usable with conditional Jumps.

As with bits in the other control registers, Flag register bits can be set or reset by instructions; however, only those instructions that do not affect the flags as an outcome of the execution should be used (e.g., Load Immediate).

5.2.1 Carry Flag (C)

The Carry flag is set to 1 whenever the result of an arithmetic operation generates a carry out of or a borrow into the high order bit 7; otherwise, the Carry flag is cleared to 0.

Following Rotate and Shift instructions, the Carry flag contains the last value shifted out of the specified register.

An instruction can set, reset, or complement the Carry flag.

RETI changes the value of the Carry flag when the saved Flag register is restored.

5.2.2 Zero Flag (Z)

For arithmetic and logical operations, the Zero flag is set to 1 if the result is zero; otherwise, the Zero flag is cleared.

If the result of testing bits in a register is 0, the Zero flag is set to 1; otherwise the flag is cleared.

If the result of a Rotate or Shift operation is 0, the Zero flag is set to 1; otherwise, the flag is cleared.

RETI changes the value of the Zero flag when the saved Flag register is restored.

5.2.3 Sign Flag (S)

The Sign flag stores the value of the most significant bit of a result following arithmetic, logical, Rotate, or Shift operations.

When performing arithmetic operations on signed numbers, binary two's complement notation is used to represent and process information. A positive number is identified by a 0 in the most significant bit position, and therefore, the Sign flag is also 0.

A negative number is identified by a 1 in the most significant bit position, and therefore, the Sign flag is also 1.

RETI changes the value of the Zero flag when the saved Flag register is restored.

5.2.4 Overflow Flag (V)

For signed arithmetic, Rotate, and Shift operations, the Overflow flag is set to 1 when the result is greater than the maximum possible number (> 127) or less than the minimum possible number (< -128) that can be represented in two's complement form. The flag is set to 0 if no overflow occurs.

Following logical operations, the Overflow flag is set to 0.

RETI changes the value of the Overflow flag when the saved Flag register is restored.

5.2.5 Decimal-Adjust Flag (D)

The Decimal-adjust flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag specifies what type of instruction was last executed so that the subsequent Decimal Adjust (DA) operation can function properly. Normally, the Decimal-adjust flag cannot be used as a test condition.

After a subtraction, the Decimal-adjust flag is set to 1; following an addition it is cleared to 0.

RETI changes the value of the Decimal-adjust flag when the saved Flag register is restored.

5.2.6 Half-Carry Flag (H)

The Half-carry flag is set to 1 whenever an addition generates a carry out of bit 3 (Overflow), or a subtraction generates a borrow into bit 3. The Half-carry flag is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. As in the case of the Decimal-adjust flag, the user does not normally access this flag.

RETI changes the value of the Half-carry flag when the saved Flag register is restored.

5.3 CONDITION CODES

Flags C, Z, S, and V control the operation of the "conditional" Jump instructions. Sixteen frequently useful functions of the flag settings are encoded in a 4-bit field called the condition code (CC), which forms bits 4-7 of the conditional instructions.

Section 5.4.2 lists the condition codes and the flag settings they represent.

5.4 NOTATION AND BINARY ENCODING

In the detailed instruction descriptions that make up the rest of this chapter, operands and status flags are represented by a notational shorthand. Operands (condition codes and address modes) and their notations are as follows:

Notation	Address Mode	Actual Operand/Range
cc	Condition Code	See condition code list below
r	Working register only	Rn: where n = 0-15
R	Register or working register	reg: where reg represents a number in the range 0-127, 240-255 Rn: where n = 0-15
RR	Register pair or working register pair	reg: where reg represents an even number in the range 0-126, 240-254 RRp: where p = 0, 2,...,14
Ir	Indirect working register only	@ Rn: where n = 0-15
IR	Indirect register or working register	@ reg: where reg represents a number in the range 0-127, 240-255 @ Rn: where n = 0-15
Irr	Indirect working register pair only	@ RRp: where p = 0, 2,...,14
IRR	Indirect register pair or working register pair	@ reg: where reg resents an even number in the range 0-126, 240-254 @ RRp: where p = 0, 2,...,14

Notation	Address Mode	Actual Operand/Range
X	Indexed	reg (Rn): where reg represent a number in the range 0-127, 240-255 and n = 0-15
DA	Direct Address	addrs: where addrs represents a number in the range 0-65,535
RA	Relative Address	addrs: where addrs represents a number in the range +127, -128 which is an offset relative to the address of the next instruction
IM	Immediate	#data: where data is a number between 0 and 255

Additional symbols used are:

Symbol	Meaning
dst	Destination operand
src	Source operand
@	Indirect address prefix
SP	Stack Pointer
PC	Program Counter
FLAGS	Flag register (R252)
RP	Register Pointer (R253)
IMR	Interrupt mask register (251)
#	Immediate operand prefix
%	Hexadecimal number prefix
OPC	Opcode

Assignment of a value is indicated by the symbol "<-". For example,

```
dst <- dst + src
```

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

```
dst (7)
```

refers to bit 7 of the destination operand.

5.4.1 Assembly Language Syntax

For proper instruction execution, Z8 PLZ/ASM assembly language syntax requires that "dst, src" be specified, in that order. The following instruction descriptions show the format of the object code produced by the assembler. This binary format should be followed by users who prefer manual program coding or who intend to implement their own assembler.

Example: If the contents of registers %43 and %08 are added and the result stored in %43, the assembly syntax and resulting object code are:

```
ASM:  ADD   %43, %08   (ADD dst, src)
OBJ:  04    08   43    (OPC src, dst)
```

In general, whenever an instruction format requires an 8-bit register address, that address can specify any register location in the range 0-127, 240-255 or a working register R0-R15. If, in the above example, register %08 is a working register, the assembly syntax and resulting object code would be:

```
ASM:  ADD   %43, R8    (ADD dst src)
OBJ:  04    E8   43    (OPC src dst)
```

For a more complete description of assembler syntax refer to the Z8 PLZ/ASM Assembly Language Manual (publication no. 03-3023-03) and ZSCAN 8 User's Tutorial (publication no. 03-8200-01).

5.4.2 Condition Codes and Flag Settings

The condition codes and flag settings are summarized in the following tables. Notation for the flags and how they are affected are as follows:

C	Carry flag	0	Cleared to 0
Z	Zero flag	1	Set to 1
S	Sign flag	*	Set or cleared according to operation
V	Overflow flag		
D	Decimal-adjust flag	-	Unaffected
H	Half-carry flag	X	Undefined

Condition Codes

Binary	Mnemonic	Meaning	Flags Settings
0000	F	Always false	-
1000	(blank)	Always true	-
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not 0	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater Than	(Z OR (S XOR V))=0
0010	LE	Less than or equal	(Z OR (S XOR V))=1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C=0 AND Z=0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1

5.5 INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	Flags Affected C Z S V D H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	* * * * 0 *
ADD dst,src dst ← dst + src	(Note 1)		0□	* * * * 0 *
AND dst,src dst ← dst AND src	(Note 1)		5□	- * * 0 - -
CALL dst SP ← SP - 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	- - - - - -
CCF C ← NOT C			EF	* - - - - -
CLR dst dst ← 0	R IR		B0 B1	- - - - - -
COM dst dst ← NOT dst	R IR		60 61	- * * 0 - -
CP dst,src dst - src	(Note 1)		A□	* * * * - -
DA dst dst ← DA dst	R IR		40 41	* * * X - -
DEC dst dst ← dst - 1	R IR		00 01	- * * * - -
DECW dst dst ← dst - 1	RR IR		80 81	- * * * - -
DI IMR (7) ← 0			8F	- - - - - -
DJNZ r,dst r ← r - 1 if r ≠ 0 PC ← PC + dst Range: +127, -128	RA		rA r=0-F	- - - - - -
EI IMR (7) ← 1			9F	- - - - - -
INC dst dst ← dst + 1	r R IR		rE r=0-F 20 21	- * * * - -
INCW dst dst ← dst + 1	RR IR		A0 A1	- * * * - -
IRET FLAGS ← @SP; SP ← SP + 1 PC ← @SP; SP ← SP + 2; IMR (7) ← 1			BF	* * * * * *
JP cc,dst if cc is true PC ← dst	DA IRR		cD c=0-F 30	- - - - - -
JR cc,dst if cc is true, PC ← PC + dst Range: +127, -128	RA		cB c=0-F	- - - - - -
LD dst,src dst ← src	r r R r X r Ir R R R IR IR	IM R r X r Ir r R IR IM IM R	rC r8 r9 r=0-F C7 D7 E3 F3 E4 E5 E6 E7 F5	- - - - - -
LDC dst,src dst ← src	r Irr	Irr r	C2 D2	- - - - - -
LDCI dst,src dst ← src r ← r + 1; rr ← rr + 1	Ir Irr	Irr Ir	C3 D3	- - - - - -
LDE dst,src dst ← src	r Irr	Irr r	82 92	- - - - - -
LDEI dst,src dst ← src r ← r + 1; rr ← rr + 1	Ir Irr	Irr Ir	83 93	- - - - - -
NOP			FF	- - - - - -
OR dst,src dst ← dst OR src	(Note 1)		4□	- * * 0 - -
POP dst dst ← @SP SP ← SP + 1	R IR		50 51	- - - - - -
PUSH src SP ← SP - 1; @SP ← src		R IR	70 71	- - - - - -
RCF C ← 0			CF	0 - - - - -
RET PC ← @SP; SP ← SP + 2			AF	- - - - - -
RL dst	R IR		90 91	* * * * - -
RLC dst	R IR		10 11	* * * * - -
RR dst	R IR		E0 E1	* * * * - -
RRC dst	R IR		C0 C1	* * * * - -
SBC dst,src dst ← dst - src - C	(Note 1)		3□	* * * * 1 *
SCF C ← 1			DF	1 - - - - -
SRA dst	R IR		D0 D1	* * * 0 - -
SRP src RP ← src		Im	31	- - - - - -
SUB dst,src dst ← dst - src	(Note 1)		2□	* * * * 1 *
SWAP dst	R IR		F0 F1	X * * X - -
TCM dst,src (NOT dst) AND src	(Note 1)		6□	- * * 0 - -
TM dst,src dst AND src	(Note 1)		7□	- * * 0 - -
XOR dst,src dst ← dst XOR src	(Note 1)		B□	- * * 0 - -

Note 1

These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

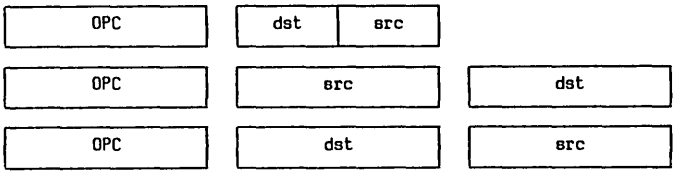
For example, to determine the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode dst	Addr Mode src	Lower Opcode Nibble
r	r	2
r	Ir	3
R	R	4
R	IR	5
R	IM	6
IR	IM	7

ADC
Add With Carry

ADC dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	src
6	12	r	r
	13	r	Ir
10	14	R	R
	15	R	IR
10	16	R	IM
	17	IR	IM

Operation: dst <-- dst + src + c

The source operand, along with the setting of the C flag, is added to the destination operand and the sum is stored in the destination. The contents of the source are not affected. Two's complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from the addition of low-order operands to be carried into the addition of high-order operands.

Flags:

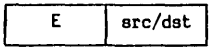
C: Set if there is a carry from the most-significant bit of the result; cleared otherwise
Z: Set if the result is zero; cleared otherwise
S: Set if the result is negative; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
D: Always cleared
H: Set if there is a carry from the most-significant bit of the low-order four bits of the result; cleared otherwise

Example: If the register named SUM contains %16, the C flag is set to 1, working register 10 contains %20 (32 decimal), and register 32 contains %10, the statement

```
ADC SUM,@R10
```

leaves the value %27 in Register SUM. The C, Z, S, V, D, and H flags are all cleared.

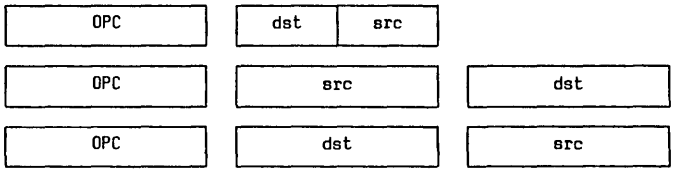
Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



ADD
Add

ADD dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	src
6	02	r	r
	03	r	Ir
10	04	R	R
	05	R	IR
10	06	R	IM
	07	IR	IM

Operation: dst <-- dst + src

The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are not affected. Two's complement addition is performed.

Flags:

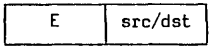
C: Set if there was a carry from the most-significant bit of the result; cleared otherwise
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Set if a carry from the low-order nibble occurs
D: Always reset to 0

Example: If the register named SUM contains %44 and the register named AUGEND contains %11, the statement

```
ADD SUM,AUGEND
```

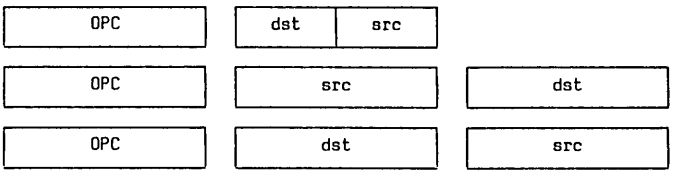
leaves the value %55 in register SUM and leaves all flags cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



AND dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	Address Mode src
6	52	r	r
	53	r	IR
10	54	R	R
	55	R	IR
10	56	R	IM
	57	IR	IM

Operation: dst <-- dst AND src

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation results in a 1 bit being stored whenever the corresponding bits in the two operands are both 1s; otherwise a 0 bit is stored. The contents of the source bit are not affected.

Flags:

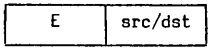
C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the source operand is the immediate value %7B (01111011) and the register named TARGET contains %C3 (11000011), the statement

```
AND TARGET, #%7B
```

leaves the value %43 (01000011) in register TARGET. The Z, V, and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



CALL
Call Procedure

CALL dst

Instruction Format:

		Cycles	OPC (Hex)	Address Mode dst
OPC	dst	20	D6	DA
OPC	dst	20	D4	IRR

Operation:

SP <-- SP - 2
@SP <-- PC
PC <-- dst

The current contents of the PC are pushed onto the top of the stack. The PC value is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the PC and points to the first instruction of a procedure.

At the end of the procedure a RETurn instruction can be used to return to the original program flow. RET pops the top of the stack back into the PC.

Flags:

No flags affected.

Example:

If the contents of the PC are %1A47 and the contents of the SP (control registers 254-5) are %3002, the statement

```
CALL %3521
```

causes the SP to be decremented to %3000, %1A4A (the address following the instruction) is stored in external data memory %3000-%3001, and the PC is loaded with %3521. The PC now points to the address of the first statement in the procedure to be executed.

Note:

When used to specify a 4-bit working-register pair address, address mode IRR uses the format:

E	dst

CCF
Complement Carry Flag

CCF

Instruction Format:

OPC

Cycles	OPC (Hex)
6	EF

Operation: C <-- NOT C

The C flag is complemented; if C = 1, it is changed to C = 0, and vice-versa.

Flags: **C:** Complemented
No other flags affected

Example: If the C flag contains a 0, the statement

```
CCF
```

will change the 0 to 1.

CLR
Clear

CLR dst

Instruction Format:

Cycles	OPC (Hex)	Address Mode dst
6	B0	R
	B1	IR

Operation: dst <-- 0

The destination location is cleared to 0.

Flags: No flags affected.

Example: If working register 6 contains %AF, the statement

```
CLR R6
```

will leave the value 0 in that register

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

COM dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	60	R
	61	IR

Operation: dst <-- NOT dst

The contents of the destination location are complemented (one's complement); all 1 bits are changed to 0, and vice-versa.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 8 contains %24 (00100100), the statement

```
COM R8
```

leaves the value %DB (11011011) in that register. The Z and V flags are cleared and the S flag is set.

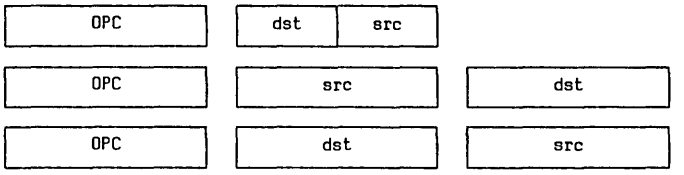
Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

CP
Compare

CP dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	src
6	A2	r	r
	A3	r	Ir
10	A4	R	R
	A5	R	IR
10	A6	R	IM
	A7	IR	IM

Operation: dst - src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags set accordingly. The contents of both operands are unaffected by the comparison.

Flags:

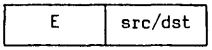
C: Cleared if there is a carry from the most significant bit of the result; set otherwise, indicating a "borrow"
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurs; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the register named TEST contains %63, working register 0 contains %30 (48 decimal), and register 48 contains %63, the statement

```
CP TEST, @R0
```

sets (only) the Z flag. If this statement is followed by "JP EQ, true_routine", the jump is taken.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



DA dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
8	40	R
	41	IR

Operation: dst <-- DA dst

The destination operand is adjusted to form two 4-bit BCD digits following a binary addition or subtraction operation on BCD encoded bytes. For addition (ADD, ADC), or subtraction (SUB, SBC), the following table indicates the operation performed:

Instruction	Carry Before DA	Bits 4-7 Value (Hex)	H Flag Before DA	Bits 0-3 Value (Hex)	Number Added To Byte	Carry After DA
ADD ADC	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
SUB SBC	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

If the destination operand is not the result of a valid addition or subtraction of BCD digits, the operation is undefined.

Flags:

C: Set if there is a carry from the most significant bit; cleared otherwise (see table above)
Z: Set if the result is 0; cleared otherwise
V: Undefined
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If addition is performed using the BCD values 15 and 27, the result should be 42. The sum is incorrect, however, when the binary representations are added in the destination location using standard binary arithmetic.

```
  0001  0101
+ 0010  0111
  0011  1100  = %3C
```

The DA statement adjusts this result so that the correct BCD representation is obtained.

```
  0011  1100
+ 0000  0110
  0100  0010  =  42
```

The C, Z, and S flags are cleared and V is undefined.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

DEC dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	00	R
	01	IR

Operation: dst <-- dst - 1

The destination operand's contents are decremented by one.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 10 contains %2A, the statement

```
DEC R10
```

leaves the value %29 in that register. The Z, V, and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

DECW
Decrement Word

DECW dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
10	80	RR
	81	IR

Operation: dst <-- dst - 1

The contents of the destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value which is decremented by one.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 0 contains %30 (48 decimal) and registers 48-49 contain the value %FAF3, the statement

```
DECW @R0
```

leaves the value %FAF2 in registers 48 and 49. The Z and V flags are cleared and S is set.

Note: When used to specify a 4-bit working-register pair address, address modes RR or IR use the format:

E	dst

DI
Disable Interrupts

DI

Instruction Format:

OPC

Cycles	OPC (Hex)
6	8F

Operation: IMR (7) <-- 0

Bit 7 of control register 251 (the Interrupt Mask Register) is reset to 0. All interrupts are disabled, although they remain potentially enabled (i.e., the Global Interrupt Enable is cleared--not the individual interrupt level enables.)

Flags: No flags affected

Example: If control register 251 contains %8A (10001010, that is, interrupts IRQ1 and IRQ3 are enabled), the statement

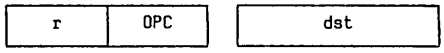
```
DI
```

sets control register 251 to %0A and disables these interrupts.

DJNZ
Decrement and Jump if Nonzero

DJNZ r,dst

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst
12 if jump taken 10 if jump not taken	rA r=0 to F	RA

Operation:

r <-- r - 1
If r ≠ 0, PC <-- PC + dst

The working register being used as a counter is decremented. If the contents of the register are not zero after decrementing, the relative address is added to the Program Counter (PC) and control passes to the statement whose address is now in the PC. The range of the relative address is +127, -128, and the original value of the PC is the address of the instruction byte following the DJNZ statement. When the working register counter reaches zero, control falls through to the statement following DJNZ.

Flags:

No flags affected

Example:

DJNZ is typically used to control a "loop" of instructions. In this example, 12 bytes are moved from one buffer area in the register file to another. The steps involved are:

- o Load 12 into the counter (working register 6)
- o Set up the loop to perform the moves
- o End the loop with DJNZ

```
     LD R6, #12          !Load Counter!
LOOP: LD R9,OLDBUF (R6)  !Move one byte to!
     LD NEWBUF (R6),R9   !New location!
     DJNZ R6,LOOP        !Decrement and !
                         !Loop until counter = 0!
```

Note:

The working register being used as a counter must be one of the registers 04-7F. Use of one of the I/O ports, control or peripheral registers will have undefined results.

EI

Instruction Format:

OPC

Cycles	OPC (Hex)
6	9F

Operation: IMR (7) <-- 1

Bit 7 of control register 251 (the Interrupt Mask Register) is set 10 to 1. This allows any potentially enabled interrupts to become enabled.

Flags: No flags affected

Example: If control register 251 contains %0A (00001010, that is, interrupts IRQ1 and IRQ3 potentially enabled), the statement

```
EI
```

sets control register 251 to %8A (10001010) and enables these interrupts.

INC
Increment

INC dst

Instruction Format:

dst	OPC

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	rE r=0 to F	r
6	20 21	R IR

Operation: dst <-- dst + 1

The destination operand's contents are incremented by one.

Flags:
C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 10 contains %2A, the statement

```
INC R10
```

leaves the value %2B in that register. The Z, V, and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

INCW
Increment Word

INCW dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
10	A0	RR
	A1	IR

Operation: dst <-- dst + 1

The contents of the destination (which must be an even address) and the byte following that location are treated as a single 16-bit value which is incremented by one.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working-register pair 0-1 contains the value %FAF3, the statement

```
INCW RR0
```

leaves the value %FAF4 in working-register pair 0-1. The Z and V flags are cleared and S is set.

Note: When used to specify a 4-bit working-register pair address, address modes RR or IR use the format:

E	dst

IRET
Interrupt Return

IRET

Instruction Format:

OPC

Cycles	OPC (Hex)
16	BF

Operation:

```
FLAGS <-- @SP
SP <-- SP + 1
PC <-- @SP
SP <-- SP + 2
IMR (7) <-- 1
```

This instruction is issued at the end of an interrupt service routine. It restores the Flag register (control register 252) and the PC. It also reenables any interrupts that are potentially enabled.

Flags: All flags are restored to original settings (before interrupt occurred).

JP cc,dst

Instruction Format:

			Cycles	OPC (Hex)	Address Mode dst
Conditional					
cc	OPC	dst	12 if jump taken 10 if jump not taken	ccD cc=0 to F	DA
Unconditional					
OPC		dst	8	30	IRR

Operation: If cc is true, PC <-- dst

A conditional jump transfers Program Control to the destination address if the condition specified by "cc" is true; otherwise, the instruction following the JP instruction is executed. See Section 6.4 for a list of condition codes.

The unconditional jump simply replaces the contents of the Program Counter with the contents of the specified register pair. Control then passes to the statement addressed by the PC, decremented by one.

Flags: No flags affected

Example: If the carry flag is set, the statement

```
JP C,%1520
```

replaces the contents of the Program Counter with %1520 and transfers control to that location. Had the carry flag not been set, control would have fallen through to the statement following the JP.

Note: When used to specify a 4-bit working-register pair address, address mode IRR uses the format:

E	dst

JR
Jump Relative

JR cc,dst

Instruction Format:

Cycles	OPC (Hex)	Address Mode dst
12 If jump taken 10 If jump not taken	ccB cc=0 to F	RA

Operation: If cc is true, PC <-- PC + dst

If the condition specified by "cc" is true, the relative address is added to the PC and control passes to the statement whose address in now in the PC; otherwise, the instruction following the JR instruction is executed. (See Section 5.3 for a list of condition codes). The range of the relative address is +127, -128, and the original value of the PC is taken to be the address of the first instruction byte following the JR statement.

Flags: No flags affected

Example: If the result of the last arithmetic operation executed is negative, the following four statements (which occupy a total of seven bytes) are skipped with the statement

```
JR MI,$+9
```

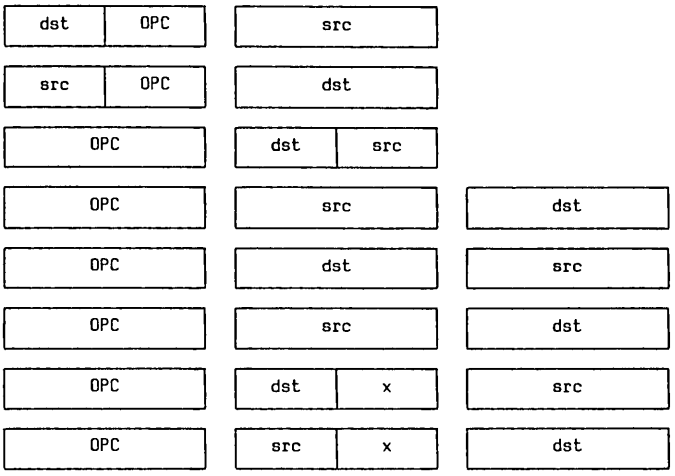
If the result is not negative, execution continues with the statement following the JR. A short form of a jump to label LO is

```
JR LO
```

where LO must be within the allowed range. The condition code is "blank" in this case, and is assumed to be "always true."

LD dst,src

Instruction Format:



Cycles	OPC (Hex)	Address dst	Mode src
6	rC	r	IM
6	r8	r	R
6	r9 r=0 to F	R*	r
6	E3	r	Ir
6	F3	Ir	r
10	E4	R	R
10	E5	R	IR
10	E6	R	IM
10	E7	IR	IM
10	F5	IR	R
10	C7	r	X
10	D7	X	r

*In this instance only a full 8-bit register address can be used.

Operation: dst <-- src

The contents of the source are loaded into the destination. The contents of the source are not affected.

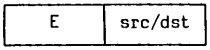
Flags: No flags affected

Example: If working register 0 contains %0B (11 decimal) and working register 10 contains %83, the statement

```
LD 240(R0),R10
```

will load the value %83 into register 251 (240 + 11). Since this is the Interrupt Mask register, the Load statement has the effect of enabling IRQ0 and IRQ1. The contents of working register 10 are unaffected by the load.

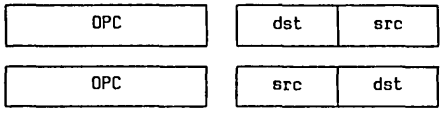
Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



LDC
Load Constant

LDC dst,src

Instruction Format:



	Cycles	OPC (Hex)	Address Mode dst	Address Mode src
OPC dst src	12	C2	r	Irr
OPC src dst	12	D2	Irr	r

Operation: dst <-- src

This instruction is used to load a byte constant from program memory into a working register, or vice-versa. The address of the program memory location is specified by a working register pair. The contents of the source are not affected.

Flags: No flags affected

Example: If the working-register pair 6-7 contains %30A2 and program-memory location %30A2 contains the value %22, the statement

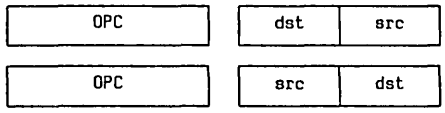
```
LDC R2, @RR6
```

loads the value %22 into working register 2. The value of location %30A2 is unchanged by the load.

LDCI
Load Constant Autoincrement

LDCI dst,src

Instruction Format:



	Cycles	OPC (Hex)	Address Mode dst	src
OPC dst src	18	C3	Ir	Irr
OPC src dst	18	D3	Irr	Ir

Operation:

dst <-- src
r <-- r + 1
rr <-- rr + 1

This instruction is used for block transfers of data between program memory and the register file. The address of the program-memory location is specified by a working-register pair, and the address of the register-file location is specified by a working register. The contents of the source location are loaded into the destination location. Both addresses are then incremented automatically. The contents of the source are not affected.

Flags:

No flags affected

Example:

If the working-register pair 6-7 contains %30A2 and program-memory locations %30A2 and %30A3 contain %22BC, and if working register R2 contains %20 (32 decimal), the statement

```
LDCI @R2, @RR6
```

loads the value %22 into register 32. A second

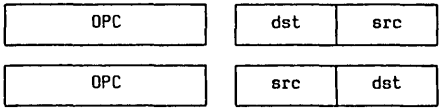
```
LDCI @R2, @RR6
```

loads the value %BC into register 33.

LDE
Load External Data

LDE dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	src
12	82	r	Irr
12	92	Irr	r

Operation: dst <-- src

This instruction is used to load a byte from external data memory into a working register or vice-versa. The address of the external data-memory location is specified by a working-register pair. The contents of the source are not affected.

Flags: No flags affected

Example: If the working-register pair 6-7 contains %404A and working register 2 contains %22, the statement

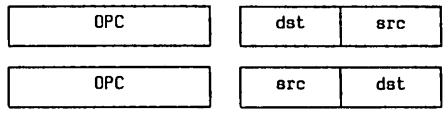
```
LDE @RR6,R2
```

loads the value %22 into external data-memory location %404A.

LDEI
Load External Data Autoincrement

LDEI dst,src

Instruction Format:



	Cycles	OPC (Hex)	Address Mode dst	src
OPC dst src	18	83	Ir	Irr
OPC src dst	18	93	Irr	Ir

Operation:

dst <-- src
r <-- r + 1
rr <-- rr + 1

This instruction is used for block transfers of data between external data memory and the register file. The address of the external data-memory location is specified by a working-register pair, and the address of the register file location is specified by a working register. The contents of the source location are loaded into the destination location. Both addresses are then incremented automatically. The contents of the source are not affected.

Flags:

No flags affected

Example:

If the working-register pair 6-7 contains %404A, working register 2 contains %22 (34 decimal), and registers 34-35 contain %ABC3, the statement

```
LDEI @RR6,@R2
```

loads the value %AB into external location %404A. A second

```
LDEI @RR6,@R2
```

loads the value %C3 into external location %404B.

NOP
No Operation

NOP

Instruction Format:

OPC

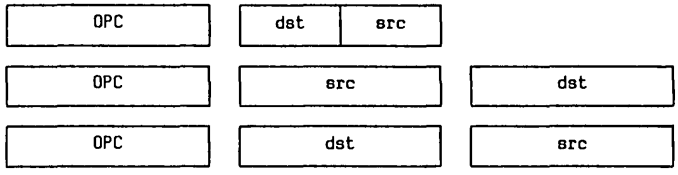
Cycles	OPC (Hex)
6	FF

Operation: No action is performed by this instruction. It is typically used for timing delays.

Flags: No flags affected

OR dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	Address Mode src
6	42	r	r
6	43	r	Ir
10	44	R	R
10	45	R	IR
10	46	R	IM
10	47	IR	IM

Operation: dst <-- dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are not affected. The OR operation results in a one bit being stored whenever either of the corresponding bits in the two operands is 1; otherwise a 0 bit is stored.

Flags:

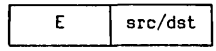
C: Unaffected
Z: Set if result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the source operand is the immediate value %7B (01111011) and the register named TARGET contains %C3 (11000011), the statement

```
OR TARGET,#%7B
```

leaves the value %FB (11111011) in register TARGET. The Z and V flags are cleared and S is set.

Note: When used to specify a 4-bit working-register address, address modes R and IR use the format:



POP
Pop

POP dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
10	50	R
10	51	IR

Operation:

dst <-- @SP
SP <-- SP + 1

The contents of the location addressed by the SP are loaded into the destination. The SP is then incremented automatically.

Flags:

No flags affected

Example:

If the SP (control registers 254-255) contains %1000, external data-memory location %1000 contains %55, and working register 6 contains %22 (34 decimal), the statement

```
POP @R6
```

loads the value %55 into register 34. After the POP operation, the SP contains %1001.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

PUSH src

Instruction Format:

OPC	src

Cycles		OPC (Hex)	Address Mode src
10	Internal stack	70	R
12	External stack		
12	Internal stack	71	IR
14	External stack		

Operation:

SP <-- SP - 1
@SP <-- src

The contents of the SP are decremented, then the contents of the source are loaded into the location addressed by the decremented SP, thus adding a new element to the top of the stack.

Flags:

No flags affected

Example:

If the SP contains %1001, the statement

PUSH FLAGS

stores the contents of the register named FLAGS in location %1000. After the PUSH operation, the SP contains %1000.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src

RCF
Reset Carry Flag

RCF

Instruction Format:

OPC

Cycles	OPC (Hex)
6	CF

Operation: C <-- 0

The C flag is reset to 0, regardless of its previous value.

Flags: C: Reset to 0
No other flags affected

RET

Instruction Format:

OPC

Cycles	OPC (Hex)
14	AF

Operation:

PC <-- @SP
SP <-- SP + 2

This instruction is normally used to return to the previously executed procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the SP are popped into the PC. The next statement executed is that addressed by the new contents of the PC.

Flags:

No flags affected

Example:

If the PC contains %35B4, the SP contains %2000, external data-memory location %2000 contains %18, and location %2001 contains %B5, then the statement

```
RET
```

leaves the value %2002 in the SP and the PC contains %18B5, the address of the next instruction.

RL
Rotate Left

RL dst

Instruction Format:

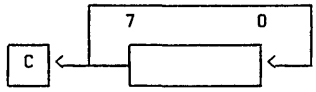
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	90	R
6	91	IR

Operation:

C <-- dst(7)
dst(0) <-- dst(7)
dst(n + 1) <-- dst(n) n = 0 - 6

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit 0 position and also replaces the carry flag.



Flags:

C: Set if the bit rotated from the most significant bit position was 1; i.e., bit 7 was 1
Z: Set if the result is zero; cleared otherwise.
V: Set if arithmetic overflow occurred; that is, if the sign of the destination changed during rotation; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the contents of the register named SHIFTER are %88 (10001000), the statement

```
RL SHIFTER
```

leaves the value %11 (00010001) in that register. The C flag and V flags are set to 1 and the Z flag is cleared.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E .	dst

RLC
Rotate Left Through Carry

RLC dst

Instruction Format:

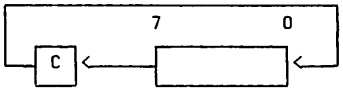
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	10	R
6	11	IR

Operation:

dst (0) <-- C
C <-- dst (7)
dst(n + 1) <-- dst(n) n = 0 - 6

The contents of the destination operand with the C flag are rotated left one bit position. The initial value of bit 7 replaces the C flag; the initial value of the C flag replaces bit 0.



Flags:

C: Set if the bit rotated from the most significant bit position was 1; i.e., bit 7 was 1
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if the sign of the destination changed during rotation; cleared otherwise
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the C flag is reset (to 0) and the register named SHIFTER contains %8F (10001111), the statement

RLC SHIFTER

sets the C flag and the V flag to 1 and SHIFTER contains %1E (00011110).

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

RR
Rotate Right

RR dst

Instruction Format:

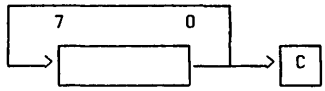
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	E0	R
6	E1	IR

Operation:

C <-- dst(0)
dst(7) <-- dst(0)
dst(n) <-- dst(n + 1) n = 0 - 6

The contents of the destination operand are rotated right one bit position. The initial value of bit 0 is moved to bit 7 and also replaces the C flag.



Flags:

C: Set if the bit rotated from the least significant bit position was 1; i.e., bit 0 was 1
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the contents of working register 6 are %31 (00110001), the statement

```
RR R6
```

sets the C flag to 1 and leaves the value %98 (10011000) in working register 6. Since bit 7 now equals 1, the S flag and the V flag are also set.

Note:

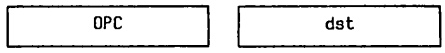
When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

RRC
Rotate Right Through Carry

RRC dst

Instruction Format:

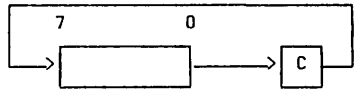


Cycles	OPC (Hex)	Address Mode dst
6	C0	R
6	C1	IR

Operation:

```
dst(7) <-- C
C <-- dst(0)
dst(n) <-- dst(n + 1)   n = 0 - 6
```

The contents of the destination operand with the C flag are rotated right one bit position. The initial value of bit 0 replaces the C flag; the initial value of the C flag replaces bit 7.



Flags:

C: Set if the bit rotated from the least significant bit position was 1; i.e., bit 0 was 1
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred, that is, the sign of the destination changed during rotation; cleared otherwise
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the contents of the register named SHIFTER are %DD (11011101) and the Carry flag is reset to 0, the statement

```
RRC SHIFTER
```

sets the C flag and the V flag and leaves the value %6E (01101110) in the register.

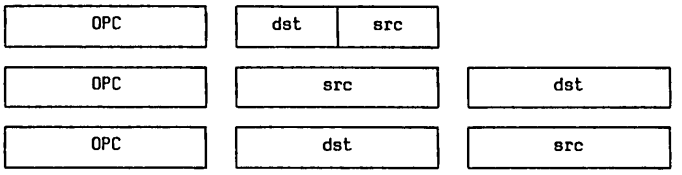
Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

SBC
Subtract With Carry

SBC dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	Address Mode src
6	32	r	r
6	33	r	Ir
10	34	R	R
10	35	R	IR
10	36	R	IM
10	37	IR	IM

Operation: dst <-- dst - src - C

The source operand, along with the setting of the C flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are not affected. Subtraction is performed by adding the two's complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of low-order operands to be subtracted from the subtraction of high-order operands.

Flags:

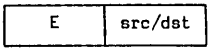
C: Cleared if there is a carry from the most significant bit of the result; set otherwise, indicating a "borrow"
Z: Set if the result is 0; cleared otherwise
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; reset otherwise
S: Set if the result is negative; cleared otherwise
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow."
D: Always set to 1

Example: If the register named MINUEND contains %16, the Carry flag is set to 1, working register 10 contains %20 (32 decimal), and register 32 contains %05, the statement

```
SBC MINUEND, @R10
```

leaves the value %10 in register MINUEND. The C, Z, V, S and H flags are cleared and D is set.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



SCF
Set Carry Flag

SCF

Instruction Format:

OPC

Cycles	OPC (Hex)
6	DF

Operation: C <-- 1

The C flag is set to 1, regardless of its previous value.

Flags: **C:** Set to 1
No other flags affected

SRA
Shift Right Arithmetic

SRA dst

Instruction Format:

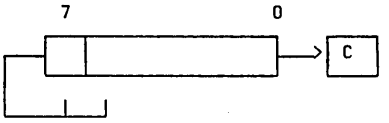
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	D0	R
6	D1	IR

Operation:

dst(7) <-- dst(7)
C <-- dst(0)
dst(n) <-- dst(n + 1) n = 0 - 6

An arithmetic shift right one bit position is performed on the destination operand. Bit 0 replaces the C flag. Bit 7 (the Sign bit) is unchanged, and its value is also shifted into bit position 6.



Flags:

C: Set if the bit shifted from the least significant bit position was 1; i.e., bit 0 was 1
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the register named SHIFTER contains %B8 (10111000), the statement

```
SRA SHIFTER
```

resets the C flag to 0 and leaves the value %DC (11011100) in register SHIFTER. The S flag is set to 1.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

SRP src

Instruction Format:

OPC	src

Cycles	OPC (Hex)	Address Mode src
6	31	IM

Operation: RP <-- src

The specified value is loaded into bits 4-7 of the Register Pointer (RP) (control register 253). Bits 0-3 of the RP are always set to 0. The source data (with bits 0-3 forced to 0) is the starting address of a working-register group. The working-register group starting addresses are:

Hex	Decimal	
%00	0	
%10	16	
%20	32	
%30	48	
%40	64	
%50	80	
%60	96	
%70	112	
%F0	240	(control and peripheral registers)

Values in the range %80-E0 are invalid.

Flags: No flags affected

Example: Assume the RP currently addresses the control and peripheral register group and the program has just entered an interrupt service routine. The statement

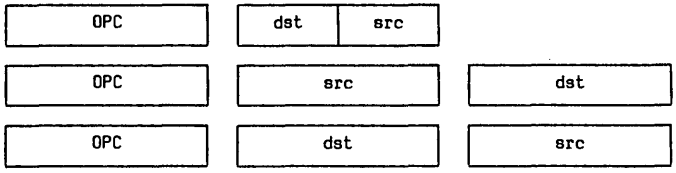
```
SRP #%70
```

saves the contents of the control and peripheral registers by setting the RP to %70 (01110000), or 112 decimal. Any reference to working registers in the interrupt routine will point to registers 112-127.

SUB
Subtract

SUB dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	src
6	22	r	r
6	23	r	Ir
10	24	R	R
10	25	R	IR
10	26	R	IM
10	27	IR	IM

Operation: dst <-- dst - src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are not affected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

Flags:

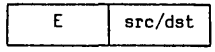
C: Cleared if there is a carry from the most significant bit of the result; set otherwise, indicating a "borrow"
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is the same as the sign of the source operand; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow."
D: Always set to 1

Example: If the register named MINUEND contains %29, the statement

```
SUB MINUEND, #%11
```

will leave the value %18 in the register. The C, Z, V, S and H flags are cleared and D is set.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



SWAP
Swap Nibbles

SWAP dst

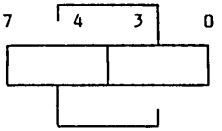
Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
8	F0	R
8	F1	IR

Operation: dst(0 - 3) <--> dst(4 - 7)

The contents of the lower four bits and upper four bits of the destination operand are swapped.



Flags:

C: Undefined
Z: Set if the result is zero; cleared otherwise
V: Undefined
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: Suppose the register named BCD_Operands contains %B3 (10110011). The statement

```
SWAP BCD_Operands
```

will leave the value %3B (00111011) in the register. The Z and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

TCM
Test Complement Under Mask

TCM dst,src

Instruction Format:

			Cycles	OPC (Hex)	Address Mode dst	Address Mode src
OPC	dst	src	6	62	r	r
			6	63	r	Ir
OPC	src	dst	10	64	R	R
			10	65	R	IR
OPC	dst	src	10	66	R	IM
			10	67	IR	IM

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logical "1" value. The bits to be tested are specified by setting a 1 bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The Zero (Z) flag can then be checked to determine the result. When the TCM operation is complete, the destination location still contains its original value.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the register named TESTER contains %F6 (11110110) and the register named MASK contains %06 (00000110), that is, bits 1 and 2 are being tested for a 1 value, the statement

```
TCM TESTER, MASK
```

complements TESTER (to 00001001) and then do a logical AND with register MASK, resulting in %00. A subsequent test of the Z flag,

```
JP Z,plabel
```

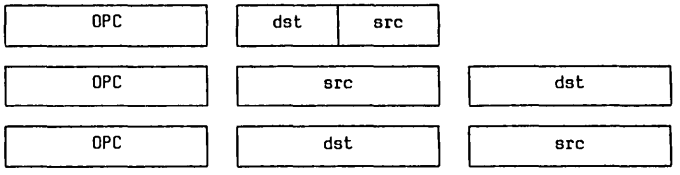
causes a transfer of program control. At the end of this sequence, TESTER still contains %F6.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

TM dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	src
6	72	r	r
6	73	r	Ir
10	74	R	R
10	75	R	IR
10	76	R	IM
10	77	IR	IM

Operation: dst AND src

This instruction tests selected bits in the destination operand for a logical "0" value. The bits to be tested are specified by setting a 1 bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The Z flag can be checked to determine the result. When the TM operation is complete, the destination location still contains its original value.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the register named TESTER contains %F6 (11110110) and the register named MASK contains %06 (00000110), that is, bits 1 and 2 are being tested for a 0 value, the statement

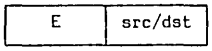
```
TM TESTER, MASK
```

results in the value %06 (00000110). A subsequent test for nonzero

```
JP NZ, plabel
```

causes a transfer of program control. At the end of this sequence, TESTER still contains %F6. The Z and S flags are cleared.

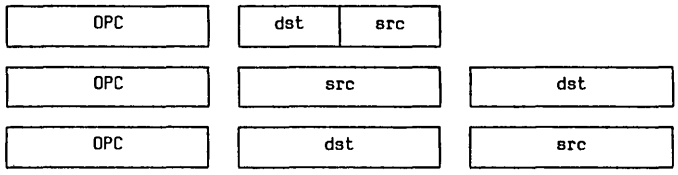
Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



XOR
Logical Exclusive OR

XOR dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	src
6	B2	r	r
6	B3	r	Ir
10	B4	R	R
10	B5	R	IR
10	B6	R	IM
10	B7	IR	IM

Operation: dst <-- dst XOR src

The source operand is logically EXCLUSIVE ORed with the destination operand and the result stored in the destination. The EXCLUSIVE OR operation results in a one bit being stored whenever the corresponding bits in the operands are different; otherwise, a 0 bit is stored.

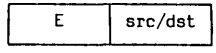
Flags:
C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the source operand is the immediate value %7B (011111011) and the register named TARGET contains %C3 (11000011), the statement

```
OR TARGET, #%7B
```

leaves the value %B8 (10111000) in the register.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



Chapter 6
External Interface (Z8601, Z8611)

6.1 INTRODUCTION

The ROM versions of the Z8 microcomputer have 40 external pins, of which 32 are programmable I/O pins. The remaining 8 pins are used for power and control. Up to 16 I/O pins can be configured as an external memory interface. This interface function is the subject of this chapter. The I/O mode of these pins is described in Chapter 9.

6.2 PIN DESCRIPTIONS

$\overline{AS}$. Address Strobe (output, active Low, 3-state, pin 9). Address Strobe is pulsed Low once at the beginning of each machine cycle. The rising edge of $\overline{AS}$ indicates that addresses, Read/Write (R/$\overline{W}$), and Data Memory ($\overline{DM}$) signals, are valid when output for external program or data memory transfers. Under program control, $\overline{AS}$ can be placed in a high-impedance state along with Ports 0 and 1, Data Strobe ($\overline{DS}$), and R/$\overline{W}$.

$\overline{DS}$. Data Strobe (output, active Low, 3-state, pin 8). Data Strobe provides the timing for data movement to or from Port 1 for each external memory transfer. During a Write cycle, data out is valid at the leading edge of $\overline{DS}$. During a Read cycle, data in must be valid prior to the trailing edge of $\overline{DS}$. $\overline{DS}$ can be placed in a high-impedance state along with Ports 0 and 1, $\overline{AS}$, and R/$\overline{W}$.

R/$\overline{W}$. Read/Write. (output, 3-state, pin 7). Read/Write determines the direction of data transfer for external memory transactions. R/$\overline{W}$ is Low when writing to external program or data memory, and High for all other transactions. R/$\overline{W}$ can be placed in a high-impedance state along with Ports 0 and 1, $\overline{AS}$, and $\overline{DS}$.

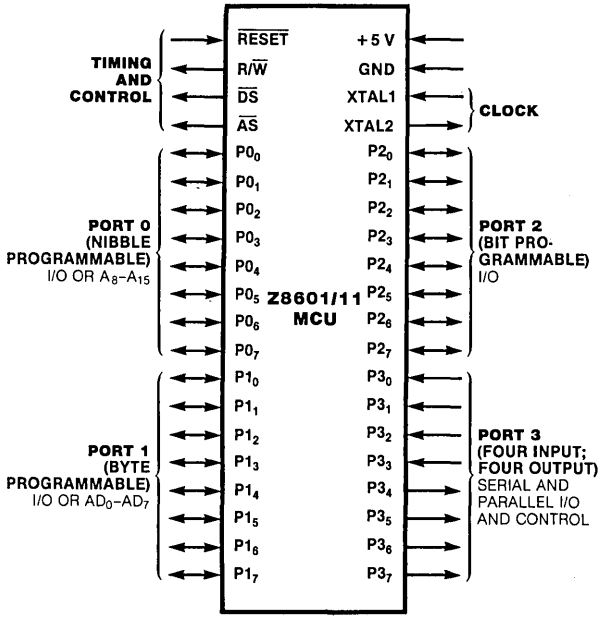


Figure 6-1. Z8601/11 Pin Functions

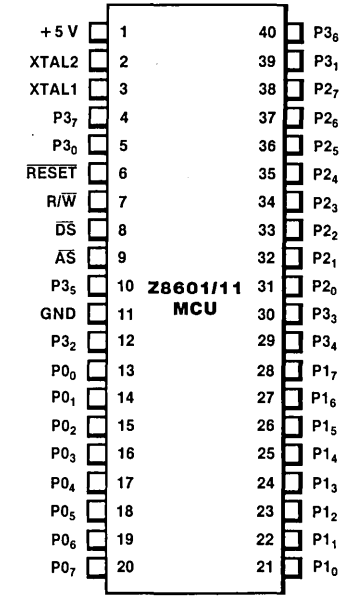


Figure 6-2. Z8601/11 Pin Assignments

$P0_0$-$P0_7$, $P1_0$-$P1_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. I/O port lines (inputs/outputs, TTL-compatible, pins 12-40). These 32 I/O lines are divided into four 8-bit I/O ports that can be configured under program control for I/O or external memory interface. Individual lines of a port are denoted by the second digit of the port number. For example, $P3_0$ refers to bit 0 of Port 3. Ports 0 and 1 can be placed in a high-impedance state along with $\overline{AS}$, $\overline{DS}$, and R/$\overline{W}$.

$\overline{RESET}$. Reset (input, active Low, pin 6). $\overline{RESET}$ initializes the Z8. When $\overline{RESET}$ is deactivated, program execution begins from internal program location %C. If held Low, $\overline{RESET}$ acts as a register file protect during power-down and power-up sequences. $\overline{RESET}$ also enables the Z8 Test mode.

XTAL1, XTAL2. Crystal 1, Crystal 2 (oscillator input and output, pins 3 and 2). These pins connect a parallel-resonant crystal (12 MHz maximum) or an external source (12 MHz maximum) to the on-board clock oscillator and buffer.

6.3 CONFIGURING FOR EXTERNAL MEMORY

Before interfacing with external memory, the user must configure Ports 0 and 1 appropriately. The minimum bus configuration uses Port 1 as a multiplexed Address/Data port (AD_0-AD_7), allowing access to 256 bytes of external memory. In this configuration, the eight lower order address bits (A_0-A_7) are multiplexed with the data (D_0-D_7).

Port 0 can be programmed to provide four additional address lines (A_8-A_{11}), which increases the externally addressable program memory to 4K bytes. Port 0 can also be programmed to provide eight additional address lines (A_8-A_{15}), which increases the externally addressable memory to 62K bytes for the Z8601 or 60K bytes for the Z8611. Refer to Chapter 3, Figures 3-5 and 3-6, for external memory maps.

Ports 0 and 1 are configured for external memory operation by writing the appropriate bits in the Port 0-1 Mode register (Figure 6-3).

For example, Port 1 can be defined as a multiplexed Address/Data port (AD_0-AD_7) by setting D_4 to 1 and D_3 to 0. The lower nibble of Port 0 can be defined as address lines A_8-A_{11}, by setting D1 to 1. Similarly, setting D7 to 1 defines the upper nibble of Port 0 as address lines A_{12}-A_{15}. Whenever Port 0 is configured to output address lines A_{12}-A_{15}, A_8-A_{11} must also be selected as address lines.

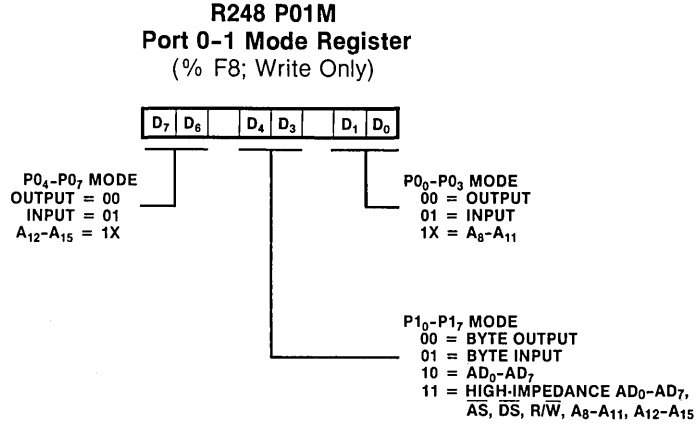


Figure 6-3. Ports 0 and 1 External Memory Operation

Once Port 1 is configured as an Address/Data port, it can no longer be used as a register. Attempting to read Port 1 returns FF; writing has no effect. Similarly, if Port 0 is configured for address lines A_8-A_{15}, it can no longer be used as a register. However, if only the lower nibble is defined as address lines A_8-A_{11}, the upper nibble is still addressable as an I/O register. Reading Port 0 with only the lower nibble defined as address outputs returns XF, where X equals the data in bits D_4-D_7. Writing to Port 0 transfers data to the I/O nibble only.

An instruction to change the modes of Ports 0 or 1 should not be immediately followed by an instruction that performs a stack operation, because this may cause indeterminate program flow. In addition, after setting the modes of Ports 0 and 1 for external memory, the next three bytes must be fetched from internal program memory.

6.4 EXTERNAL STACKS

Z8 architecture supports stack operations in either the register file or data memory. A stack's location is determined by bit D_2 in the Port 0-1 Mode register. For example, if D_2 is set to 1, the stack is in internal data memory (Figure 6-4).

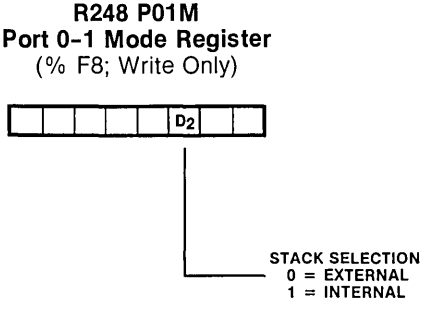


Figure 6-4. Ports 0 and 1 Stack Selection

The instruction used to change the stack selection bit should not be immediately followed by the instructions RET or IRET, because this will cause indeterminate program flow.

6.5 DATA MEMORY

The two external memory spaces, data and program, can be addressed as a single memory space or as two separate spaces of equal size; i.e., 62K bytes each for the Z8601 and 60K bytes each for the Z8611. If the memory spaces are separated, program memory and data memory are logically selected by the Data Memory select output ($\overline{DM}$). DM is available on Port 3, line 4 ($P3_4$) by setting bits D_4 and D_3 in the Port 3 Mode register to 10 or 01 (Figure 6-5). $\overline{DM}$ is active Low during the execution of the LDE, LDEI instructions. $\overline{DM}$ is also active during the execution of CALL, POP, PUSH, RET and IRET instructions if the stack resides in external memory.

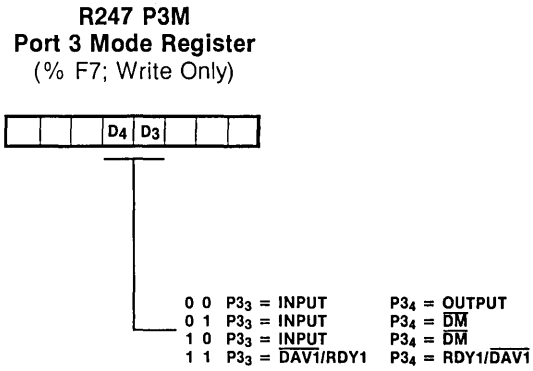


Figure 6-5. Data Memory Operation

6.6 BUS OPERATION

The timing for typical data transfers between the Z8 and external memory is illustrated in Figure 6-6. Machine cycles can vary from six to twelve clock periods depending on the operation being performed. The notations used to describe the basic timing periods of the Z8 are: machine cycles (Mn), timing states (Tn), and clock periods. All timing references are made with respect to the output signals $\overline{AS}$ and $\overline{DS}$. The clock is shown for clarity only and does not have a specific timing relationship with other signals.

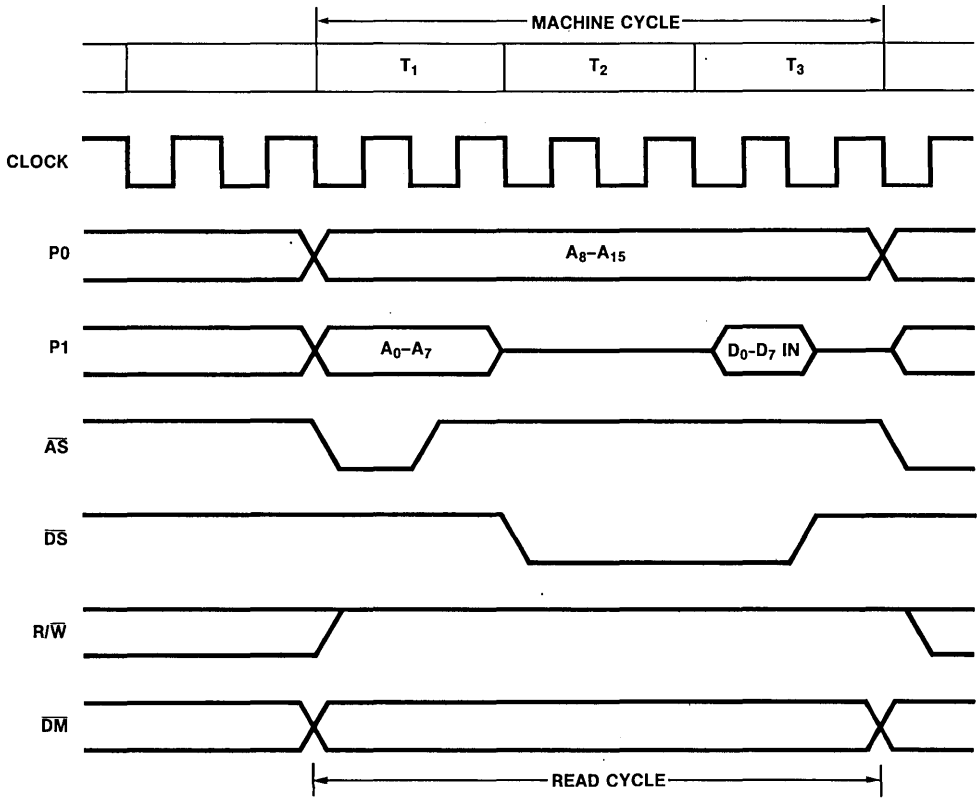


Figure 6-6a. External Instruction Fetch, or Memory Read Cycle

6.6.1 Address Strobe ($\overline{AS}$)

All transactions start with $\overline{AS}$ driven Low and then raised High by the Z8. The rising edge of $\overline{AS}$ indicates that R/$\overline{W}$, $\overline{DM}$, and the addresses output from Ports 0 and 1 are valid. The addresses output via Port 1 remain valid only during MnT1 and typically need to be latched using $\overline{AS}$, whereas Port 0 address outputs remain stable throughout the machine cycle.

6.6.2 Data Strobe

The Z8 uses $\overline{DS}$ to time the actual data transfer. For Write operations (R/$\overline{W}$ = Low), a Low on $\overline{DS}$ indicates that valid data is on the Port 1 AD_0-AD_7 lines. For Read operations, (R/$\overline{W}$ = High), the Address/Data bus is placed in a high-impedance state before driving $\overline{DS}$ Low so that the addressed device can put its data on the bus. The Z8 samples this data prior to raising $\overline{DS}$ High.

6.6.3 External Memory Operations

Whenever the Z8 is configured for external memory operation, the addresses of all internal program memory references appear on the external bus. This should have no effect on the external system since the bus control lines, $\overline{DS}$ and R/$\overline{W}$, remain in their inactive High state. $\overline{DS}$ and R/$\overline{W}$ become active only during external memory references.

CAUTION

Do not use LDC, LDCI, LDE or LDEI to write to internal program memory. The execution of these instructions causes the Z8 to assume that an external write operation is being performed and this will activate control signals $\overline{DS}$ and R/$\overline{W}$.

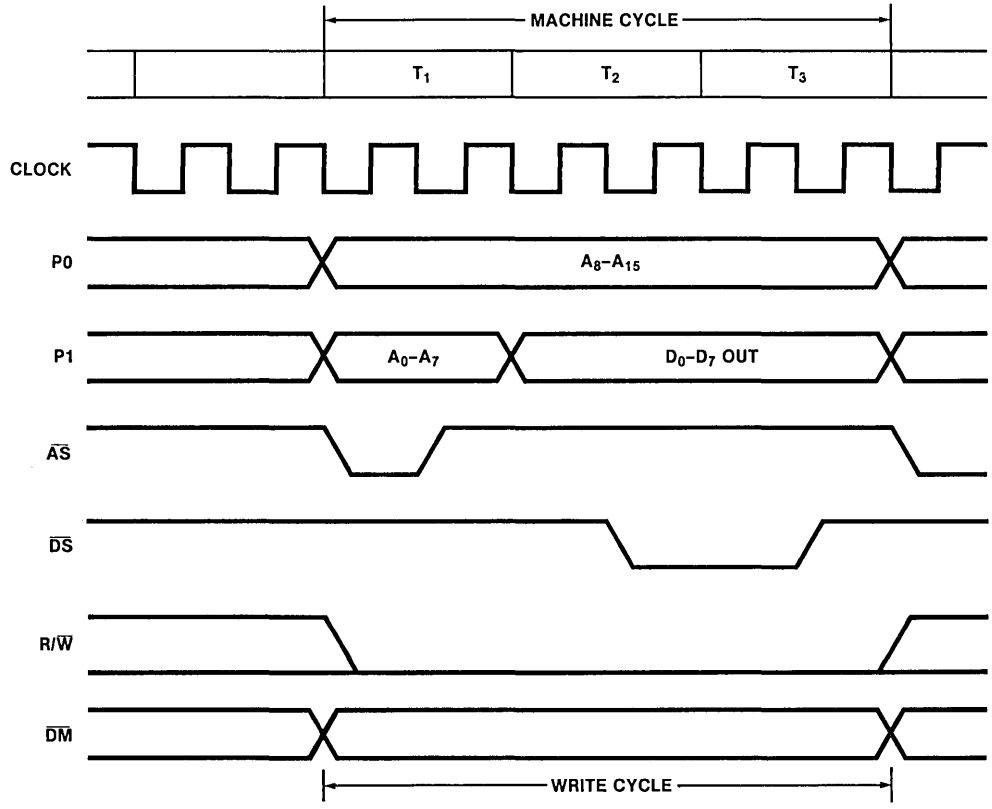


Figure 6-6b. External Memory Write Cycle

6.7 SHARED BUS

Port 1, along with $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, and Port 0 nibbles configured as address lines, can be placed in a high-impedance state, allowing the Z8601 or the Z8611 to share common resources with other bus masters. This shared bus mode is under software control and is programmed by setting Port 0-1 Mode register bits D_4 and D_3 both to 1 (Figure 6-7).

Data transfers can be controlled by assigning, for example, $P3_3$ as a Bus Acknowledge input and $P3_4$ as a Bus Request output. Bus Request/Acknowledge control sequences must be software driven.

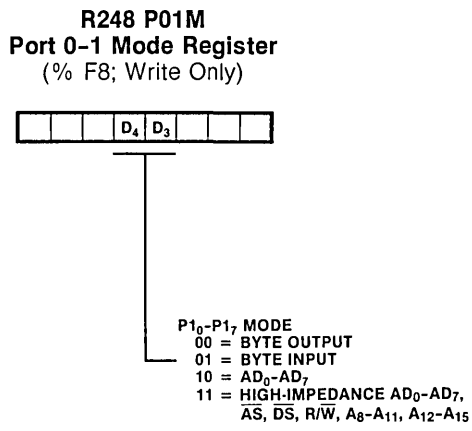


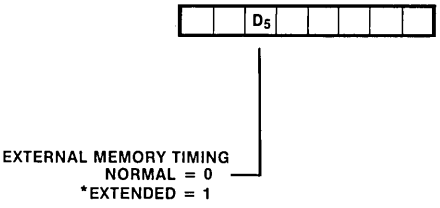
Figure 6-7. Shared Bus Operation

6.8 EXTENDED BUS TIMING

The Z8601 and Z8611 can accommodate slow memory access times by automatically inserting an additional state time (Tx) into the bus cycle. This stretches the $\overline{DS}$ timing by two clock periods, though internal memory access time is not affected. Timing is extended by setting bit D_5 in the Port 0-1 Mode register to 1 (Figure 6-8).

Figures 6-9a and 6-9b illustrate extended memory Read and Write cycles.

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 6-8. Extended Bus Timing

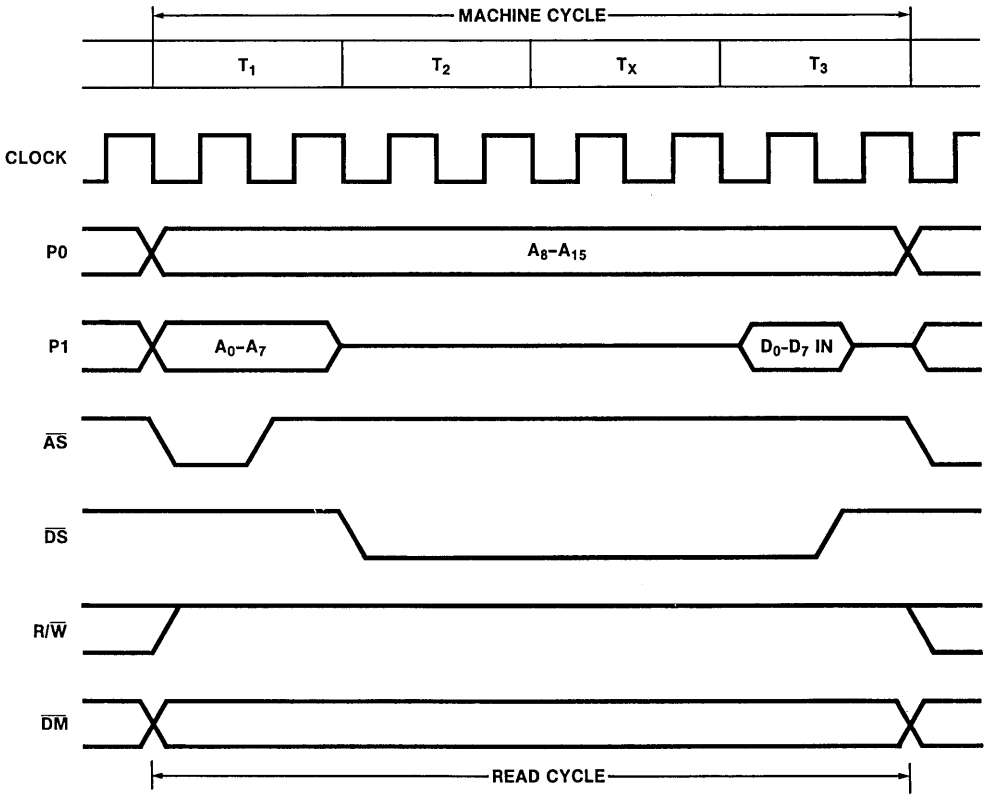


Figure 6-9a. Extended External Instruction Fetch, or Memory Read Cycle

6.9 INSTRUCTION TIMING

The high throughput of the Z8 is due, in part, to the use of instruction pipelining, in which the instruction fetch and execution cycles are overlapped. During the execution of an instruction the opcode of the next instruction is fetched. This is illustrated in Figure 6-10.

Figures 6-11 and 6-12 show typical instruction cycle timing for instructions fetched from external memory. (It should be noted that all instruction fetch cycles have the same machine timing regardless of whether memory is internal or external.) For those instructions that require execution time longer then that of the overlapped fetch, or instructions that reference program or data memory as part of their execution, the pipe must be flushed. In order to calculate the execution time of a program, the internal clock periods shown in the cycles column of the instruction formats in Section 5.4 should be added together. The cycles are equal to one-half the crystal or input clock rate.

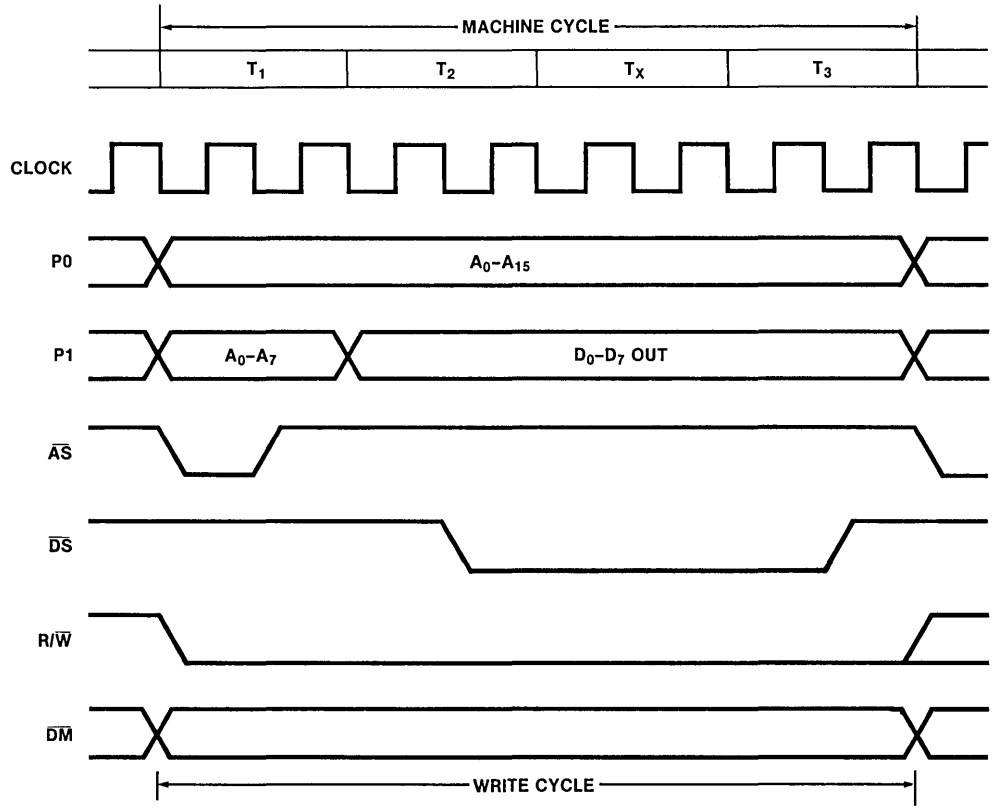


Figure 6-9b. Extended External Memory Write Cycle

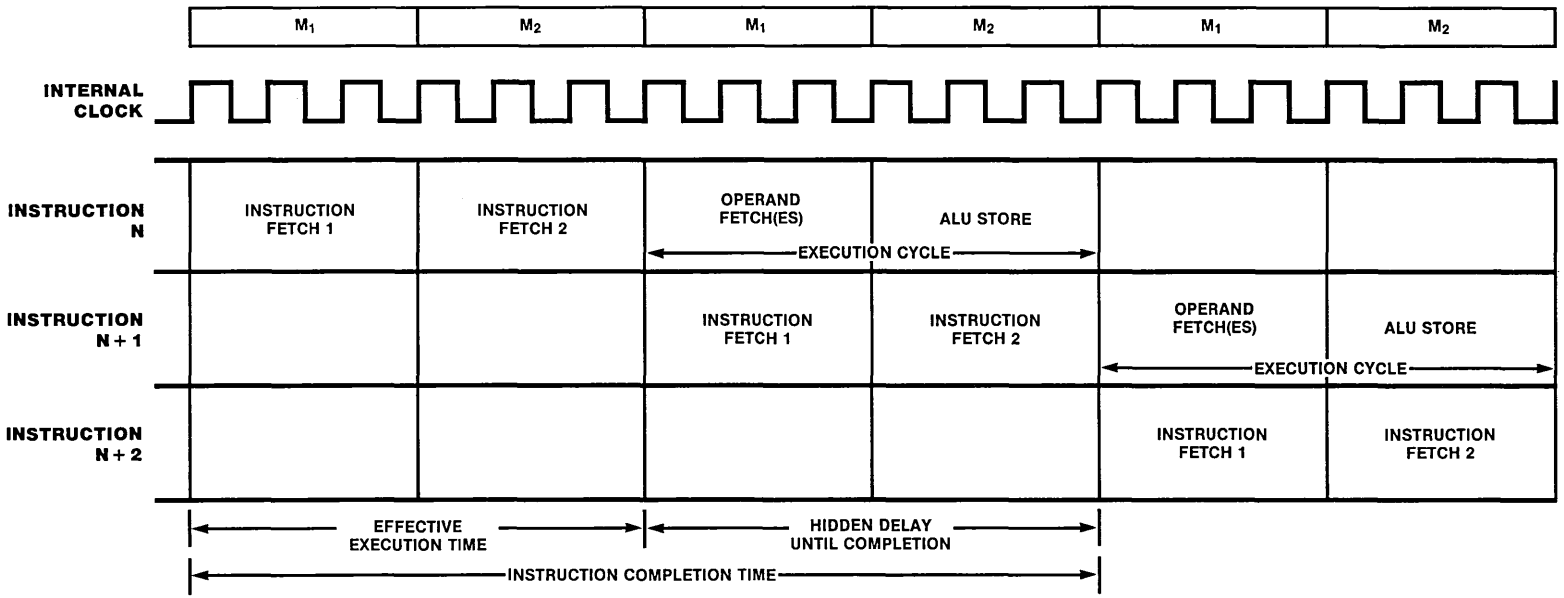


Figure 6-10. Instruction Pipelining

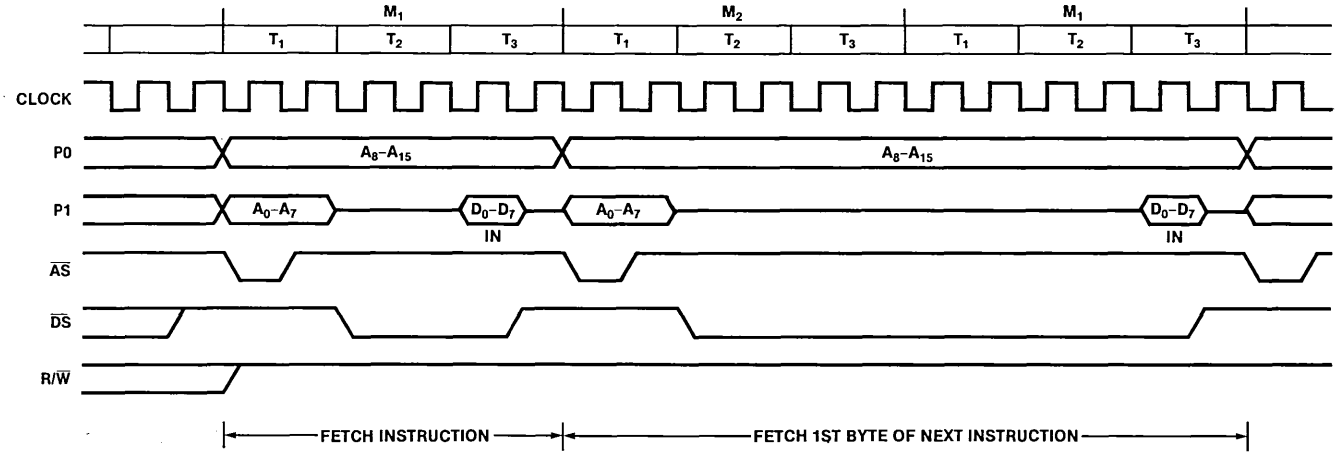


Figure 6-11. Instruction Cycle Timing (One Byte Instructions)

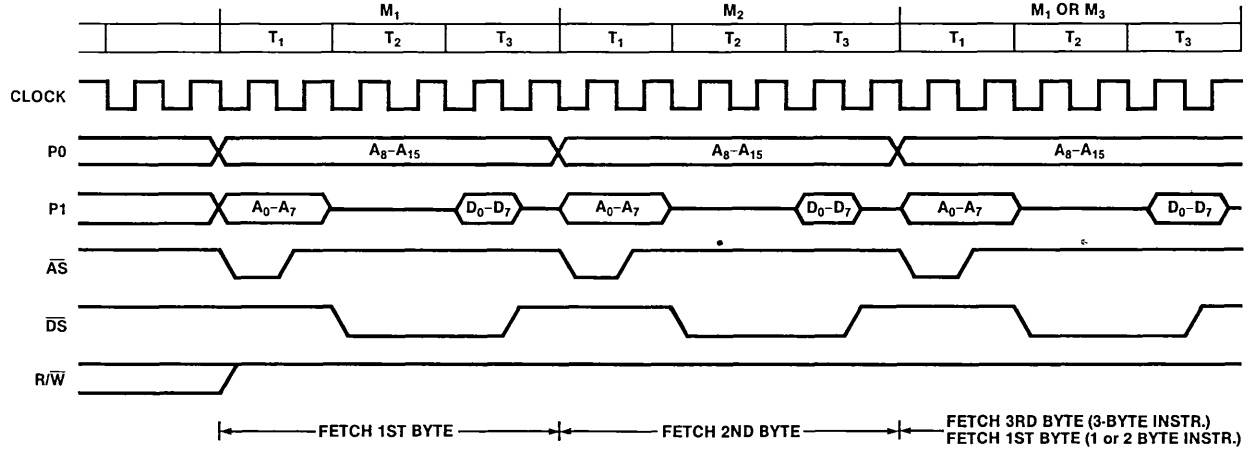
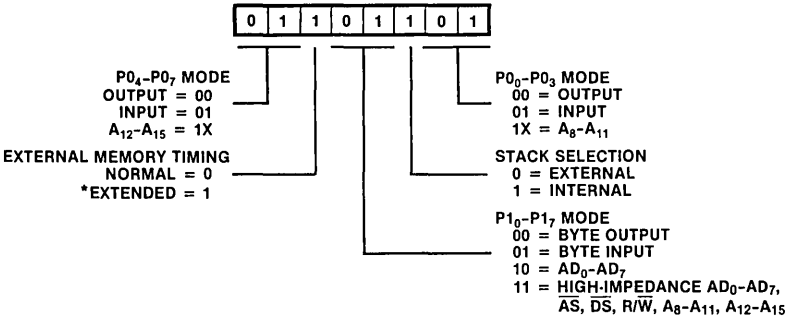


Figure 6-12. Instruction Cycle Timing (Two and Three Byte Instructions)

6.10 RESET CONDITIONS

After a hardware reset, Ports 0 and 1 are configured as input ports, memory and stack are internal, extended timing is set and DM is inactive. Figure 6-13 shows the binary values reset into P01M.

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 6-13. Ports 0 and 1 Reset

Chapter 7
External Interface (Z8681, Z8682)

7.1 INTRODUCTION

The ROMless versions of the Z8 microcomputer have 40 external pins, of which 24 are programmable I/O pins. Of the remaining 16 pins, 8 form an Address/Data bus and the others are used for power and control. Up to 8 I/O pins can be programmed as additional address lines to be used for external memory interface.

7.2 PIN DESCRIPTIONS

$\overline{AS}$. Address Strobe (output, active Low, pin 9). Address Strobe is pulsed Low once at the beginning of each machine cycle. The rising edge of $\overline{AS}$ indicates that addresses, Read/Write (R/$\overline{W}$), and Data Memory ($\overline{DM}$) signals are valid when output for program or data memory transfers.

$\overline{DS}$. Data Strobe (output, active Low, pin 8). Data Strobe provides the timing for data movement to or from Port 1 for each memory transfer. During a Write cycle, data out is valid at the leading edge of $\overline{DS}$. During a Read cycle, data in must be valid prior to the trailing edge of $\overline{DS}$.

R/$\overline{W}$. Read/Write. (output, pin 7). Read/Write determines the direction of data transfer for memory transactions. R/$\overline{W}$ is Low when writing to program or data memory, and High for all other transactions.

$P0_1$-$P0_7$. Address/Data Port (inputs/outputs, TTL-compatible, pins 13-20). Port 1 is permanently configured as a multiplexed Address/Data memory interface. The lower eight address lines (A_0-A_7) are multiplexed with data (D_0-D_7).

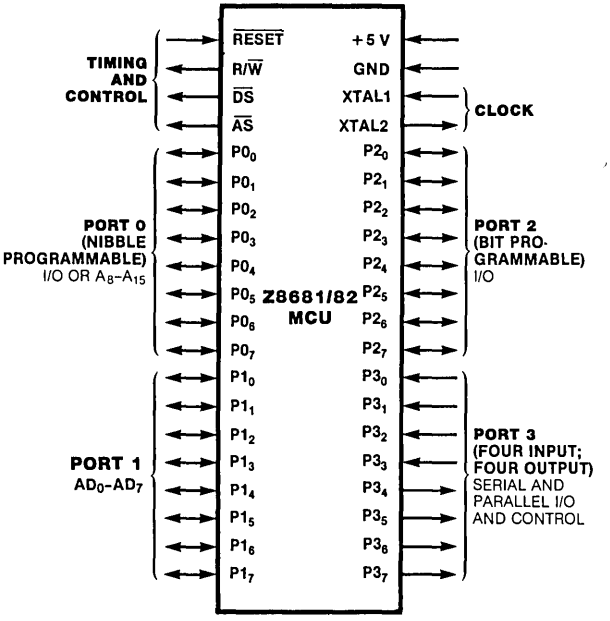


Figure 7-1. Z8681/82 Pin Functions

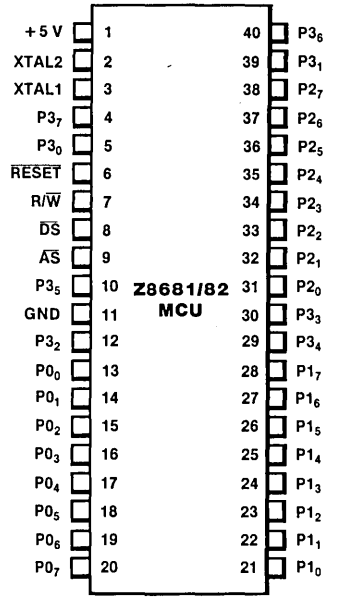


Figure 7-2. Z8681/82 Pin Assignments

$P0_0$-$P0_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. I/O Port Lines (inputs/outputs, TTL-compatible). These 24 I/O lines are divided into 3 8-bit I/O ports that can be configured under program control for I/O or memory interface. Individual lines of a port are denoted by the second digit of the port number. For example, $P3_0$ refers to bit 0 of Port 3.

$\overline{\text{RESET}}$. Reset (input, active Low, pin 6). $\overline{\text{RESET}}$ initializes the Z8681/82. When $\overline{\text{RESET}}$ is deactivated, program execution begins from external program location %C for the Z8681 and location %812 for the Z8682. If held Low, $\overline{\text{RESET}}$ acts as a register file protect during power-down and power-up sequences.

XTAL1, XTAL2. Crystal 1, Crystal 2 (oscillator input and output, pins 3 and 2). These pins connect a parallel resonant crystal or an external source to the on-board clock oscillator and buffer.

7.3 CONFIGURING PORT 0

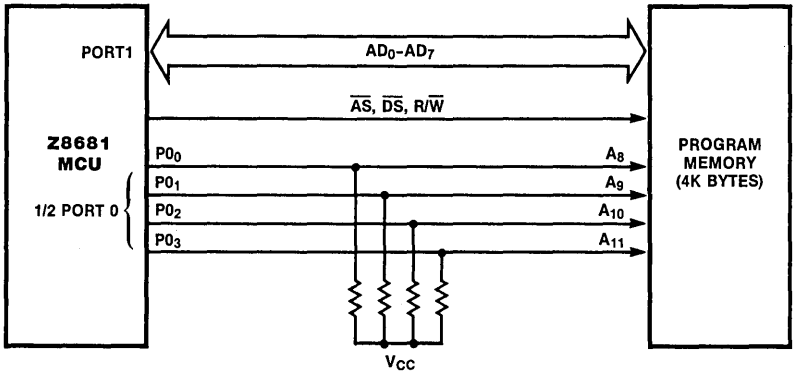
The minimum bus configuration uses Port 1 as a multiplexed Address/Data port (AD_0-AD_7) allowing access to 256 bytes of memory. In this configuration, the eight low order address bits (A_0-A_7) are multiplexed with the data (D_0-D_7).

Port 0 can be programmed to provide either four additional address lines (A_8-A_{11}) which increases the addressable memory to 4K bytes, or eight additional address lines (A_8-A_{15}) which increases the addressable memory to 64K bytes for the Z8681 and 62K bytes for the Z8682. Refer to Chapter 3, Figures 3-5 and 3-6, for the memory maps.

In the Z8681, Port 0 lines intended for use as address lines are automatically configured as inputs after a Reset. These lines therefore float and their logic state remains unknown until an initialization routine configures Port 0. In the Z8682, Port 0 lines are configured as address lines A_8-A_{15} following a Reset.

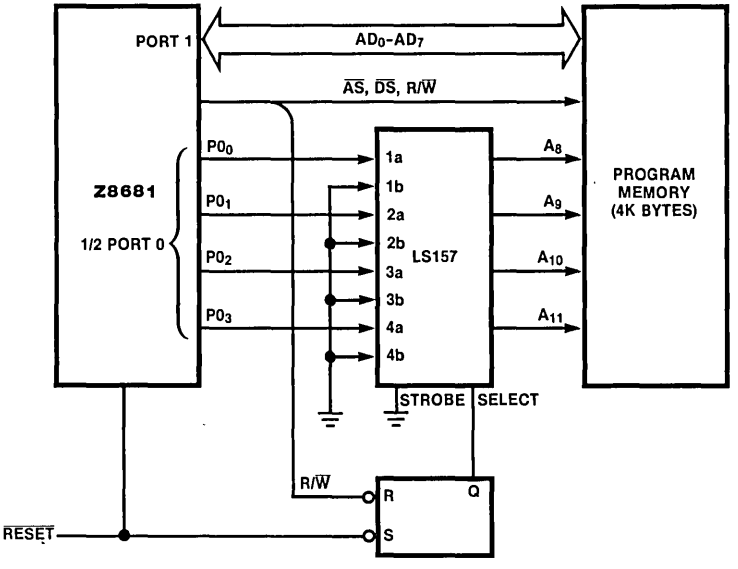
7.3.1 Z8681 Initialization

The initialization routine must reside within the first 256 bytes of executable code and must be physically mapped into memory by forcing the port 0 address lines to a known state. Figures 7-3 and 7-4 illustrate how a 4K byte memory space can be addressed.



The initialization routine is mapped in the top 256 bytes of program memory. Depending on the application, the interrupt vectors may need to be written in the first 12 byte locations of program memory by the initialization routine.

Figure 7-3. Example Z8681/Memory Interface



The initialization routine is mapped in the first 256 bytes of program memory. Any memory write operation will cause the flip-flop to select Port 0 outputs as addresses.

Figure 7-4. Example Z8681/Memory Interface

Port 0 is programmed for memory operation by writing the appropriate bits in the Port 0-1 Mode register (Figure 7-5). The proper port initialization sequence is:

- Load Port 0 with initial address value.
- Configure Port 0-1 Mode register.
- Fetch the next three bytes without changing the address in Port 0. (This is necessary due to instruction pipelining.)

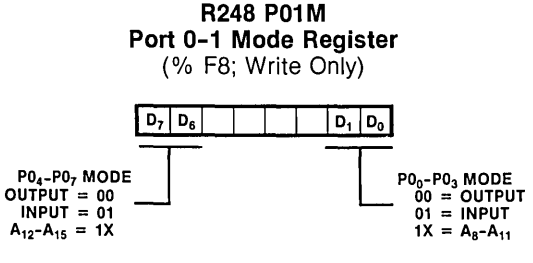


Figure 7-5. Z8681 Port 0 Memory Operation

The lower nibble of Port 0 can be defined as address lines A_8-A_{11}, by setting D_1 to 1. Similarly, setting D_7 to 1 defines the upper nibble of Port 0 as address lines A_{12}-A_{15}.

Whenever Port 0 is configured to output address lines A_{12}-A_{15}, A_8-A_{11} must also be selected as address lines.

7.3.2 Z8682 Initialization

The Z8682 must be operated in Test mode only. Section 8.4 gives a complete description of the proper technique for entering Test mode.

The user initialization routine must begin at location %812 and must reside in memory fast enough for normal memory timing. In the Z8682, the user is not protected from reconfiguring Port 1 by writing to R248 (P01M). Therefore whenever a write is made to P01M, the value 10 (binary) must be written to bits D_4 and D_3. Any other value will cause complete loss of program control.

The lower nibble of Port 0 can be defined as address lines A_8-A_{11}, by setting D_1 to 1. Similarly, setting D_7 to 1 defines the upper nibble of Port 0 as address lines A_{12}-A_{15}.

Whenever Port 0 is configured to output address lines A_{12}-A_{15}, A_8-A_{11} must also by selected as address lines.

7.3.3 Read/Write Operations

If Port 0 is configured for address lines A_7-A_{15}, it can no longer be used as a register; however, if only the lower nibble of Port 0 is defined as address lines A_8-A_{11}, the upper nibble is still addressable as an I/O register. When only the lower nibble is defined as address outputs, reading Port 0 returns XF, where X equals the data in bits D_4-D_7. Writing to Port 0 transfers data to the I/O nibble only.

The instruction used to change the mode of Port 0 should not be immediately followed by an instruction that performs a stack operation, because this will cause indeterminate program flow. In addition, after setting the mode of Port 0 for memory, the next three bytes must be fetched without changing the value of the upper byte of the Program Counter (PC).

7.4 EXTERNAL STACKS

The Z8681/82 architecture supports stack operations in either the register file or data memory. A stack's location is determined by bit D_2 in the Port 0-1 Mode register. For example, if D_2 is set to 0, the stack is in external data memory (Figure 7-7).

The instruction used to change the stack selection bit should not be immediately followed by the instructions RET or IRET, because this will cause indeterminate program flow.

7.5 DATA MEMORY

The two memory spaces, data and program, can be addressed as a single memory space or as two separate spaces of equal size; i.e. 64K bytes each for the Z8681 and 62K bytes each for the Z8682. If the memory spaces are separated, program memory and data memory are logically selected by Data Memory select output ($\overline{DM}$). $\overline{DM}$ is made available on Port 3, line 4 ($P3_4$) by setting bits D_4 and D_3 in the Port 3 Mode register to 10 or 01 (Figure 7-8). $\overline{DM}$ is active Low during the execution of the LDE, LDEI instructions. $\overline{DM}$ is also active Low during the execution of CALL, POP, PUSH, RET and IRET instructions if the stack resides in memory.

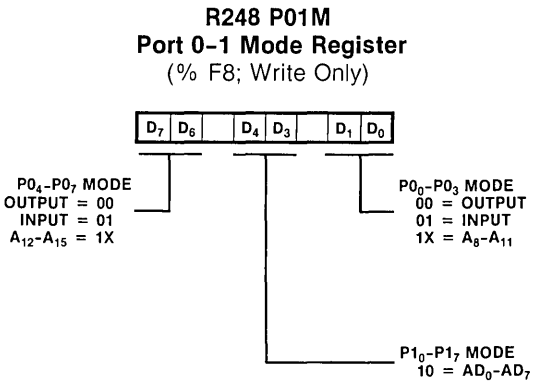


Figure 7-6. Z8682 Port 0 Memory Operation

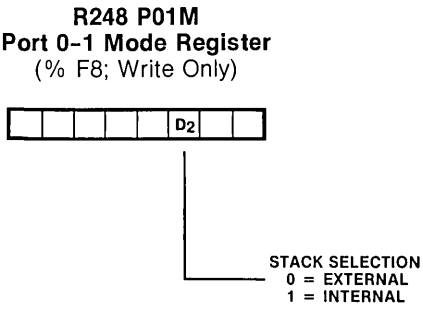


Figure 7-7. External Stack Operation

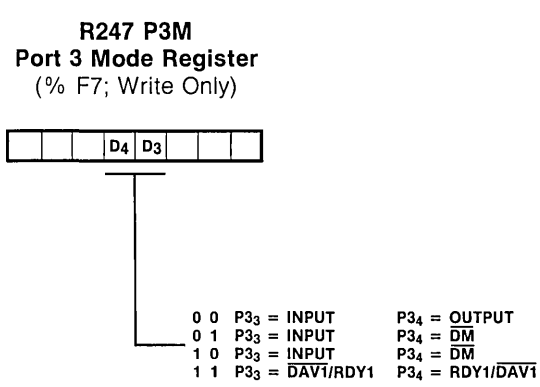


Figure 7-8. Port 3 Data Memory Operation

7.6 BUS OPERATION

Typical data transfers between the Z8681/82 and memory are illustrated in Figure 6-6. Machine cycles can vary from six to twelve clock periods depending on the operation being performed. The notations used to describe the basic timing periods of the Z8681/82 are: machine cycles (Mn), timing states (Tn), and clock periods. All timing references are made with respect to the output signals $\overline{AS}$ and $\overline{DS}$. The clock is shown for clarity only and does not have a specific timing relationship with other signals.

7.6.1 Address Strobe ($\overline{AS}$)

All transactions start with $\overline{AS}$ driven Low and then raised High by the Z8681/82. The rising edge of $\overline{AS}$ indicates that R/$\overline{W}$, $\overline{DM}$ (if used), and the addresses output from Ports 0 and 1 are valid. The addresses output via Port 1 remain valid only during MnT1 and typically need to be latched using $\overline{AS}$, whereas Port 0 address outputs remain stable throughout the machine cycle.

7.6.2 Data Strobe ($\overline{DS}$)

The Z8681/82 uses $\overline{DS}$ to time the actual data transfer. For Write operations (R/$\overline{W}$ = Low), a Low on $\overline{DS}$ indicates that valid data is on the Port 1 AD_0-AD_7 lines. For Read operations (R/$\overline{W}$ = High), the Address/Data bus is placed in a high-impedance state before driving $\overline{DS}$ Low so that the addressed device can put its data on the bus. The Z8681/82 samples this data prior to raising $\overline{DS}$ High.

7.7 EXTENDED BUS TIMING

The Z8681/82 accommodates slow memory access times by automatically inserting an additional software-controlled state time (Tx). This stretches the $\overline{DS}$ timing by two clock periods. Timing is extended by setting bit D_5 in the Port 0-1 Mode register to 1 (Figure 7-9).

Refer to Section 6.7 for other figures pertaining to extended bus timing.

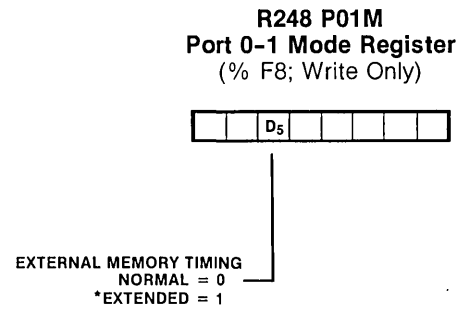


Figure 7-9. Extended Bus Timing

7.8 INSTRUCTION TIMING

The high throughput of the Z8681/82 is due, in part, to the use of instruction pipelining, in which the instruction fetch and execution cycles are overlapped. During the execution of the current instruction the opcode of the next instruction is fetched as illustrated in Figure 6-10.

Figures 6-11 and 6-12 show typical instruction cycle timing for instructions fetched from memory. For those instructions that require execution time longer than that of the overlapped fetch, or reference program or data memory as part of their execution, the pipe must be flushed. In order to calculate the execution time of a program, the internal clock periods shown in the cycles column of the instruction formats in Section 5.6 should be added together. The cycles are equal to one-half the crystal or input clock rate.

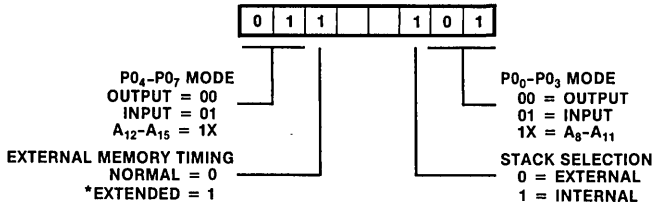
7.9 Z8681 RESET CONDITIONS

After a hardware reset, Port 0 is configured as input port, extended timing is set to accommodate slow memory access during the configuration routine, $\overline{DM}$ is inactive, and the stack resides in the register file. Figure 7-10 shows the binary values reset into P01M.

7.10 Z8682 RESET CONDITIONS

After a hardware reset, Port 0 is configured as address lines A_8-A_{15}, memory timing is normal, $\overline{DM}$ is inactive, and the stack resides in the register file. Figure 7-11 shows the binary values reset into P01M.

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 7-10. Z8681 Port 0 and 1 Reset Conditions

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)

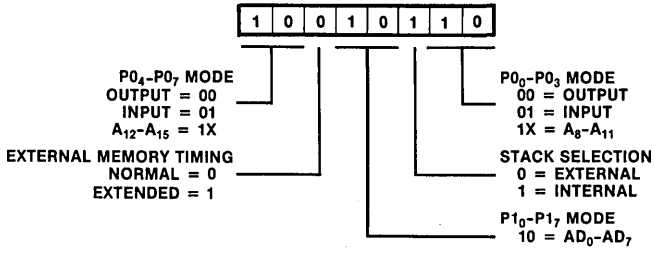


Figure 7-11. Z8682 Ports 0 and 1 Reset Conditions

Chapter 8
Reset and Clock

8.1 RESET

This section describes Z8 reset conditions, reset timing, and register initialization procedures.

A system reset overrides all other operating conditions and puts the Z8 into a known state. To initialize the chip's internal logic, the reset input must be held Low for at least 18 clock periods.

While $\overline{\text{RESET}}$ is Low, $\overline{\text{AS}}$ is output at the internal clock rate (XTAL frequency divided by 2), $\overline{\text{DS}}$ is forced Low and R/$\overline{\text{W}}$ remains High. (Zilog Z-BUS compatible peripherals use the $\overline{\text{AS}}$ and $\overline{\text{DS}}$ coincident Low state as a peripheral reset function.) In addition, interrupts are disabled, Ports 0, 1, and 2 are put in input mode, and %C is loaded into the Program Counter.

The hardware Reset initializes the control and peripheral registers, as shown in Table 8.1. Specific reset values are shown by 1s or 0s, while bits whose states are unknown are indicated by the

Table 8-1. Control and Peripheral Register Reset Values

Register	D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Comments
%F0 Serial I/O	undefined	
%F1 Timer Mode	0 0 0 0 0 0 0 0	Counter/Timers stopped
%F2 Counter/Timer 1	undefined	
%F3 T1 Prescaler	u u u u u u 0 0	Single Pass count mode, external clock source
%F4 Counter/Timer 0	undefined	
%F5 T0 Prescaler	u u u u u u u 0	Single Pass count mode
%F6 Port 2 Mode	1 1 1 1 1 1 1 1	All lines input
%F7 Port 3 Mode	0 0 0 0 0 0 u 0	Port 2 open-drain $P3_0$-$P3_3$ input; $P3_4$-$P3_7$ output
%F8 Port 0-1 Mode Z8601/Z8611	0 1 1 0 1 1 0 1	Ports 0 and 1 inputs; internal stack; extended external memory timing
%F8 Port 0-1 Mode Z8681	0 1 1 1 0 1 0 1	Port 0 inputs Port 1 Address/Data; internal stack; extended external memory timing
%F8 Port 0-1 Mode Z8682	1 0 0 1 0 1 1 0	Port 0 Address Port 1 Address/Data internal stack; normal external memory timing
%F9 Interrupt Priority	undefined	
%FA Interrupt Request	u u 0 0 0 0 0 0	Reset all interrupt disabled
%FB Interrupt Mask	0 u u u u u u u	Interrupts disabled
%FC Flags	undefined	
%FD Register Pointer	undefined	
%FE Stack Pointer	undefined	Most significant byte
%FF Stack Pointer	undefined	Least significant byte

letter u. Registers that are not predictable are listed as undefined.

Program execution starts four clock cycles after $\overline{\text{RESET}}$ has returned High. The initial instruction fetch is from location %C. Figure 8-1 shows reset timing.

After a reset, the first program executed should be a routine that initializes the control registers to the required system configuration. The Interrupt Request register remains inactive until an EI instruction is executed. This guarantees that program execution can proceed free from interrupts.

$\overline{\text{RESET}}$ is the input of a Schmitt trigger circuit. To form the internal reset line, the output of the trigger is synchronized with the internal clock (xtal frequency divided by 2). The clock must therefore be running for $\overline{\text{RESET}}$ to function. For a power-up reset operation, the $\overline{\text{RESET}}$ input must be held Low for at least 50 ms after the power supply is within tolerance. This allows the on-board clock oscillator to stabilize. An internal pull-up combined with an external capacitor of 1 ^{e}F provides enough time to properly reset the Z8 (Figure 8-2).

8.2 CLOCK

The Z8 derives its timing from on-board clock circuitry connected to pins XTAL1 and XTAL2. The clock circuitry consists of an oscillator, a divide-by-2 shaping circuit, and a clock buffer. Figure 8-3 illustrates the clock circuitry. The oscillator's input is XTAL1; its output is XTAL2. The clock can be driven by a crystal, a ceramic resonator, or an external clock source.

Crystals and ceramic resonators should have the following characteristics to ensure proper oscillator operation:

Cut: AT (crystal only)
Mode: Parallel, Fundamental
Output Frequency: 1 MHz - 12 MHz
Resistance: 100 ohms max

Depending on operation frequency, the oscillator may require the addition of capacitors C1 and C2 (shown in Figure 8-4). The range of recommended capacitance values is dependent on crystal specifications but should not exceed 15 pF. The ratio of the values of C1 to C2 can be adjusted to shift the operating frequency of the circuit by approximately ±.005%.

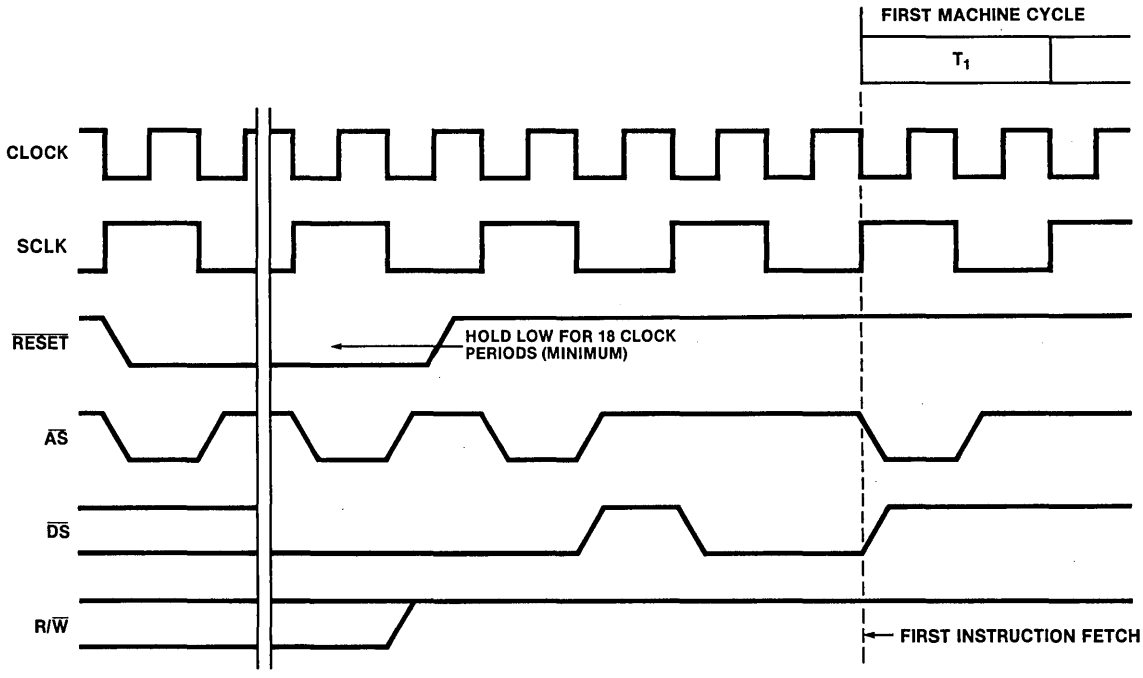


Figure 8-1. Reset Timing

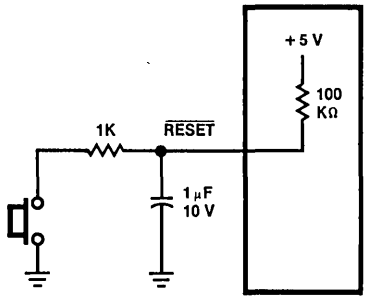


Figure 8-2. Power-Up Reset Circuit

When an external frequency source is used, it must drive both XTAL1 and XTAL2 inputs. This differential drive requirement arises from the loading on the oscillator output (XTAL2) without the reactive feedback network of a crystal or resonator. A typical clock interface circuit is shown in Figure 8-5.

The capacitors shown represent the maximum parasitic loading when using a 74LS04 driver. The pull-up resistors can be eliminated by using a 74HC04 driver.

8.3 POWER-DOWN OPERATION

The Z8 has a power-down option which can be used to maintain the contents of the register file with a low-power battery. The circuitry has its XTAL2 output replaced by a power supply input (V_{MM}). V_{MM} powers the general-purpose registers %04 - %7F as well as a portion of the reset logic that protects the register file. When V_{MM} is maintained at 3 to 5 V, this power-down option preserves the contents of the general-purpose registers whenever V_{CC} is removed. During normal operation, V_{MM} provides +5 V along with V_{CC}.

The following sequence is necessary to preserve data:

- Power failure must be externally detected early enough for a software routine to store the required data that is not already in the register file. An interrupt is typically used for this purpose.
- $\overline{\text{RESET}}$ must be held Low after data is saved and during the removal of V_{CC}. $\overline{\text{RESET}}$ is a write protect input to the register file.

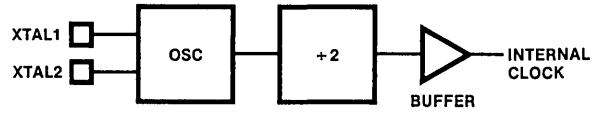


Figure 8-3. Z8 Clock Circuit

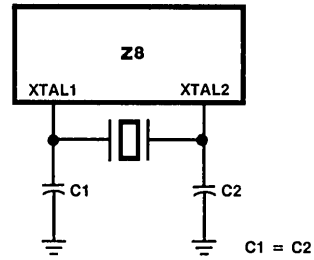


Figure 8-4. Crystal/Ceramic Resonator Oscillator

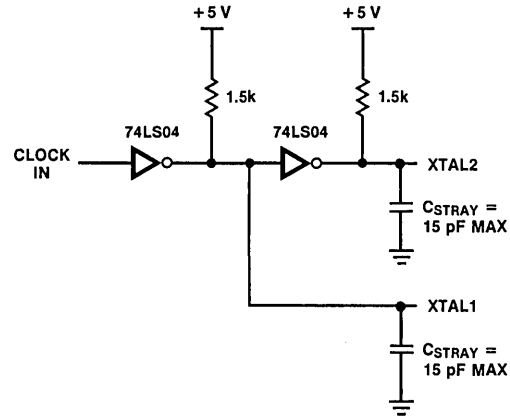


Figure 8-5. External Clock Interface

- $\overline{\text{RESET}}$ must be held Low during the power-up sequence. Again, $\overline{\text{RESET}}$ is a write protect input to the register file.

As V_{CC} powers down, on-board circuitry associated with $\overline{\text{RESET}}$ automatically protects the general-purpose registers. The circuit shown in Figure 8-2 satisfies the power-up requirement of holding $\overline{\text{RESET}}$ Low to protect the register file data.

Figure 8-6 shows the recommended circuit configuration for a battery-backed supply system.

Since XTAL2 is replaced by V_{MM}, an external clock generator must be used to input the Z8 clock via the XTAL1 input.

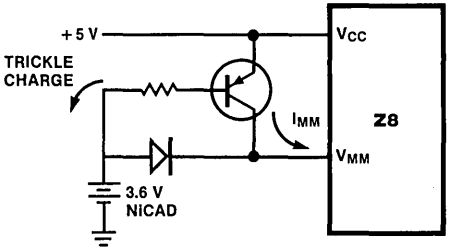


Figure 8-6. Battery-Backed Register Supply

8.4 TEST MODE

Test mode is a special mode of operation that facilitates testing of Z8 devices containing on-board ROM (Z8601 and Z8611). Test mode must also be used to reset the Z8682. When Test mode is invoked, an additional on-board ROM is mapped into the first 64 locations of program memory. Figure 8-7 shows the difference between Normal and Test modes of operation.

Test mode is entered by driving the $\overline{\text{RESET}}$ input to a voltage level of V_{CC} + 2.5 V after a normal Reset cycle (Figure 8-8). This voltage is absolutely essential for proper operation.

After entering Test mode, instructions are fetched from the internal test ROM. Port 1 is configured for Address/Data operation, followed by a JUMP to external memory location %812 for the Z8601 and Z8682, or %1012 for the Z8611. Once in external memory, diagnostic routines, invoked via the

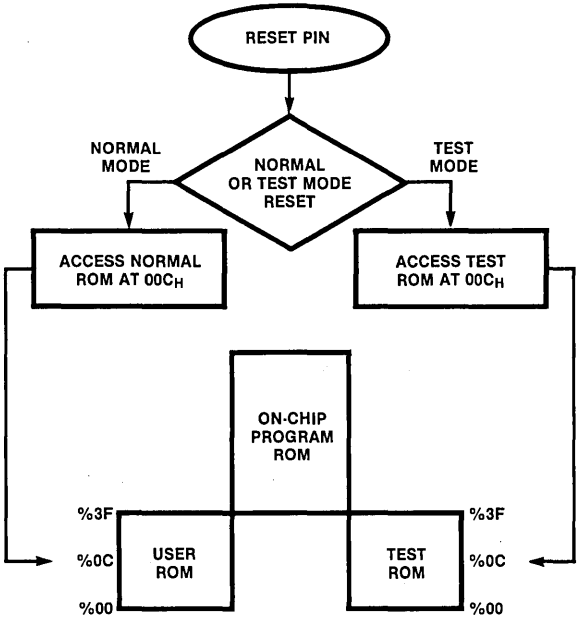


Figure 8-7. Normal and Test Mode Flow

Address/Data bus, verify the Z8's functionality. Since Port 1 is used only in Address/Data mode in this process, additional routines in the test ROM verify Port 1's I/O and Handshake modes.

Programs run with Test mode active can use the LDE instruction to access contents of the test ROM. The LDC instruction accesses the normal program ROM.

The Z8 stays in the Test mode until a normal reset occurs.

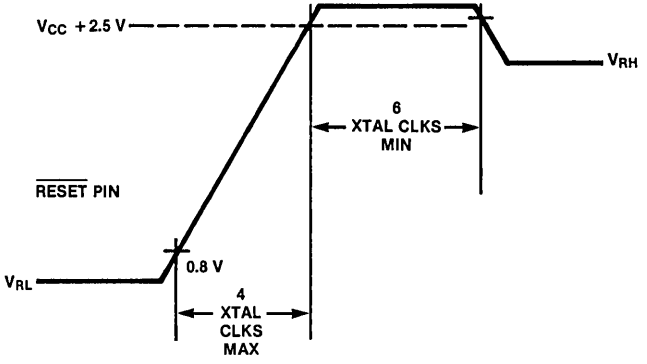
8.4.1 Interrupt Testing

To test the interrupt structure, the first twelve locations of test ROM contain interrupt vectors. Interrupt vectors in the Z8601 and Z8682 point to external memory locations %800, %803, %806, %809, %80C, and %80F; interrupt vectors in the Z8611 point to external memory locations %1000, %1003, %1006, %1008. %100C, and %100F. These interrupt vectors allow the external program to have a 2- or 3-byte JUMP instruction to each interrupt service routine.

Programs that are run with Test mode active can use the LDE instruction for accessing the contents of the Test ROM. The LDC instruction can be used for accessing the program ROM as normal.

8.4.2 ROMless Operation

ROMless operation of the Z8601 or Z8611 can be achieved by always entering Test mode after a reset. Execution begins at %812 or %1012, respectively. (The Z8682 is a modified Z8601 sold as a ROMless part.)



Note the maximum ramp for application of +7.5 VDC to $\overline{\text{RESET}}$ pin. After a minimum of 6 XTAL CLK cycles, the $\overline{\text{RESET}}$ voltage can be relaxed to VRH.

Figure 8-8. Voltage Waveform for Test Mode

Chapter 9
I/O Ports

9.1 INTRODUCTION

The Z8 has 32 lines dedicated to input and output. These lines are grouped into four 8-bit ports and are configurable as input, output, or address/data. Under software control, the ports can be programmed to provide address/data, timing, status, serial, and parallel input/output with or without handshake.

All ports have active pull-ups and pull-downs compatible with TTL loads. In addition, the pull-ups of Port 2 can be turned off for open-drain operation.

9.1.1 Mode Registers

Each port has an associated mode register which determines the port's functions and allows dynamic change in port functions during program execution. Ports and mode registers are mapped into the register file as shown in Figure 9-1.

Because of their close association, ports and mode registers are treated like any other general-purpose register. There are no special instructions for port manipulation; any instruction that addresses a register can address the ports. Data can be directly accessed in the port register, with no extra moves.

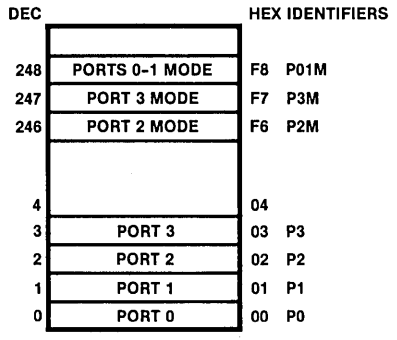


Figure 9-1. I/O Port and Port Mode Registers

9.1.2 Input and Output Registers

Each bit of Ports 0, 1, and 2 has an input register, an output register, associated buffer, and control logic. Since there are separate input and output registers associated with each port, writing to bits defined as inputs stores the data in the output register. This data cannot be read as long as the bits are defined as inputs. However, if the bits are reconfigured as output, the data stored in the output register is reflected on the output pins and can then be read. This mechanism allows the user to initialize the outputs prior to driving their loads.

Since port inputs are asynchronous to the Z8's internal clock, a Read operation could occur during an input transition. In this case, the logic level might be uncertain--somewhere between a logic 1 and 0. To eliminate this meta-stable condition, the Z8 latches the input data two clock periods prior to the execution of the current instruction. The input register uses these two clock periods to stabilize to a legitimate logic level before the instruction reads the data.

9.2 PORT 0

This section deals only with the I/O operation of Port 0. Refer to Sections 6.2 and 7.2 for a description of the port's external memory interface operation.

Port 0 is a general I/O port. Bits within each nibble can be independently programmed as inputs, outputs or address lines. Figure 9-2 shows a block diagram of Port 0. This diagram also applies to Ports 1 and 2.

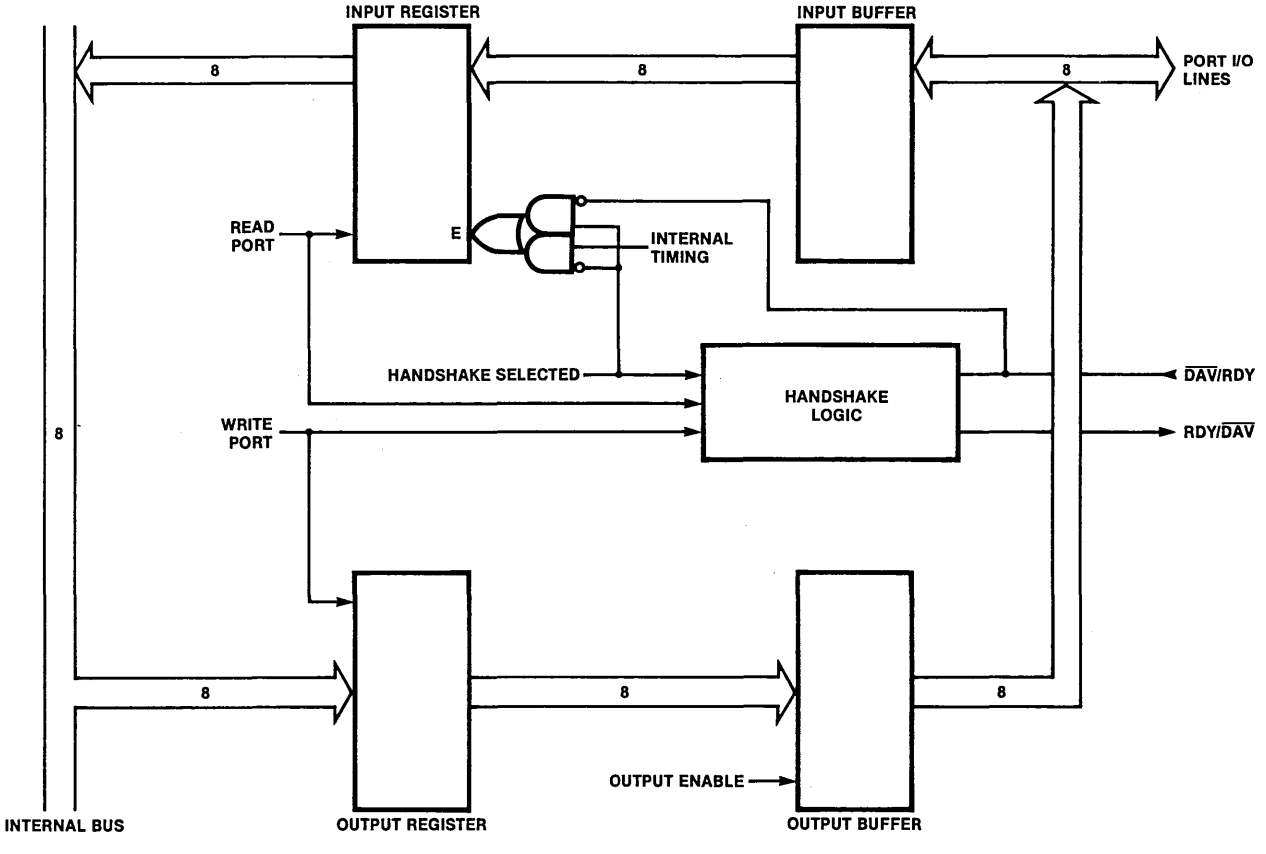


Figure 9-2. Ports 0, 1, and 2 Block Diagram

9.2.1 Read/Write Operations

In the nibble I/O mode, Port 0 is accessed as general-purpose register P0 (%00). The port is written by specifying P0 as an instruction's destination register. Writing the port causes data to be stored in the port's output register.

The port is read by specifying P0 as the source register of an instruction. When an output nibble is read, data on the external pins is returned. Under normal loading conditions this is equivalent to reading the output register. Reading a nibble defined as input also returns data on the external pins. However, input bits under handshake control return data latched into the input register via the input strobe.

The Port 0-1 Mode register bits D_1D_0 and D_7D_6 are used to configure Port 0 nibbles (Figure 9-3). The lower nibble ($P0_0$-$P0_3$) can be defined as inputs by setting bits D_1 to 0 and D_0 to 1, or as outputs by setting both D_1 and D_0 to 0. Likewise, the upper nibble ($P0_4$-$P0_7$) can be defined as inputs by setting bits D_7 to 0 and D_6 to 1, or as outputs by setting both D_6 and D_7 to 0.

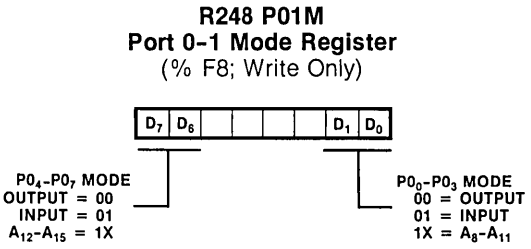


Figure 9-3. Port 0 I/O Operation

9.2.2 Handshake Operation

When used as an I/O port, Port 0 can be placed under handshake control by programming the Port 3 Mode register bit D_2 to 1 (Figure 9-4). In this configuration, handshake control lines are $\overline{DAV_0}$ ($P3_2$) and RDY_0 ($P3_5$) when Port 0 is an input port, or RDY_0 ($P3_2$) and $\overline{DAV_0}$ ($P3_5$) when Port 0 is an output port.

Handshake direction is determined by the configuration (input or output) assigned to Port 0's upper nibble, $P0_4$-$P0_7$. The lower nibble must have the same I/O configuration as the upper nibble to be under handshake control. Figure 9-5 illustrates the Port 0 upper and lower nibbles, and the associated handshake lines of Port 3.

Handshake operation is discussed in detail in Section 9.6.

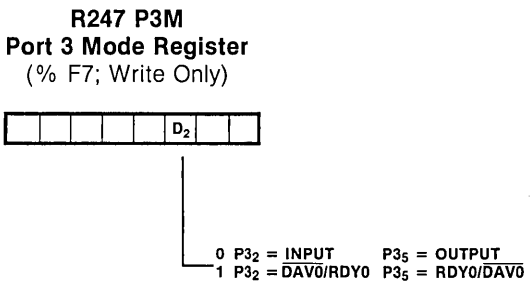


Figure 9-4. Port 0 Handshake Operation

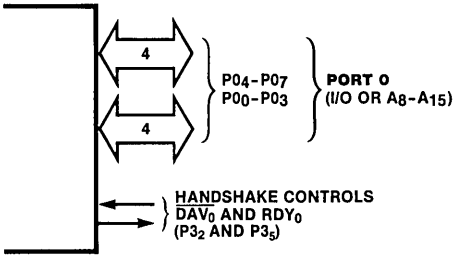


Figure 9-5. Port 0

9.3 PORT 1

This section deals only with the I/O operation of Port 1 and does not apply to the Z8681/82 ROMless devices. Refer to Sections 6.2 and 7.2 for a description of the port's external memory interface operation.

Port 1 is a general-purpose I/O port that can be programmed as a byte I/O port with or without handshake, or as an address/data port for interfacing with external memory. Refer to Figure 9-2 for a block diagram of Port 1.

9.3.1 Read/Write Operations

In byte input or byte output mode, the port is accessed as general-purpose register P1 (%01). The port is written by specifying P1 as an instruction's destination register. Writing the port causes data to be stored in the port's output register.

The port is read by specifying P1 as the source register of an instruction. When an output is read, data on the external pins is returned. Under normal loading conditions, this is equivalent to reading the output register. When Port 1 is defined as an input, reading also returns data on the external pins. However, inputs under handshake control return data latched into the input register via the input strobe.

Using the Port 0-1 Mode register, Port 1 is configured as an output port by setting bits D_4 and D_3 to 0s, or as an input port by setting D_4 to 0 and D_3 to 1 (Figure 9-6).

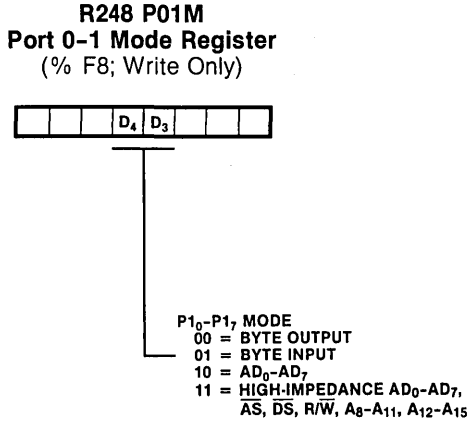


Figure 9-6. Port 1 I/O Operation

9.3.2 Handshake Operations

When used as an I/O port, Port 1 can be placed under handshake control by programming the Port 3 Mode register bits D_4 and D_3 both to 1 (Figure 9-7). In this configuration, handshake control lines are $\overline{DAV_1}$ ($P3_3$) and RDY_1 ($P3_4$) when Port 1 is an input port, or RDY_1 ($P3_3$) and $\overline{DAV_1}$ ($P3_4$) when Port 1 is an output port.

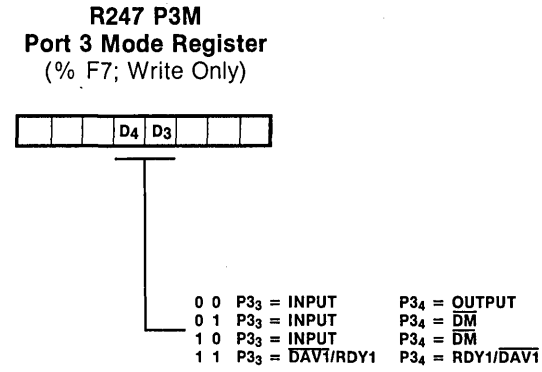


Figure 9-7. Port 1 Handshake Operation

Handshake direction is determined by the configuration (input or output) assigned to Port 1. For example, if Port 1 is an output port then handshake is defined as output. Figure 9-8 illustrates the Port 1 lines and the associated handshake lines of Port 3.

Handshake operation is discussed in detail in Section 9.6.

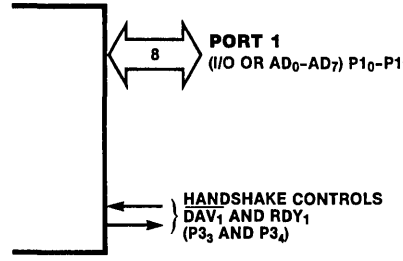


Figure 9-8. Port 1

9.4 PORT 2

Port 2 is a general-purpose port. Each of its lines can be independently programmed as input or output via the Port 2 Mode register (Figure 9-9). A bit set to a 1 in P2M configures the corresponding bit in Port 2 as an input, while a bit set to 0 determines an output line.

R246 P2M
Port 2 Mode Register
(%F6; Write Only)

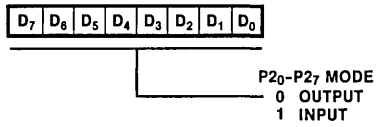


Figure 9-9. Port 2 I/O Operation

9.4.1 Read/Write Operations

Port 2 is accessed as general-purpose register P2 (%02). The port is written by specifying P2 as an instruction's destination register. Writing the port causes data to be stored in the port's output register, and reflected externally on any bit configured as an output.

The port is read by specifying P2 as the source register of an instruction. When an output bit is read, data on the external pin is returned. Under normal loading conditions, this is equivalent to reading the output register. However, if a bit of Port 2 is defined as an open-drain output, the data returned is the value forced on the output pin by the external system. This may not be the same as the data in the output register.

Reading input bits of Port 2 also returns data on the external pins. However, inputs under handshake control return data latched into the input register via the input strobe.

9.4.2 Handshake Operation

Port 2 can be placed under handshake control by programming the Port 3 Mode register (Figure 9-10). In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake control lines $\overline{DAV}_2$ and RDY_2 for input handshake, or RDY_2 and $\overline{DAV}_2$ for output handshake.

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

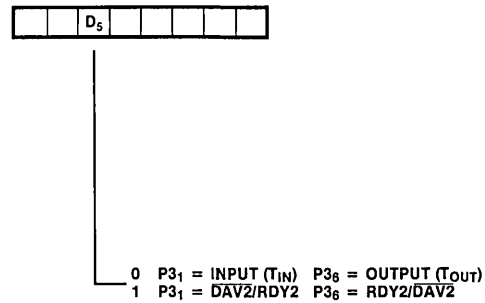


Figure 9-10. Port 3 Handshake Operation

Handshake direction is determined by the configuration (input or output) assigned to bit 7 of Port 2. Only those bits with the same configuration as $P2_7$ will be under handshake control. Figure 9-11 illustrates Port 2's bit lines and the associated handshake lines of Port 3.

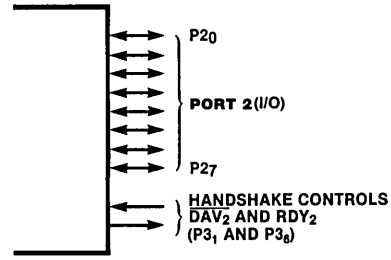


Figure 9-11. Port 2

Port 2 can also by configured to provide open-drain outputs by programming Port 3 Mode register (P3M) bit D_0 to 0 (Figure 9-12).

Regardless of the bit input/output configuration, Port 2 is always written and read as a byte-wide port.

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

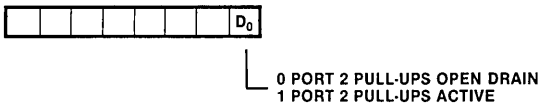


Figure 9-12. Port 2 Open-Drain Outputs

9.5 PORT 3

Port 3 differs structurally from the other three ports. Port 3 lines are fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$) and do not have an input and output register for each bit. Instead, all the input lines have one input register, and output lines have an output register. Under software control, the lines can be configured as input or output, special control lines for handshake, or as I/O lines for the on-board serial and timer facilities. Figure 9-13 is a block diagram of Port 3.

9.5.1 Read/Write Operations

Port 3 is accessed as general-purpose register P3 (%03). The port is written by specifying P3 as an instruction's destination register. However,

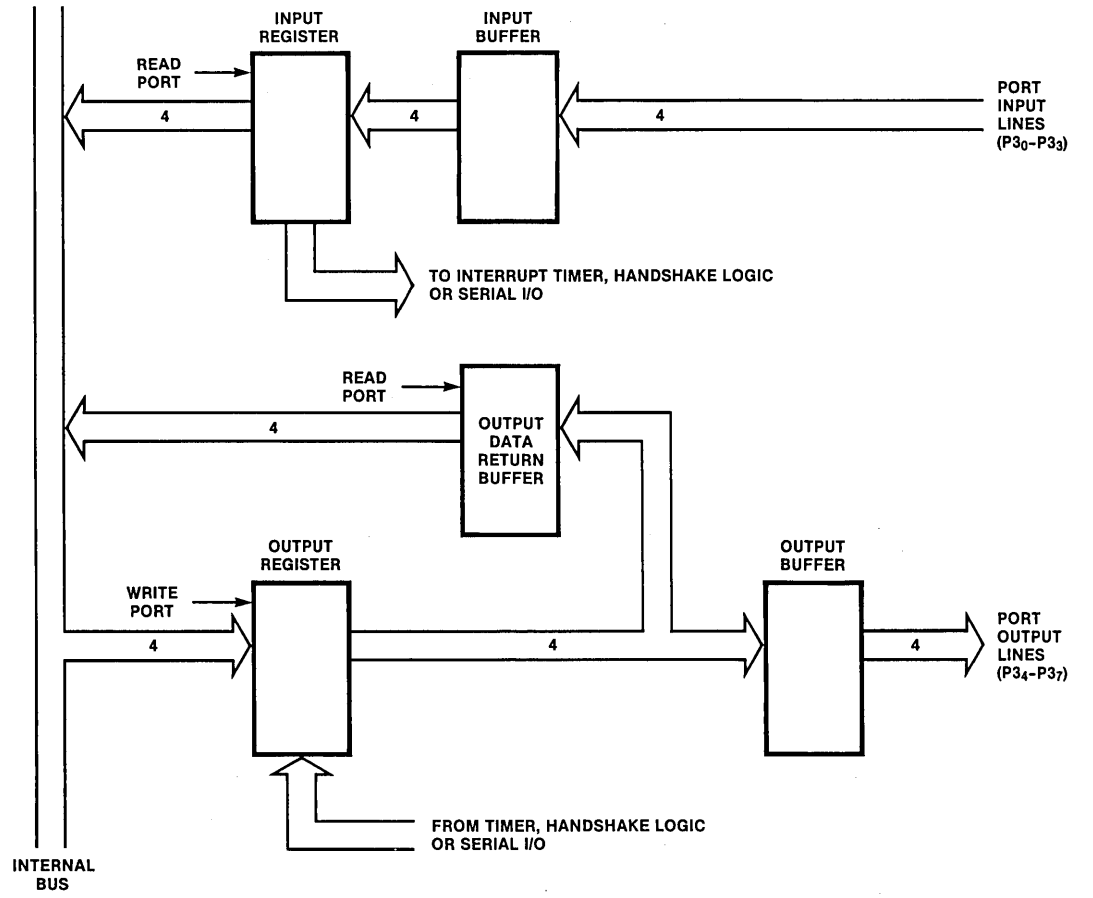


Figure 9-13. Port 3 Block Diagram

Port 3 outputs cannot be written if they are used for special functions. When writing to Port 3, data is stored in the output register.

The port is read by specifying P3 as the source register of an instruction. When reading from Port 3, the data returned is both the data on the input pins and in the output register.

9.5.2 Special Functions

Special functions for Port 3 are defined by programming the Port 3 Mode register. By writing 0s in D_2-D_6, lines $P3_0$-$P3_7$ ar configured in input/output pairs (Figure 9-14). Table 9-1 shows available functions for Port 3. The special functions indicated in the table are discussed in detail in their corresponding sections in this manual.

Port 3 input lines $P3_0$-$P3_3$ always function as interrupt requests regardless of the configuration specified in the Port 3 Mode register. Unwanted interrupts must be masked off as described in Chapter 10.

Table 9.1 Port 3 Line Functions

Function	Line	Signal
Input	$P3_0$-$P3_3$	Input
Output	$P3_4$-$P3_7$	Output
Handshake Inputs	$P3_1$	$\overline{DAV}_2/RDY_2$
	$P3_2$	$\overline{DAV}_0/RDY_0$
	$P3_3$	$\overline{DAV}_1/RDY_1$
Handshake Outputs	$P3_4$	$RDY_1/\overline{DAV}_1$
	$P3_5$	$RDY_0/\overline{DAV}_0$
	$P3_6$	$RDY_2/\overline{DAV}_2$
Interrupt Requests	$P3_0$	IRQ_3
	$P3_1$	IRQ_2
	$P3_2$	IRQ_0
	$P3_3$	IRQ_1
Serial Input	$P3_0$	SI
Output	$P3_7$	SO
Counter/Timer	$P3_1$	T_{in}
	$P3_6$	T_{out}
Status	$P3_4$	$\overline{DM}$

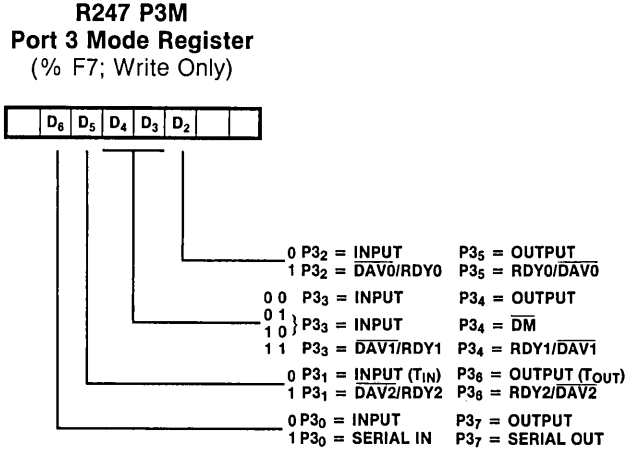


Figure 9-14. Port 3 I/O Operation

9.6 PORT HANDSHAKE

When Ports 0, 1, or 2 are configured for handshake operation, a pair of lines from Port 3 is used for handshake controls for each port. The handshake controls are interlocked to properly time asynchronous data transfers between the Z8 and its peripheral. One control line ($\overline{DAV}_n$) functions as a strobe from the sender to indicate to the receiver that data is available. The second control line (RDY_n) acknowledges receipt of the sender's data, and indicates when the receiver is ready to accept another data transfer.

In the input mode, data is latched into the port's input register by the first $\overline{DAV}$ signal, and is protected from being overwritten if additional pulses occur on the $\overline{DAV}$ line. This overwrite protection is maintained until the port data is read. In the output mode, data written to the port is not protected and can be overwritten by the Z8 during the handshake sequence. To avoid losing data, the software must not overwrite the port until the corresponding interrupt request indicates that the external device has latched the data.

The software can always read Port 3 output and input handshake lines, but cannot write to the output handshake lines.

Following is the recommended setup sequence when configuring a port for handshake operation for the first time after a reset:

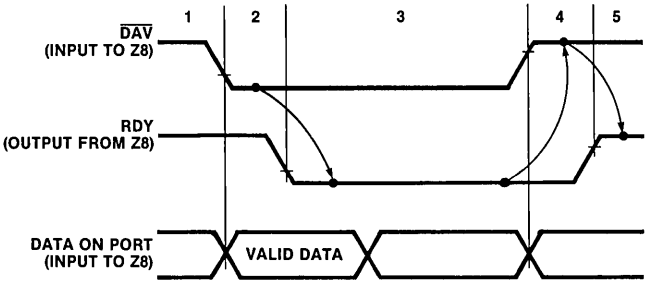
- Load P01M or P2M to configure the port for input/output.
- Load P3 to set the Output Handshake bit to a logic 1.
- Load P3M to select the Handshake mode for the port.

Once a data transfer begins, the configuration of the handshake lines should not be changed until handshake is completed.

Figures 9-15 and 9-16 show detailed operation for the handshake sequence.

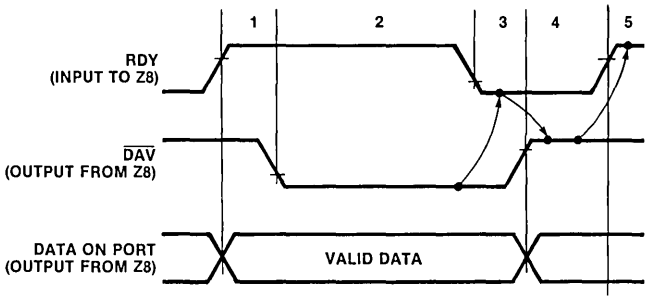
In applications requiring a strobed signal instead of the interlocked handshake, the Z8 can satisfy this requirement as follows:

- In the Strobed Input mode, data can be latched in the port input register using the $\overline{DAV}$ input. The data transfer rate must allow enough time for the software to read the port before strobing in the next character. The RDY output is ignored.
- In the Strobed Output mode, the RDY input should be tied to the $\overline{DAV}$ output.



State 1. Port 3 Ready output is High, indicating that the Z8 is ready to accept data.
State 2. The I/O device puts data on the port and then activates the $\overline{DAV}$ input. This causes the data to be latched into the port input register and generates an interrupt request.
State 3. The Z8 forces the Ready (RDY) output Low, signaling to the I/O device that the data has been latched.
State 4. The I/O device returns the $\overline{DAV}$ line High in response to RDY going Low.
State 5. The Z8 software must respond to the interrupt request and read the contents of the port in order for the handshake sequence to be completed. The RDY line goes High if and only if the port has not been read and $\overline{DAV}$ is High. This returns the interface to its initial state.

Figure 9-15. Z8 Input Handshake



State 1. RDY input is High indicating that the I/O device is ready to accept data.
State 2. The Z8 writes to the port register to initiate a data transfer. Writing the port outputs new data and forces $\overline{DAV}$ Low if and only if RDY is High.
State 3. The I/O device forces RDY Low after latching the data. RDY Low causes an interrupt request to be generated. The Z8 can write new data in response to RDY going Low; however, the data is not output until State 5.
State 4. The $\overline{DAV}$ output from the Z8 is driven High in response to RDY going Low.
State 5. After $\overline{DAV}$ goes High, the I/O device is free to raise RDY High thus returning the interface to its initial state.

Figure 9-16. Z8 Output Handshake

Figures 9-17 and 9-18 illustrate the strobed handshake connections.

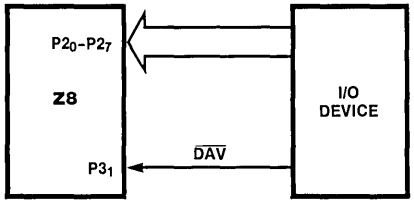


Figure 9-17. Input Strobed Handshake using Port 2

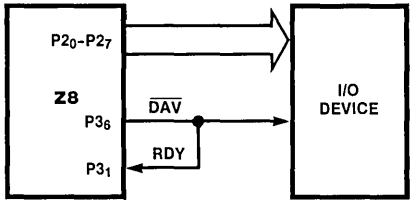


Figure 9-18. Output Strobed Handshake using Port 2

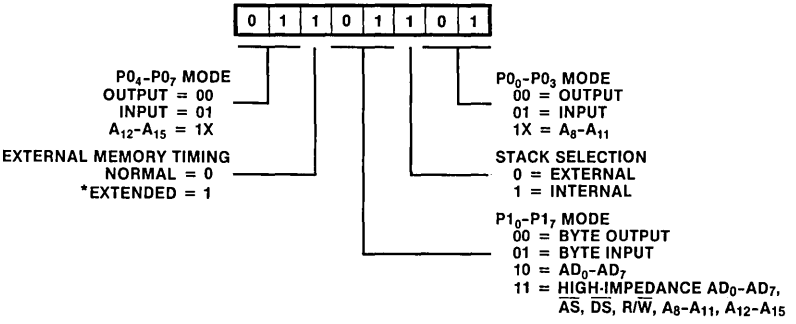
9.7 I/O PORT RESET CONDITIONS

After a hardware reset, mode registers P01M, P2M, and P3M are set as shown in Figures 9-19 - 9-22. Ports 0, 1 and 2 are configured for input operation on all bits, except Port 1 in the Z8681 and Ports 0 and 1 in the Z8682 as shown.

The pull-ups of Port 2 are set for open-drain. If active pull-ups are desired for Port 3 outputs, remember to configure them using P3M (Figure 9-22).

All special I/O functions of Port 3 are inactive, with $P3_0$-$P3_3$ set as inputs and $P3_4$-$P3_7$ set as outputs (Figure 9-23).

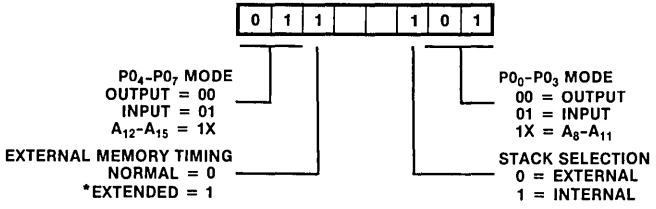
R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 9-19. Z8601/11 Port 0 and 1 Reset

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 9-20. Z8681 Ports 0 and 1 Reset

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)

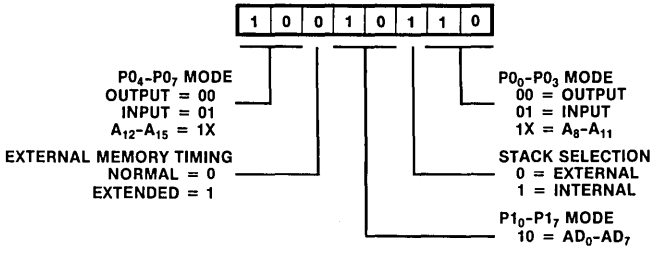


Figure 9-21. Z8682 Ports 0 and 1 Reset

R246 P2M
Port 2 Mode Register
(% F6; Write Only)

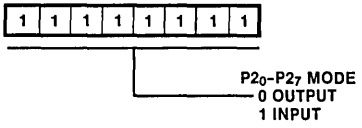


Figure 9-22. Port 2 Reset

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

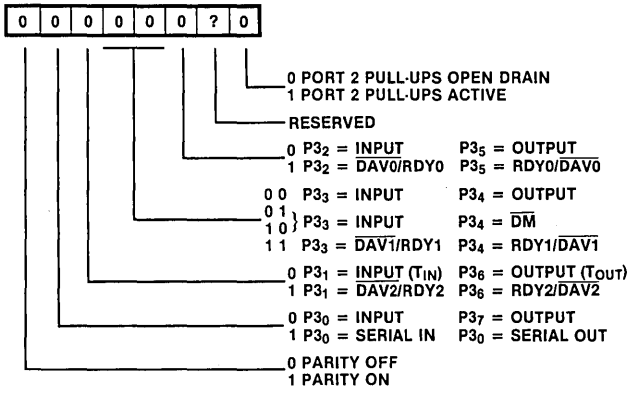


Figure 9-23. Port 3 Reset

Chapter 10
Interrupts

10.1 INTRODUCTION

The Z8 microcomputer allows six different interrupt levels from eight sources: the four Port 3 lines $P3_0$-$P3_3$ make up the external interrupt sources while serial in, serial out, and the two counter/timers make up the internal sources. These interrupts can be masked and their priorities set by using the Interrupt Mask and the Interrupt Priority registers. All six interrupts can be globally disabled by resetting the master Interrupt Enable bit D_7 in the Interrupt Mask register with a Disable Interrupt (DI) instruction. Interrupts are globally enabled by setting D_7 with an Enable Interrupt (EI) instruction.

There are three interrupt control registers: the Interrupt Request register (IRQ), the Interrupt Mask register (IMR), and the Interrupt Priority register (IPR). Figure 10-1 shows addresses and identifiers for the interrupt control registers. Figure 10-2 is a block diagram showing the Interrupt Mask and Interrupt Priority logic.

The Z8 family supports both vectored and polled interrupt handling. Details on vectored and polled interrupts can be found in Sections 10.6 and 10.7.

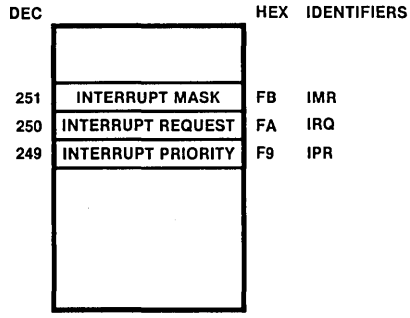


Figure 10-1. Interrupt Control Registers

10.2 INTERRUPT SOURCES

Table 10-1 presents the interrupt types, sources, and vectors available in the Z8 family of processors.

10.2.1 External Interrupt Sources

External sources involve interrupts request lines IRQ_0-IRQ_3. IRQ_0, IRQ_1, and IRQ_2 are always generated by a negative edge signal on the corresponding Port 3 pin ($P3_2$, $P3_3$, $P3_1$ correspond to IRQ_0, IRQ_1, and IRQ_2, respectively). Figure 10-3 is a block diagram for interrupt sources IRQ_0, IRQ_1, and IRQ_2.

When the Port 3 pin ($P3_1$, $P3_2$, or $P3_3$) goes Low, the first flip-flop is set. The next two flip-flops synchronize the request to the internal clock and delay it by four external clock periods. The output of the last flip-flop (IRQ_0, IRQ_1, or IRQ_3) goes to the corresponding Interrupt Request register.

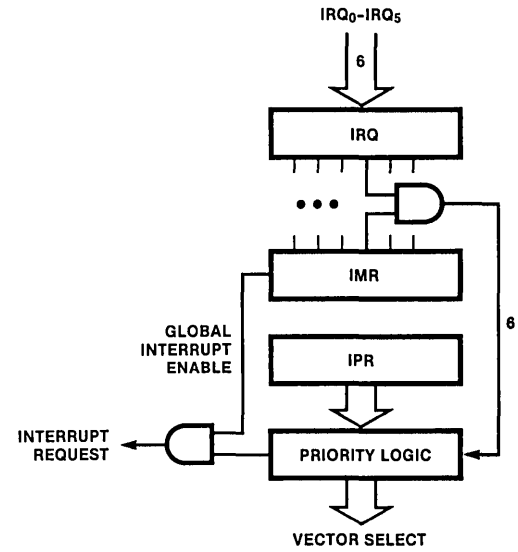


Figure 10-2. Interrupt Block Diagram

Table 10-1.
Interrupt Types, Sources, and Vectors

Name	Source	Vector Location	Comments
IRQ_0	$\overline{DAV}_0$, IRQ_0	0,1	External ($P3_2$), ↓ Edge Triggered
IRQ_1	$\overline{DAV}_1$, IRQ_1	2,3	External ($P3_3$), ↓ Edge Triggered
IRQ_2	$\overline{DAV}_2$, IRQ_2, T_{IN}	4,5	External ($P3_1$), ↓ Edge Triggered
IRQ_3	IRQ_3	6,7	External ($P3_0$), ↓ Edge Triggered
	Serial In	6,7	Internal
IRQ_4	T_0	8,9	Internal
	Serial Out	8,9	Internal
IRQ_5	T_1	10,11	Internal

IRQ_3 can be generated from an external source only if Serial In is not enabled; otherwise, its source is internal. The external request is generated by a negative edge signal on $P3_0$ as shown in Figure 10-4. Again, the external request is synchronized and delayed before reaching IRQ.

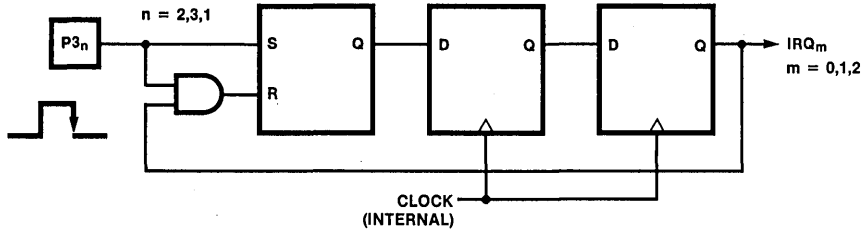


Figure 10-3. Interrupt Sources IRQ_0-IRQ_2 Block Diagram

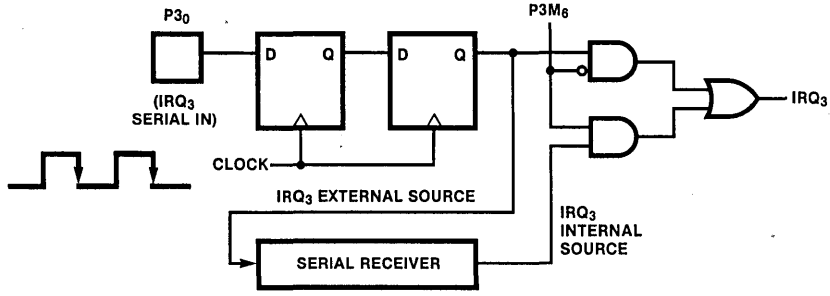


Figure 10-4. Interrupt Source IRQ_3 Block Diagram

10.2.2 Internal Interrupt Sources

Internal sources involve interrupt requests IRQ_3-IRQ_5. If Serial In is enabled, IRQ_3 generates an interrupt request whenever the receiver assembles a complete byte. Interrupt level IRQ_4 has two mutually exclusive sources, Counter/Timer 0 (T_0) and the Serial Out transmitter. If Serial Out is enabled, an interrupt request is generated when the transmit buffer is empty. If T_0 is enabled, an interrupt request is generated at T_0 end-of-count. IRQ_5 generates an interrupt request at Counter/Timer 1's (T_1) end-of-count.

For more details on the internal interrupt sources, refer to the chapters describing serial I/O and the counter/timers.

10.3 INTERRUPT REQUEST (IRQ) REGISTER LOGIC AND TIMING

Figure 10-5 shows the logic diagram for the Interrupt Request register. The leading edge of the request will set the first flip-flop, which will remain set until interrupt requests are sampled.

Requests are sampled internally during the last clock cycle before an opcode fetch (Figure 10-6). External requests are sampled two internal clocks earlier, due to the synchronizing flip-flops shown in Figures 10-3 and 10-4.

At sample time the request is transferred to the second flip-flop in Figure 10-5, which drives the interrupt mask and priority logic. When an interrupt cycle occurs, this flip-flop will be reset only for the highest priority level that is enabled.

The user has direct access to the second flip-flop by reading and writing the IRQ register. IRQ is read by specifying it as the source register of an instruction and written by specifying it as the destination register.

10.4 INTERRUPT INITIALIZATION

After reset, all interrupts are disabled and must be initialized before vectored or polled interrupt processing can begin. The Interrupt Priority register (IPR), Interrupt Mask register (IMR) and Interrupt Request register (IRQ) must be initialized, in that order, to start the interrupt process. However, IPR need not be initialized for polled processing.

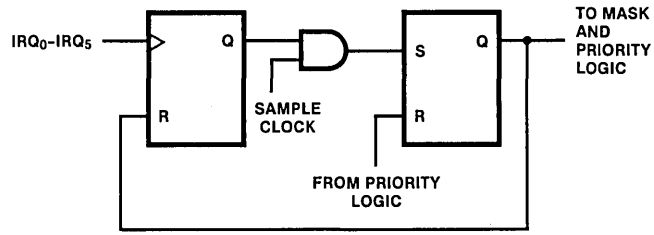


Figure 10-5. IRQ Register Logic

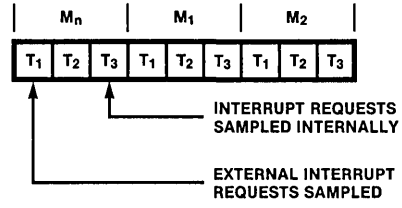


Figure 10-6. Interrupt Request Timing

10.4.1 Interrupt Priority Register (IPR) Initialization

IPR (Figure 10-7) is a write-only register that sets priorities for the six levels of vectored interrupts in order to resolve simultaneous interrupt requests. (There are 48 sequence possibilities for interrupts.) The six interrupt levels IRQ_0-IRQ_5 are divided into three groups of two interrupt requests each. One group contains IRQ_3 ($SI/P3_0$) and IRQ_5 (T_1), another group contains IRQ_0 ($P3_2$) and IRQ_2 ($P3_1$), and the third group contains IRQ_1 ($P3_3$) and IRQ_4 (SO/T_0).

Priorities can be set both within and between groups as shown in Table 10-2. Bits D_1, D_2, and D_5 define the priority of the individual members within the three groups. Bits D_0, D_3, and D_4 are encoded to define six priority orders between the three groups. Bits D_6 and D_7 are not used.

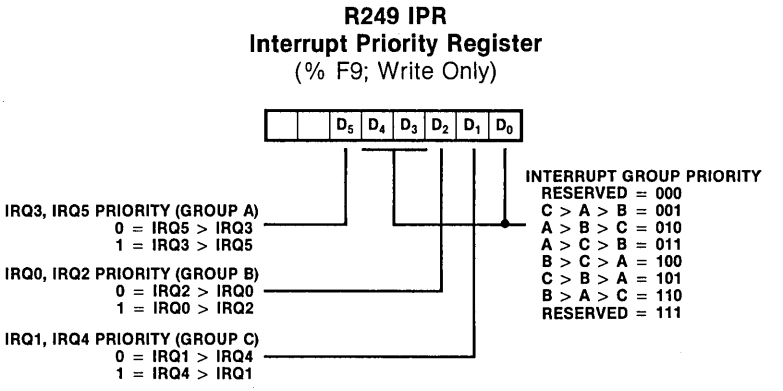


Figure 10-7. Interrupt Priority Register

Table 10-2. Interrupt Priority

Group	Bit	Priority Highest	Priority Lowest
C	D_1=0	IRQ_1	IRQ_4
	1	IRQ_4	IRQ_1
B	D_2=0	IRQ_2	IRQ_0
	1	IRQ_0	IRQ_2
A	D_5=0	IRQ_5	IRQ_3
	1	IRQ_3	IRQ_5

Bit Pattern D_4	D_2	D_0	Group Priority Highest --> Lowest
0	0	0	NOT USED
0	0	1	C A B
0	1	0	A B C
0	1	1	A C B
1	0	0	B C A
1	0	1	C B A
1	1	0	B A C
1	1	1	NOT USED

10.4.2 Interrupt Mask Register (IMR) Initialization

IMR (Figure 10-8) individually or globally enables or disables the six interrupt requests. When bits D_0-D_5 are set to 1, the corresponding interrupt requests are enabled. D_7 is the master enable and must be set before any of the individual interrupt requests can be recognized. Resetting D_7 globally disables all of the interrupt requests. D_7 is set and reset by the EI and DI instructions. It is automatically reset during an interrupt machine cycle and set following the execution of an Interrupt Return (IRET) instruction.

NOTE

D_7 must be reset by the DI instruction before the contents of the Interrupt Mask register or the Interrupt Priority register are changed except:

- Immediately after a hardware reset, or
- Immediately after executing an interrupt cycle and before IMR_7 has been set by any instruction.

10.4.3 Interrupt Request (IRQ) Register Initialization

IRQ (Figure 10-9) is a read/write register that stores the interrupt requests for both vectored and polled interrupts. When an interrupt is made on any of the six levels, the corresponding bit position in the register is set to 1. Bits D_0-D_5 are assigned to interrupt requests IRQ_0-IRQ_5, respectively.

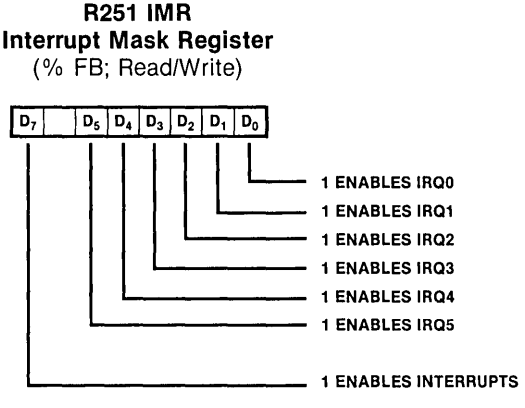


Figure 10-8. Interrupt Mask Register

IRQ is held in a Reset state until an EI instruction is executed. For polled processing, IRQ must still be initialized by an EI instruction, but IMR should first be cleared to 0 to individually inhibit all interrupt requests while interrupts are globally enabled:

```
CLR     IMR
EI
DI
```

10.5 IRQ SOFTWARE INTERRUPT GENERATION

IRQ can be used to generate software interrupts by specifying IRQ as the destination of any instruction referencing the register file. These Software Interrupts (SWI) are controlled in the same manner as hardware-generated requests, i.e., the IPR and the IMR control the priority and enabling of each SWI level.

To generate an SWI, the desired request bit in the IRQ is set as follows:

```
OR IRQ,#IRQLVL
```

where the immediate data, IRQLVL, has a 1 in the bit position corresponding to the level of the SWI desired. For example, if an SWI on level 5 is desired, IRQLVL would have a 1 in the bit 5 position:

```
OR IRQ,#%200100000
```

where the immediate data is preceded by %2 to indicate a binary constant. With this instruction, if the interrupt system is globally enabled, level 5 is enabled, and there are no higher priority pending requests, control is transferred to the service routine pointed to by the level 5 vector.

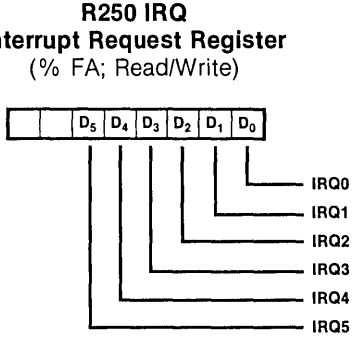


Figure 10-9. Interrupt Request Register

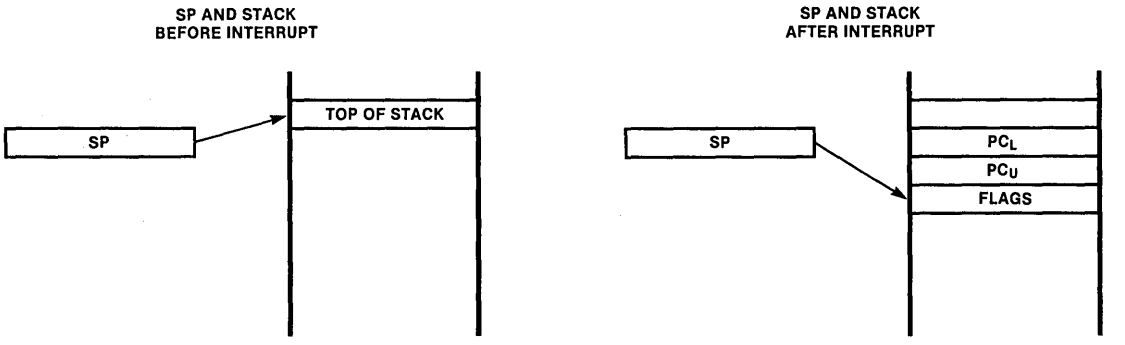


Figure 10-10. Effect of Interrupt on Stack

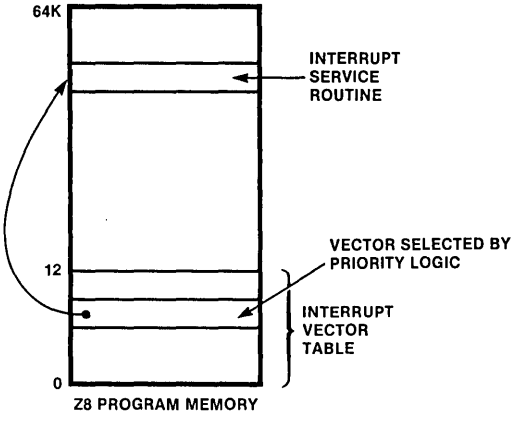


Figure 10-11. Interrupt Vectoring

10.6 VECTORED PROCESSING

Each Z8 interrupt level has its own vector. When an interrupt occurs, control passes to the service routine pointed to by the interrupt's location in program memory. The sequence of events for vectored interrupts is as follows:

- PUSH PC lower byte on stack
- PUSH PC upper byte on stack
- PUSH FLAGS on stack
- Fetch upper byte of vector
- Fetch lower byte of vector
- Branch to service routine specified by vector

Figures 10-10 and 10-11 show the vectored interrupt operation.

10.6.1 Vectored Interrupt Cycle Timing

Interrupt cycle timing for all Z8 devices except the Z8681 is diagrammed in Figure 10-12. Timing for the Z8681 ROMless device is different and is shown in Figure 10-13.

10.6.2 Nesting of Vectored Interrupts

Nesting of vectored interrupts allows higher priority requests to interrupt a lower priority request. To initiate vectored interrupt nesting, do the following during the interrupt service routine:

- Push the old IMR on the stack.
- Load IMR with a new mask to disable lower priority interrupts.
- Execute EI instruction.
- Proceed with interrupt processing.
- After processing is complete, execute DI instruction.
- Restore the IMR to its original value by returning the previous mask from the stack.
- Execute IRET.

Depending on the application, some simplification of the above procedure may be possible.

10.7 POLLED PROCESSING

Polled interrupt processing is supported by masking off the IRQ levels to be polled. This is accomplished by clearing the corresponding bit in the IMR to 0.

To initiate polled processing, check the bits of interest in the IRQ using the Test Under Mask (TM) instruction. If the bit is set, call or branch to the service routine. The service routine services the request, resets its Request bit in the IRQ, and branches or returns back to the main program. An example of a polling routine is as follows:

```
      TM IRQ,#MASK     !Test for request          !
      JR Z   NEXT      !If no request go to NEXT !
      CALL SERVICE     !If request is there      !
                       !then service it          !
   NEXT:
          .
          .
          .
SERVICE:               !Process Request          !
          .
          .
          .
      AND IRQ,#MASK_   !Clear Request bit        !
      RET              !Return to next           !
```

In this example, if IRQ_2 is being polled, MASK will be %200000100 (in binary) and MASK_ will be %211111011.

10.8 RESET CONDITIONS

During a reset, all bits in IPR are undefined.

In IMR, bit D_7 is 0 and bits D_0-D_5 are undefined. Bit D_6 is not implemented, though reading this bit returns 0.

IRQ bits D_0-D_5 are held at 0 until an EI instruction is executed. Bits D_6 and D_7 are not implemented, but reading these bits returns 0.

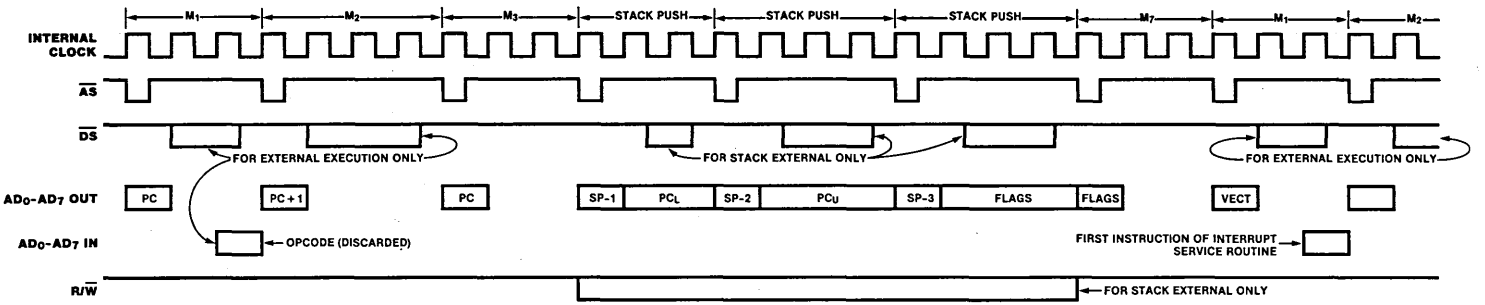


Figure 10-12. ROM Z8 Interrupt Timing (shrink parts)

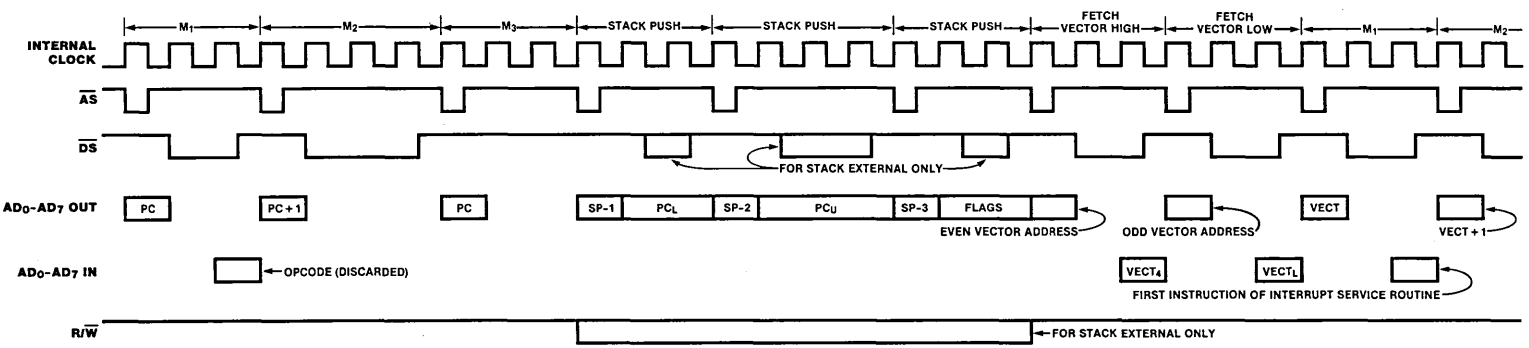


Figure 10-13. Z8681 ROMless Z8 Interrupt Timing

Chapter 11
Counter/Timers

11.1 INTRODUCTION

The Z8 provides two 8-bit counter/timers, T_0 and T_1, each driven by its own 6-bit prescaler, PRE_0 and PRE_1. Both counter/timers are independent of the processor instruction sequence, which relieves software from time-critical operations such as interval timing or event counting.

Each counter/timer operates in either Single-Pass or Continuous mode. At the end-of-count, counting either stops or the initial value is reloaded and counting continues. Under software control, new values are loaded immediately or when the end-of-count is reached. Software also controls counting mode, how a counter/timer is started or stopped, and its use of I/O lines. Both the counter and prescaler registers can be altered while the counter/timer is running.

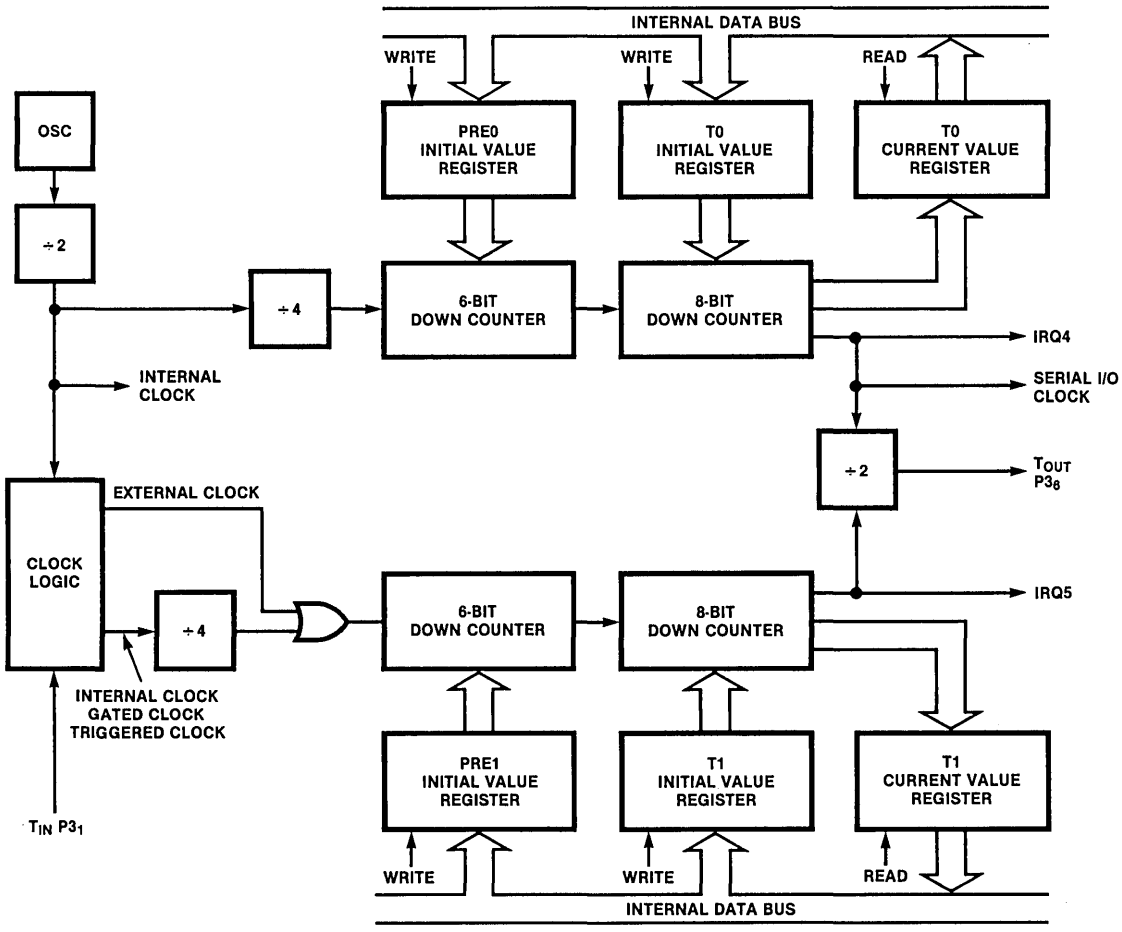


Figure 11-1. Counter/Timer Block Diagram

Counter/timers 0 and 1 are driven by a timer clock generated by dividing the internal clock by four. The divide-by-four stage, the 6-bit prescaler, and the 8-bit counter/timer form a synchronous 16-bit divide chain. Counter/timer 1 can also be driven by an external input (T_{IN}) via Port 3 line $P3_1$. Port 3 line $P3_6$ can serve as a timer output (T_{OUT}) through which T_0, T_1, or the internal clock can be output. The timer output will toggle at the end-of-count. Figure 11-1 is a block diagram of the counter/timers.

The counter/timer, prescaler, and associated mode registers are mapped into the register file as shown in Figure 11-2. This allows the software to treat the counter/timers as general-purpose registers, and eliminates the need for special instructions.

11.2 PRESCALERS AND COUNTER/TIMERS

The prescalers, PRE_0 (%F5) and PRE_1 (%F3), each consist of an 8-bit register and a 6-bit down-counter as shown in Figure 11-1. The prescaler registers are write-only registers. Reading the prescalers returns the value %FF. Figures 11-3 and 11-4 show the prescaler registers.

The six most significant bits (D_2-D_7) of PRE_0 or PRE_1 hold the prescalers count modulo, a value from 1 to 64 decimal. The prescaler registers also contain control bits that specify T_0 and T_1 counting modes. These bits also indicate whether the clock source for T_1 is internal or external. These control bits will be discussed in detail throughout this chapter.

The counter/timers, T_0 (%F4) and T1 ($\%F_2$), each consist of an 8-bit down-counter, a write-only register which holds the initial count value, and a read-only register which holds the current count value (Figure 11-1). The initial value can range from 1 to 256 decimal (%01,%02,..,%00). Figure 11-5 illustrates the counter/timer registers.

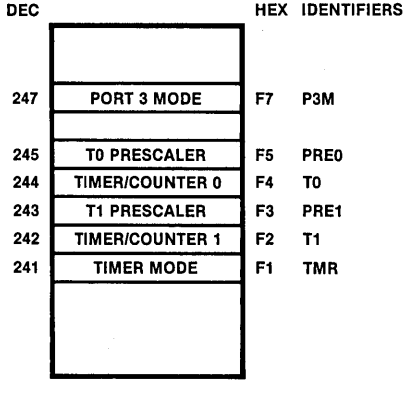


Figure 11-2. Counter/Timer Register Map

R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

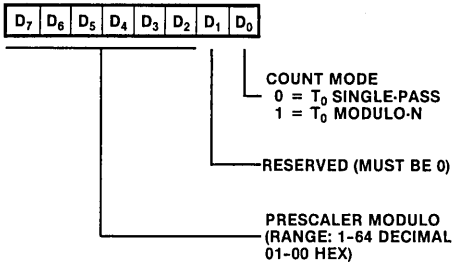


Figure 11-3. Prescaler 0 Register

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)

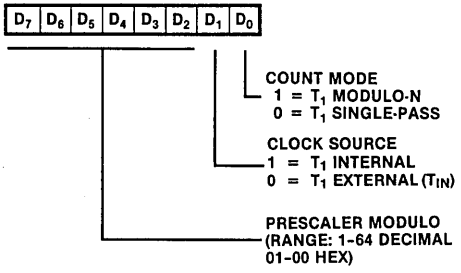


Figure 11-4. Prescaler 1 Register

R242 T1
Counter/Timer 1 Register
(% F2; Read/Write)
R244 T0
Counter/Timer 0 Register
(% F4; Read/Write)

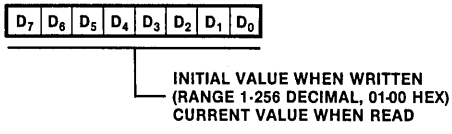


Figure 11-5. Counter/Timers 0 and 1 Registers

11.3 COUNTER/TIMER OPERATION

Under software control, counter/timers are started and stopped via the Timer Mode register (%F1) bits D_0-D_3 (Figure 11-6). Each counter/timer is associated with a Load bit and an Enable Count bit.

11.3.1 Load and Enable Count Bits

Setting the Load bit (D_0 to 1 for T_0 and D_2 to 1 for T_1) transfers the initial value in the prescaler and the counter/timer registers into their respective down-counters. The next internal clock resets bits D_0 and D_2 to 0, readying the Load bit for the next load operation. The initial values may be loaded into the down-counters at any time. If the counter/timer is running, it continues to do so and starts the count over with the initial value. Therefore, the Load bit actually functions as a software re-trigger.

The counter/timers remain at rest as long as the Enable Count bits D_1 and D_3 are both 0. To enable counting, the Enable Count bit (D_1 for T_0 and D_3 for T_1) must be set to 1. Counting actually starts when the Enable Count bit is written by an instruction. The first decrement occurs four internal clock periods after the Enable Count bit has been set.

The Load and Enable Count bits can be set at the same time. For example, using the instruction OR TMR #%03 sets both D_0 and D_1 of TMR to 1. This loads the initial values of PRE_0 and T_0 into their respective counters and starts the count after the M2T2 machine state after the operand is fetched (Figure 11-7).

11.3.2 Prescaler Operations

During counting, the programmed clock source drives the prescaler 6-bit counter. The counter is counted down from the value specified by bits D_2-D_7 of the corresponding prescaler register, PRE_0 or PRE_1 (Figure 11-8). When the prescaler counter reaches its end-of-count, the initial value is reloaded and counting continues. The prescaler never actually reaches 0. For example, if the prescaler is set to divide by 3, the count sequence is:

3-2-1-3-2-1-3-2....

Each time the prescaler reaches its end-of-count a carry is generated, which allows the counter/timer to decrement by one on the next timer clock input. When the counter/timer and the prescaler both reach their end-of-count, an interrupt request is generated -- IRQ_4 for T_0 and IRQ_5 for T_1. Depending on the counting mode selected, the counter/timer will either come to rest with its value at %00 (Single-Pass mode) or the initial value will be automatically reloaded and counting will continue (Continuous mode).

R241 TMR
Timer Mode Register
(% F1; Read/Write)

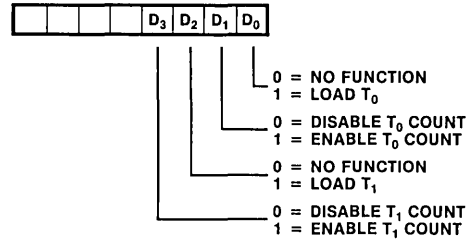


Figure 11-6. Timer Mode Register

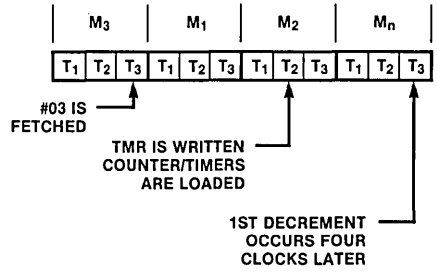


Figure 11-7. Starting The Count

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)
R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

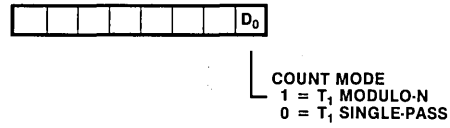


Figure 11-8. Counting Modes

The counting modes are controlled by bit D_0 of PRE_0 and PRE_1, with D_0 cleared to 0 for Single-pass counting mode or set to 1 for Continuous mode.

The counter/timers can be stopped at any time by setting the Enable Count bit to 0, and restarted by setting it back to 1. The counter/timer will continue its count value at the time it was stopped. The current value in the counter/timer (T_0 or T_1) can be read at any time without affecting the counting operation.

New initial values can be written to the prescaler or the counter/timer registers at any time. These values will be transferred to their respective down-counters on the next load operation. If the counter/timer mode is Continuous, the next load occurs on the timer clock following an end-of-count. New initial values should be written before the desired load operation, since the prescalers always effectively operate in Continuous count mode.

The time interval (i) until end-of-count, is given by the equation

$$i = t \times p \times v$$

in which t is 8 divided by XTAL frequency, p is the prescaler value (1 - 64), and v is the counter/timer value (1 - 256). It should be apparent that the prescaler and counter/timer are true divide-by-n counters.

11.4 T_{OUT} MODES

The Timer Mode register TMR (%F1) (Figure 11-10) is used in conjunction with the Port 3 Mode register P3M (%F7) (Figure 11-9) to configure $P3_6$ for T_{OUT} operation. In order for T_{OUT} to function, $P3_6$ must be defined as an output line by setting P3M bit D_5 to 0. Output is controlled by one of the counter/timers (T_0 or T_1) or the internal clock.

The counter/timer to be output is selected by TMR bits D_7 and D_6. T_0 is selected to drive the T_{OUT} line by setting D_7 to 0 and D_6 to 1. Likewise, T1 is selected by setting D_7 and D_6 to 1 and 0 respectively. The counter/timer T_{OUT} mode is turned off by setting TMR bits D_7 and D_6 both to 0, freeing $P3_6$ to be a data output line.

T_{OUT} is initialized to a logic 1 whenever the TMR Load bit (D_0 for T_0 or D_2 for T_1) is set to 1.

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

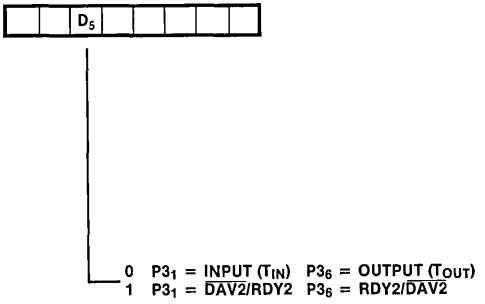


Figure 11-9.
Port 3 Mode Register T_{OUT} Operation

R241 TMR
Timer Mode Register
(% F1; Read/Write)

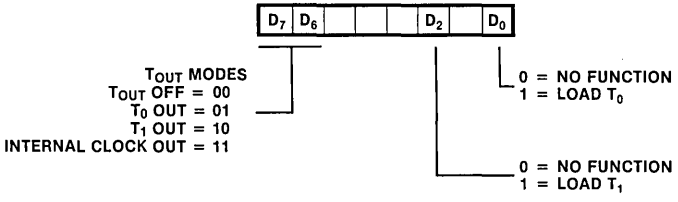


Figure 11-10. Timer Mode Register T_{OUT} Operation

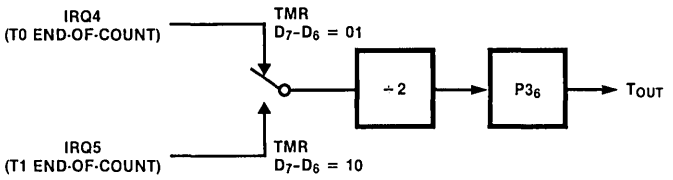


Figure 11-11. Counter/Timers Output Via T_{OUT}

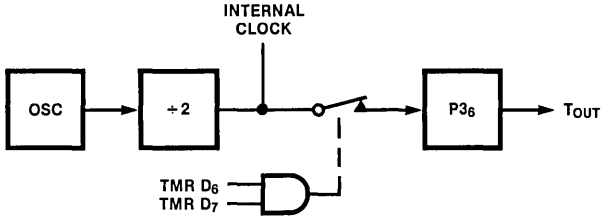


Figure 11-12. Internal Clock Output Via T_{OUT}

At end-of-count, the interrupt request line (IRQ_4 or IRQ_5), clocks a toggle flip-flop. The output of this flip-flop drives the T_{OUT} line, $P3_6$. In all cases, when the selected counter/timer reaches its end-of-count, T_{OUT} toggles to its opposite state (Figure 11-11). If, for example, the counter/timer is in Continuous counting mode, T_{OUT} will have a 50% duty cycle output. This duty cycle can easily be controlled by varying the initial values after each end-of-count.

The internal clock can be selected as output instead of T_0 or T_1 by setting TMR bits D_7 and D_6 both to 1. The internal clock (XTAL frequency/2) is then directly output on $P3_6$ (Figure 11-12).

While programmed as T_{OUT}, $P3_6$ cannot be modified by a write to port register P3. However, the Z8 software can examine $P3_6$'s current output by reading the port register.

11.5 T_{IN} MODES

The Timer Mode register TMR (%F1) (Figure 11-13) is used in conjunction with the Prescaler register PRE_1 (%F3) (Figure 11-14) to configure $P3_1$ as T_{IN}. T_{IN} is used in conjunction with T_1 in one of four modes:

- External clock input
- Gated internal clock
- Triggered internal clock
- Retriggerable internal clock

R241 TMR
Timer Mode Register
(% F1; Read/Write)

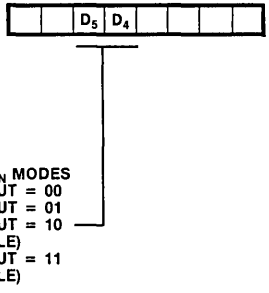


Figure 11-13. Timer Mode Register T_{IN} Operation

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)

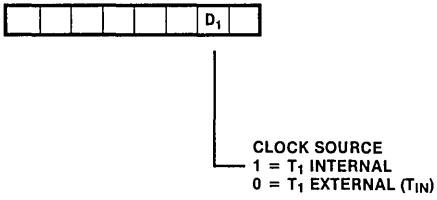


Figure 11-14. Prescaler 1 T_{IN} Operation

The counter/timer clock source must be configured for external by setting PRE_1 bit D_2 to 0. The Timer Mode register bits D_5 and D_4 can then be used to select the desired T_{IN} operation.

For T_1 to start counting as a result of a T_{IN} input, the Enable Count bit D_3 in TMR must be set to 1. When using T_{IN} as an external clock or a gate input, the initial values must be loaded into the down-counters by setting the Load bit D_2 in TMR to a 1 before counting begins. In the descriptions of T_{IN} that follow, it is assumed that the programmer has performed these operations. Initial values are automatically loaded in Trigger and Retrigger modes so software loading is unnecessary.

It is suggested that $P3_1$ be configured as an input line by setting P3M bit D_5 to 0 although T_{IN} is still functional if $P3_1$ is configured as a handshake input.

Each High-to-Low transition on T_{IN} generates interrupt request IRQ_2, regardless of the selected T_{IN} mode or the enabled/disabled state of T_1. IRQ_2 must therefore be masked or enabled according to the needs of the application.

11.5.1 External Clock Input Mode

The T_{IN} External Clock Input mode (TMR bits D_5 and D_4 both set to 0) supports counting of external events, where an event is considered to be a High-to-Low transition on T_{IN} (Figure 11-15). occurrence (Single-Pass mode) or on every nth occurrence (Continuous mode) of that event.

11.5.2 Gated Internal Clock Mode

The T_{IN} Gated Internal Clock mode (TMR bits D_5 and D_4 set to 0 and 1 respectively) measures the duration of an external event. In this mode, the T_1 prescaler is driven by the internal timer clock, gated by a High level on T_{IN} (Figure 11-16). T_1 counts while T_{IN} is High and stops counting while T_{IN} is Low. Interrupt request IRQ_2 is generated on the High-to-Low transition of T_{IN}, signaling the end of the gate input. Interrupt request IRQ_5 is generated if T_1 reaches its end-of-count.

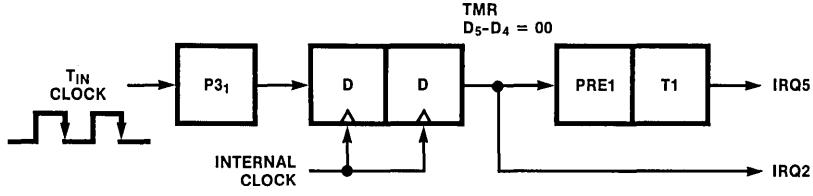


Figure 11-15. External Clock Input Mode

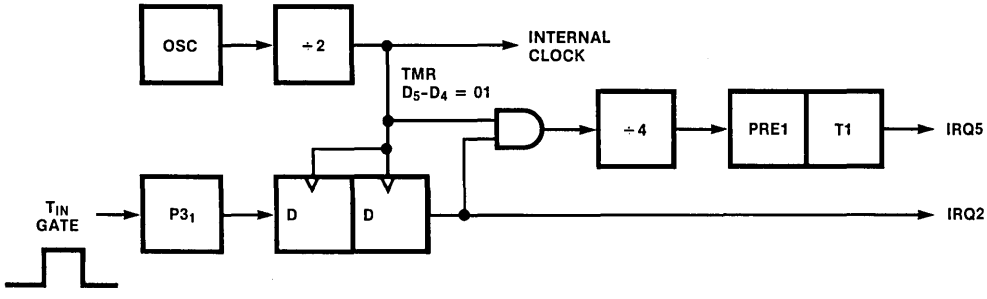


Figure 11-16. Gated Clock Input Mode

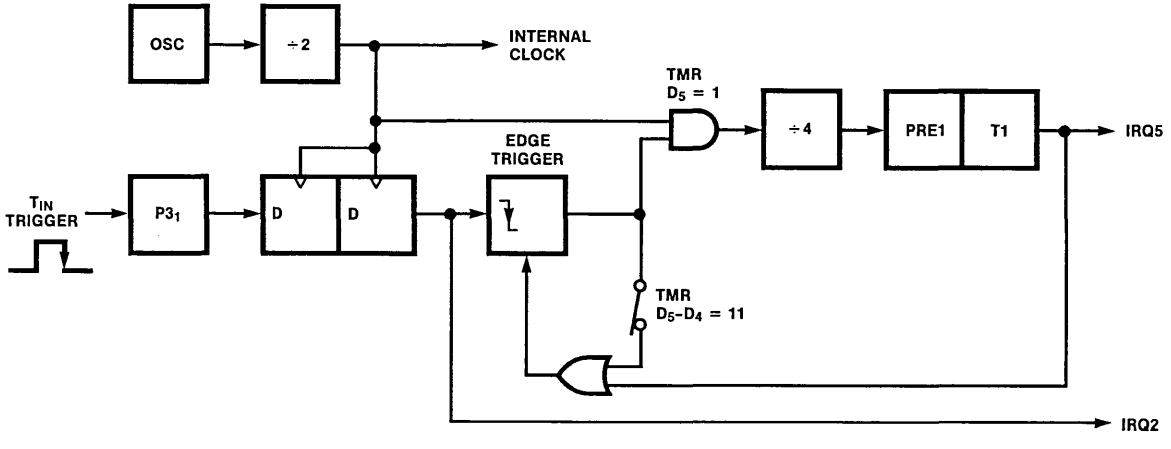


Figure 11-17. Triggered Clock Mode

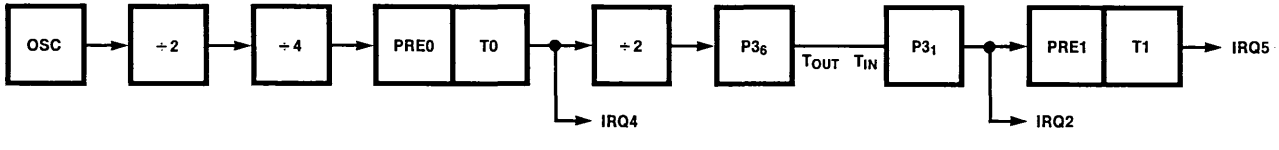


Figure 11-18. Cascaded Counter/Timers

11.5.3 Triggered Input Mode

The T_{IN} Triggered Input mode (TMR bits D_5 and D_4 set to 1 and 0 respectively) causes T_1 to start counting as the result of an external event (Figure 11-17). T_1 is then loaded and clocked by the internal timer clock following the first High-to-Low transition on the T_{IN} input. Subsequent T_{IN} transitions do not affect T_1. In the Single-Pass mode, the Enable bit is reset whenever T_1 reaches its end-of-count. Further T_{IN} transitions will have no effect on T_1 until software sets the Enable Count bit again. In Continuous mode, once T_1 is triggered counting continues until software resets the Enable Count bit. Interrupt request IRQ_5 is generated when T_1 reaches its end-of-count.

11.5.4 Retriggerable Input Mode

The T_{IN} Retriggerable Input mode (TMR bits D_5 and D_4 both set to 1) causes T_1 to load and start counting on every occurrence of a High-to-Low transition on T_{IN} (Figure 11-17). Interrupt request IRQ_5 will be generated if the programmed time interval (determined by T_1 prescaler and counter/timer register initial values) has elapsed since the last High-to-Low transition on T_{IN}. In Single-Pass mode, the end-of-count resets the Enable Count bit. Subsequent T_{IN} transitions will not cause T_1 to load and start counting until software sets the Enable Count bit again. In Continuous mode, counting continues once T_1 is triggered until software resets the Enable Count bit. When enabled, each High-to-Low T_{IN} transition causes T_1 to reload and restart counting. Interrupt request IRQ_5 is generated on every end-of-count.

11.6 CASCADING COUNTER/TIMERS

For some applications, it may be necessary to measure a time interval greater than a single counter/timer can measure. In this case, T_{IN} and T_{OUT} can be used to cascade T_0 and T_1 as a single unit (Figure 11-18). T_0 should be configured to operate in Continuous mode and to drive T_{OUT}. T_{IN} should be configured as an external clock input to T_1 and wired back to T_{OUT}. On every other T_0 end-of-count, T_{OUT} undergoes a High-to-Low transition which causes T_1 to count. T_1 can operate in either Single-Pass or Continuous mode. Each time T_1's end-of-count is reached, interrupt request IRQ_5 is generated. Interrupt requests IRQ_2 (T_{IN} High-to-Low transitions) and IRQ_4 (T_0 end-of-count) are also generated but are most likely of no importance in this configuration and should be disabled.

11.7 RESET CONDITIONS

After a hardware reset, the counter/timers are disabled and the contents of both the counter/timer registers and the prescaler modulos are undefined. However, the counting modes are configured for Single-Pass and T_1's clock source is set for external. T_{IN} is set for External Clock mode, and the T_{OUT} mode is off. Figures 11-19 through 11-22 show the binary reset values of the Prescaler, Counter/Timer, and Timer Mode registers.

R242 T1
Counter/Timer 1 Register
(% F2; Read/Write)
R244 T0
Counter/Timer 0 Register
(% F4; Read/Write)

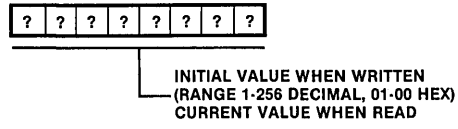


Figure 11-19. Counter/Timer Reset

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)

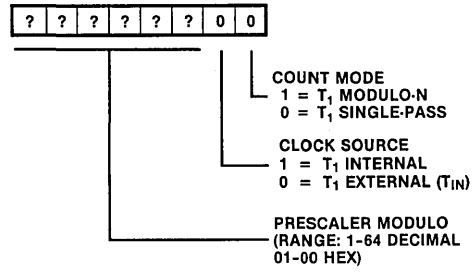


Figure 11-20. Prescaler 1 Register Reset

R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

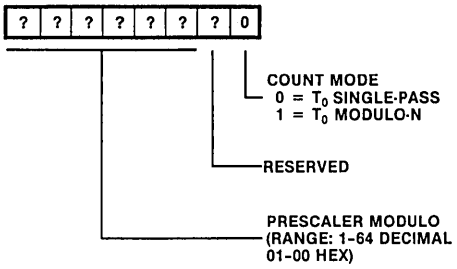


Figure 11-21. Prescaler 0 Reset

R241 TMR
Timer Mode Register
(% F1; Read/Write)

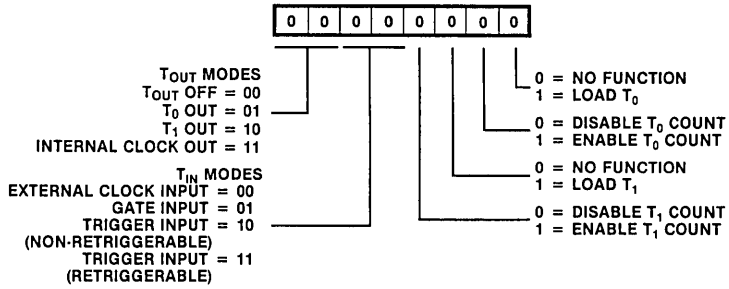


Figure 11-22. Timer Mode Register Reset

Chapter 12
Serial I/O

12.1 INTRODUCTION

The Z8 microcomputer contains an on-board full-duplex receiver/transmitter for asynchronous data communications. The receiver/transmitter consists of a Serial I/O register SIO (%F1) and its associated control logic (Figure 12-1). The SIO is actually two registers--the receiver buffer and the transmitter buffer--which are used in conjunction with counter/timer T_0 and Port 3 I/O lines $P3_0$ (input) and $P3_7$ (output). Counter/timer T_0 provides the clock input for control of the data rates.

Configuration of the serial I/O is controlled by the Port 3 Mode register, P3M. The Z8 always transmits 8 bits between the start and stop bits; that is, 8 data bits or 7 data bits and 1 parity bit. Odd parity generation and detection is supported.

The Serial I/O register and its associated Mode Control registers are mapped into the register file as shown in Figure 12-2. This organization allows the software to access the serial I/O as general-purpose registers, eliminating the need for special instructions.

12.2 BIT RATE GENERATION

When Port 3 Mode register bit D_6 is set to 1, the serial I/O is enabled and T_0 automatically becomes the bit rate generator (Figure 12-3). T_0's end-of-count signal no longer generates interrupt request IRQ_4; instead, the signal is used as the input to the divide-by-16 counters (one each for the receiver and the transmitter) which clock the data stream.

The divide chain that generates the bit rate is shown in Figure 12-4. The bit rate is given by the following equation:

bit rate = XTAL frequency/(2 x 4 x p x t x 16)

where p and t are the initial values in the Prescaler and Counter/Timer registers, respectively.

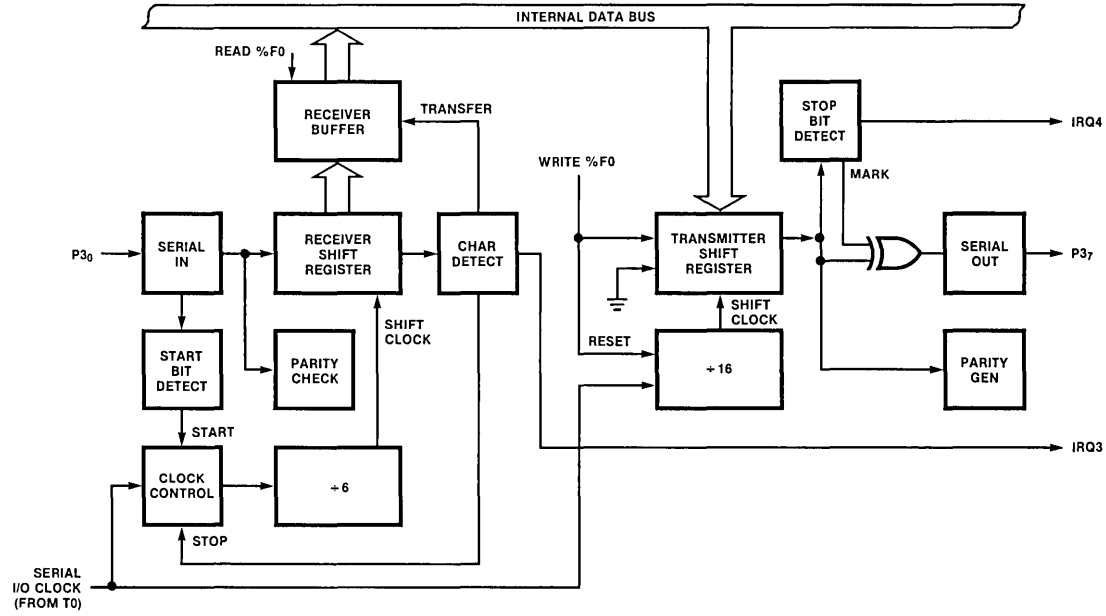


Figure 12-1. Serial I/O Block Diagram

The final divide-by-16 is required since T_0 runs at 16 times the bit rate in order to synchronize on the incoming data.

To configure the Z8 for a specific bit rate, appropriate values as determined by the above equation must be loaded into registers PRE_0 (%F5) and T_0 (%F4). PRE_0 also controls the counting mode for T_0 and should therefore be set to the Continuous mode (D_1 set to 1).

For example, given an input clock frequency (fXTAL) of 11.9808 MHz and a selected bit rate of 1200 bits per second, the equation is satisfied by p=39 and t=2. Counter/timer T_0 should be set to %02. With T_0 in Continuous mode, the value of PRE_0 becomes %9D (Figure 12-5).

Table 12-1 lists several commonly used bit rates and the values of fXTAL, p, and t required to derive them. This list is presented for convenience and is not intended to be exhaustive.

The bit rate generator is started by setting the Timer Mode register TMR (%F1) bits D_1 and D_0 both to 1 (Figure 12-6). This transfers the contents of the Prescaler and Counter/Timer registers to their corresponding down-counters. In addition, counting is enabled so that serial I/O operations begin.

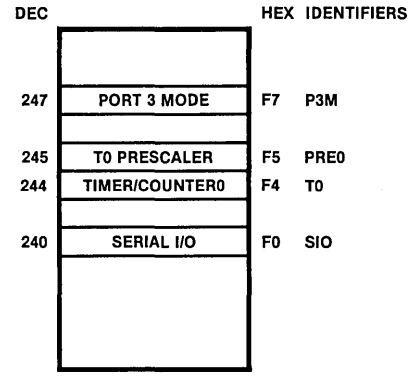


Figure 12-2. Serial I/O Register Map

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

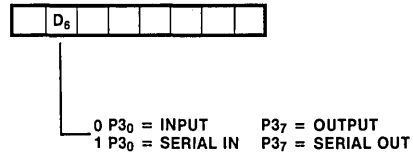


Figure 12-3. Port 3 Mode Register and Bit Rate Generation

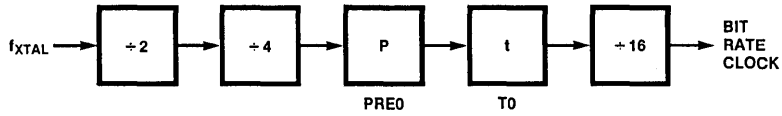


Figure 12-4. Bit Rate Divide Chain

Table 12-1. Bit Rate

Bit Rate	7,3728		7,9872		9,8304		11,0592		11,6736		11,9808		12,2880	
	p	t	p	t	p	t	p	t	p	t	p	t	p	t
19200	3	1	--	--	4	1	--	--	--	--	--	--	5	1
9600	3	2	--	--	4	2	9	1	--	--	--	--	5	2
4800	3	4	13	1	4	4	9	2	19	1	--	--	5	4
2400	3	8	13	2	4	8	9	4	19	2	39	1	5	8
1200	3	16	13	4	4	16	9	8	19	4	39	2	5	16
600	3	32	13	8	4	32	9	16	19	8	39	4	5	32
300	3	64	13	16	4	64	9	32	19	16	39	8	5	64
150	3	128	13	32	4	128	9	64	19	32	39	16	5	128
110	3	175	3	189	4	175	5	157	4	207	17	50	8	109

R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

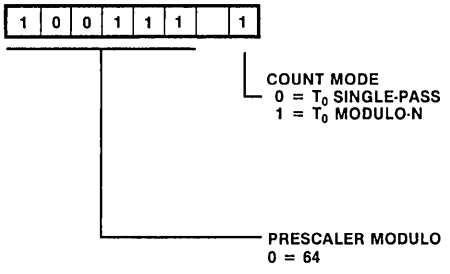


Figure 12-5. Prescaler 0 Register and Bit Rate Generation

R241 TMR
Timer Mode Register
(% F1; Read/Write)

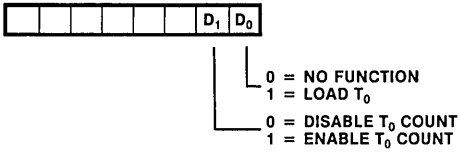


Figure 12-6. Timer Mode Register and Bit Rate Generation

12.3 RECEIVER OPERATION

The receiver consists of a receiver buffer (SIO [%F0]), a serial-in, parallel-out Shift register, parity checking, and data synchronizing logic. The receiver block diagram is shown as part of Figure 12-1.

12.3.1 Receiver Shift Register

After a hardware reset or after a character has been received, the Receiver Shift register is initialized to all 1s and the shift clock is stopped. Serial data, input through Port 3 pin $P3_0$, is synchronized to the internal clock by two D-type flip flops before being input to the Shift register and the start bit detection circuitry.

The start bit detection circuitry monitors the incoming data stream, looking for a start bit (a High-to-Low input transition). When a start bit is detected, the shift clock logic is enabled. The T_0 input is divided by 16 and, when the count equals 8, the divider outputs a shift clock. This clock shifts the start bit into the Receiver Shift register at the center of the bit time. Before the shift actually occurs, the input is rechecked to ensure that the start bit is valid. If the detected start bit is false, the receiver is reset and the process of looking for a start bit is repeated. If the start bit is valid, the data is shifted into the Shift register every sixteen counts until a full character is assembled (Figure 12-7).

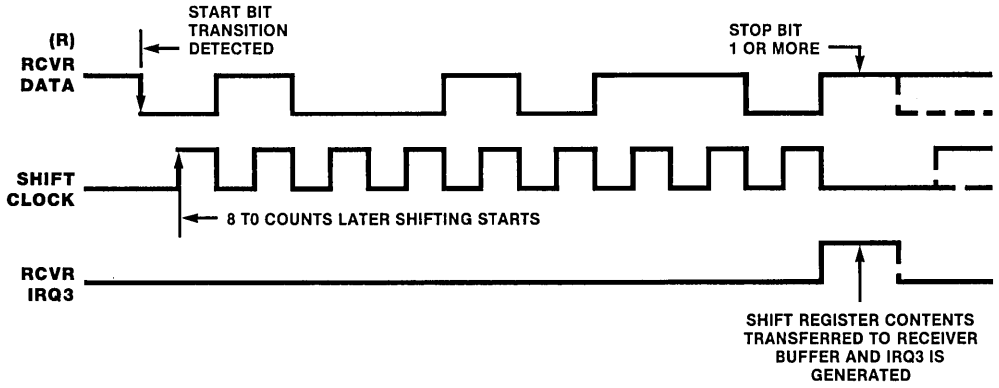


Figure 12-7. Receiver Timing

After a full character has been assembled in the Shift register, the data is transferred to the receiver's buffer, SIO (%F0), and interrupt request IRQ_3 is generated. The shift clock is stopped and the Shift register reset to all 1s. The start bit detection circuitry begins monitoring the data input for the next start bit. This cycle allows the receiver to synchronize on the center of the bit time for each incoming character.

12.3.2 Overwrites

Although the receiver is buffered, it is not protected from being overwritten, so the software must read the SIO register within one character time after the interrupt request. The Z8 does not have a flag to indicate this overrun condition. If polling is used, the IRQ_3 bit in the Interrupt Request register must be reset by software.

12.3.3 Framing Errors

Framing error detection is not supported by the receiver hardware, but by responding to the interrupt request within one character bit time, the software can test for a stop bit at $P3_0$. Port 3 bits are always readable, which facilitates break detection. For example, if a null character is received, testing $P3_0$ results in a 0 being read.

12.3.4 Parity

The data format supported by the receiver must have a start bit, eight data bits, and at least one stop bit. If parity is on, bit D_7 of the data received will be replaced by a Parity Error flag. A parity error sets D_7 to 1; otherwise, D_7 is set to 0. Figure 12-8 shows these data formats.

The Z8 hardware supports odd parity only, which is enabled by setting Port 3 Mode register bit D_7 to 1 (Figure 12-9). If even parity is required, the Parity mode should be disabled (i.e. P3M D_7 set to 0), and software must calculate the received data's parity.

12.4 TRANSMITTER OPERATION

The transmitter consists of a transmitter buffer (SIO (%F0)), a parity generator, and associated control logic. The transmitter block diagram is shown as part of Figure 12-1.

After a hardware reset or after a character has been transmitted, the transmitter is forced to a marking state (output always High) until a character is loaded into the transmitter buffer, SIO (%F0). The transmitter is loaded by specifying the SIO as the destination register of any instruction.

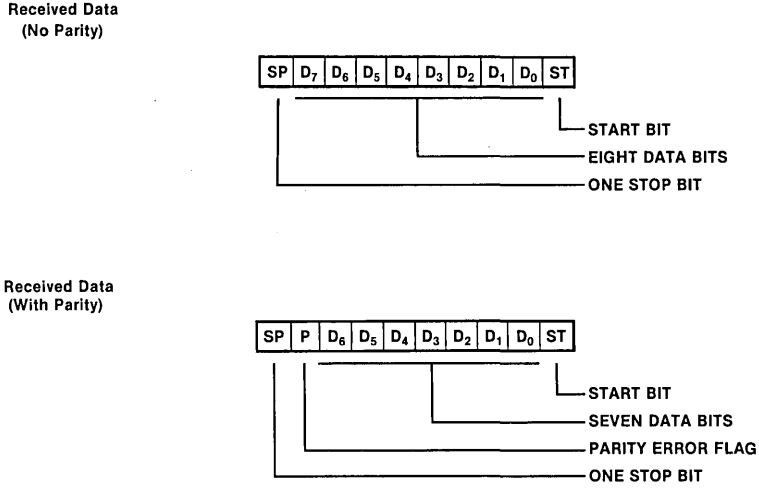


Figure 12-8. Receiver Data Formats

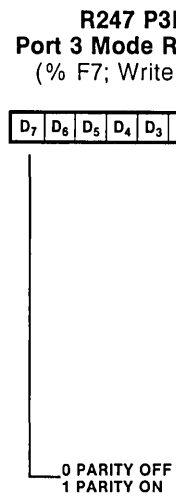


Figure 12-9. Parity and Port 3 Mode Register

T_0's output drives a divide-by-16 counter which in turn generates a shift clock every 16 counts. This counter is reset when the transmitter buffer is written by an instruction. This reset synchronizes the shift clock to the software. The transmitter then outputs one bit per shift clock, through Port 3 pin $P3_7$, until a start bit, the character written to the buffer, and two stop bits have been transmitted. After the second stop bit has been transmitted, the output is again forced to a marking state. Interrupt request IRQ_4 is generated and this notifies the processor that the transmitter is ready to accept another character.

12.4.1 Overwrites

The user is not protected from overwriting the transmitter, so it is up to the software to respond to IRQ_4 appropriately. If polling is used, the IRQ_4 bit in the Interrupt Request register must be reset.

12.4.2 Parity

The data format supported by the transmitter has a start bit, eight data bits, and at least two stop bits. If parity is on, bit D_7 of the data transmitted will be replaced by an odd parity bit. Figure 12-10 shows the transmitter data formats.

Parity is enabled by setting Port 3 Mode register bit D_7 to 1. If even parity is required, the parity mode should be disabled (i.e. P3M D_7 set to 0), and software must modify the data to include even parity.

Since the transmitter can be overwritten, the user is able to generate a break signal. This is done by writing null characters to the transmitter buffer (SIO, %F0) at a rate which does not allow the stop bits to be output. Each time the SIO is loaded, the divide-by-16 counter is re-synchronized and a new start bit is output followed by data.

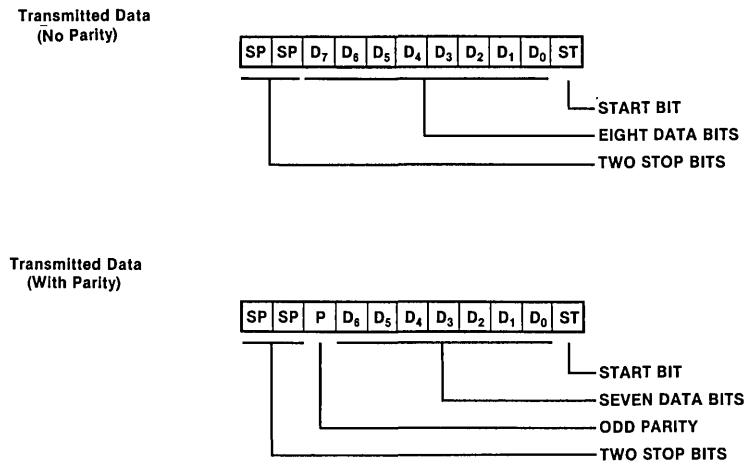


Figure 12-10. Transmitter Data Formats

12.5 RESET CONDITIONS

After a hardware reset, the Serial I/O register contents are undefined, and Serial mode and parity are disabled. Figures 12-11 and 12-12 show the binary reset values of the Serial I/O register and its associated mode register P3M.

R240 SIO
Serial I/O Register
(% F0; Read/Write)

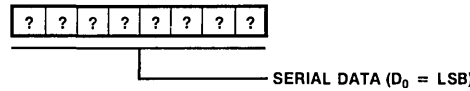


Figure 12-11. Serial I/O Register Reset

R247 P3M
Port 3 Mode Register
(% F78; Write Only)

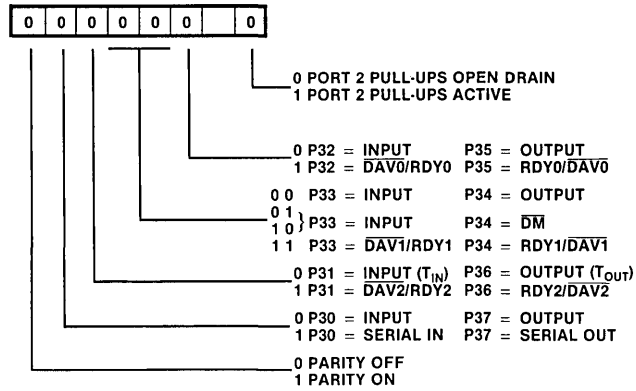


Figure 12-12. Port 3 Register Reset

A

Zilog

Appendix A
Pin Descriptions and Functions

This appendix contains pin information and physical descriptions for the Z8 development device (Z8612) and Protopack emulator (Z8603/13). Pin descriptions for the Z8601/11 and Z8681/82 microcomputers can be found in Chapters 6 and 7, respectively.

A.1 DEVELOPMENT DEVICE (Z8612)

The pin mnemonics and descriptions presented for the Z8 microcomputers (Chapter 6) also apply to the development device. Additional pin descriptions are as follows:

A_0-A_{11}. Program Memory Address (outputs). These lines are used to access the first 4K bytes of the external program memory.

D_0-D_7. Program Data (inputs). Data from the external program memory is input through these pins.

$\overline{IACK}$. Interrupt Acknowledge (output, active High). $\overline{IACK}$ is driven High in response to an interrupt during the interrupt machine cycle.

$\overline{MDS}$. Program Memory Data Strobe (output, active Low). $\overline{MDS}$ is Low during an instruction fetch cycle when the first 4K bytes of program memory are being accessed.

SCLK. System Clock (output). SCLK is the internal clock output through a buffer. The clock rate is equal to one-half the crystal frequency.

$\overline{SYNC}$. Instruction Sync (output, active Low). This strobe output is forced Low during the internal clock period preceding an opcode fetch.

A.2 PROTOPACK EMULATOR (Z8603/13)

Both the Z8603 and Z8613 devices use a 40-pin package that also has a 24-pin "piggy-back" socket. An EPROM or ROM can be installed on the back of the emulator's standard 40-pin package via the socket (Figure A-3). A single +5 V dc power source is required. Figure A-4 illustrates the pinout for the socket carried piggyback. The socket is designed to accept a 2716 EPROM for the Z8603 and a 2732 EPROM for the Z8613 device.

Pin mnemonics and descriptions are the same as those for the Z8601/11 microcomputer (Chapter 6). Descriptions for the additional (24-pin socket) memory interface lines are the same as those given for the development devices above.

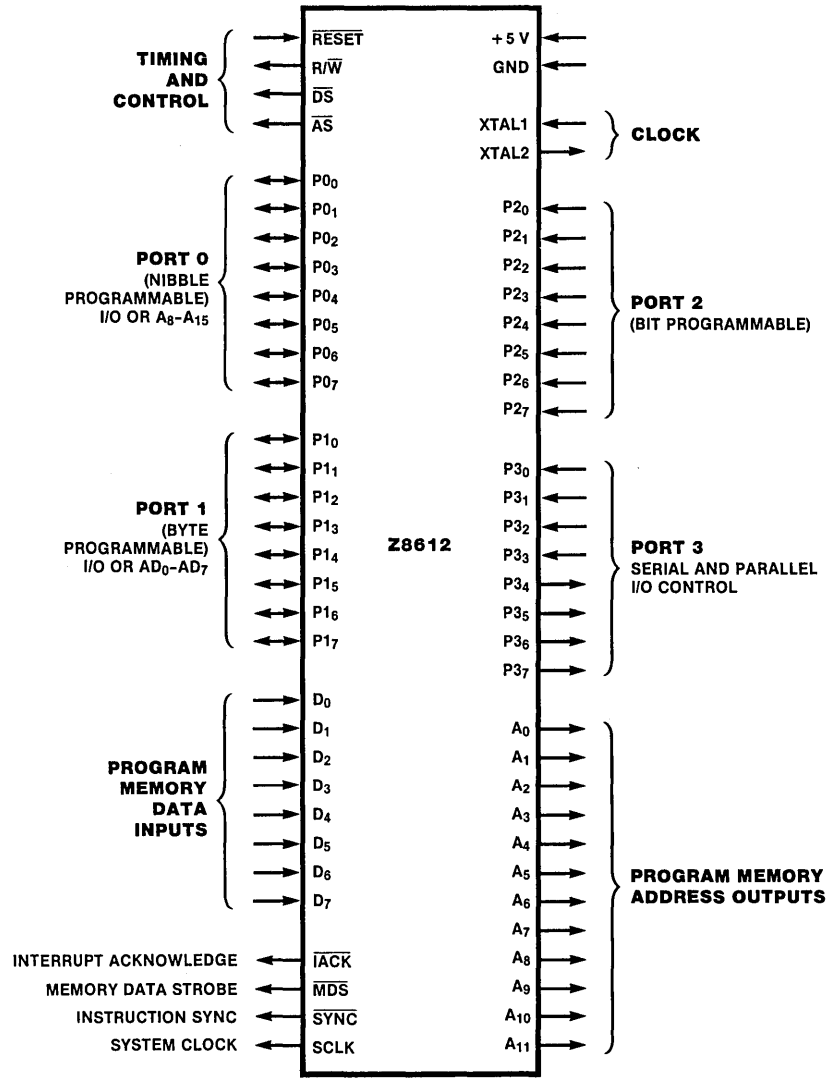


Figure A-1. Z8612 Pin Functions

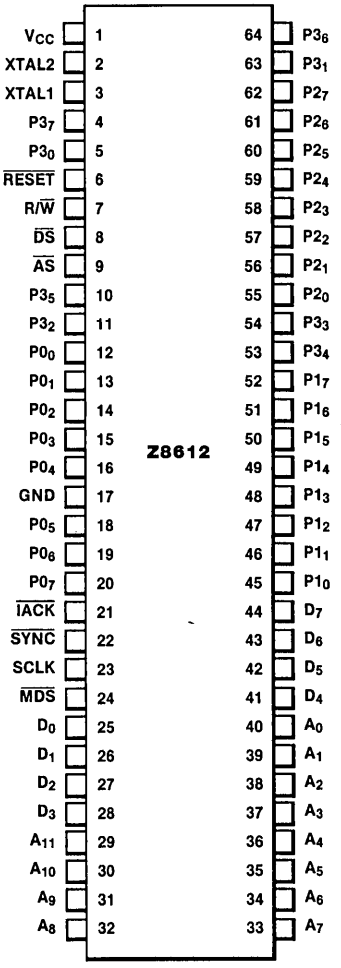


Figure A-2. Z8612 Pin Assignments

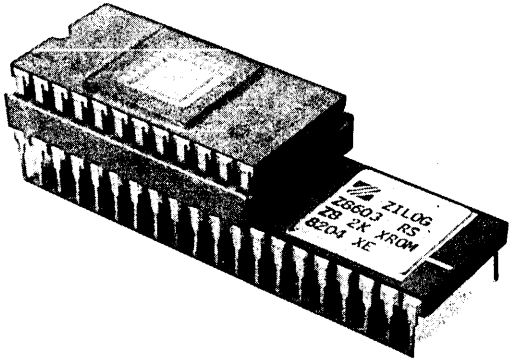


Figure A-3. Protopack Emulator

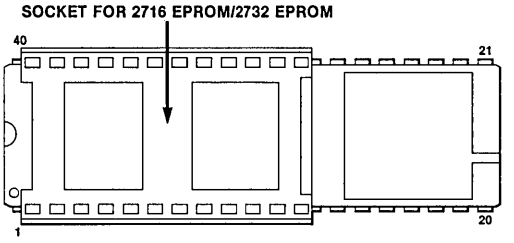


Figure A-4. Protopack EPROM Socket

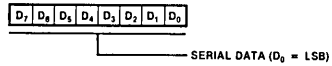
B

Zilog

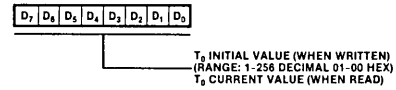
Appendix B
Control Registers

Registers

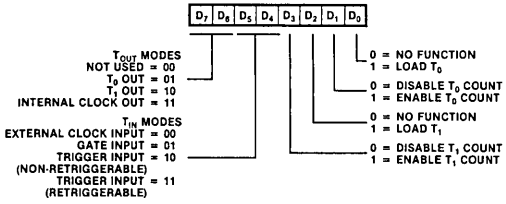
R240 SIO
Serial I/O Register
($F0_H$; Read/Write)



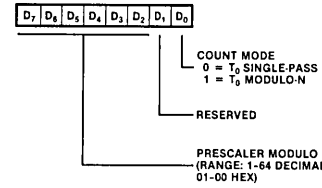
R244 T0
Counter/Timer 0 Register
($F4_H$; Read/Write)



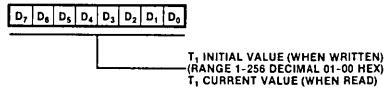
R241 TMR
Timer Mode Register
($F1_H$; Read/Write)



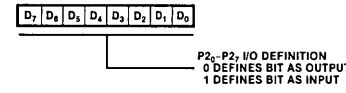
R245 PRE0
Prescaler 0 Register
($F5_H$; Write Only)



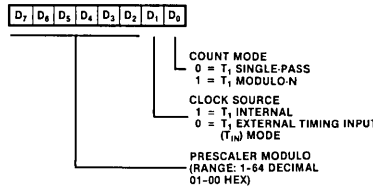
R242 T1
Counter Timer 1 Register
($F2_H$; Read/Write)



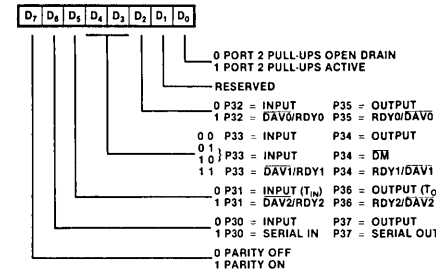
R246 P2M
Port 2 Mode Register
($F6_H$; Write Only)



R243 PRE1
Prescaler 1 Register
($F3_H$; Write Only)

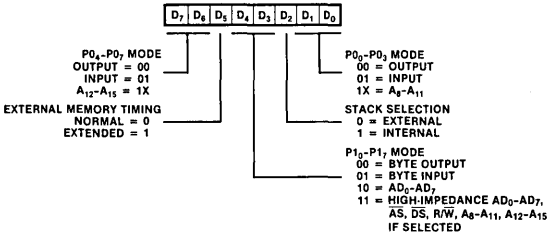


R247 P3M
Port 3 Mode Register
($F7_H$; Write Only)

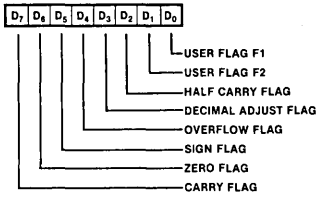


Registers
(Continued)

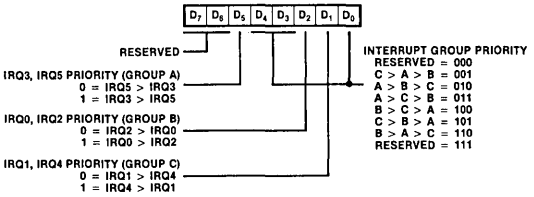
R248 P01M
Port 0 and 1 Mode Register
($F8_H$; Write Only)



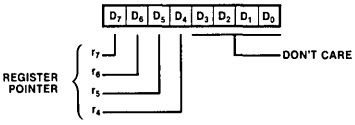
R252 FLAGS
Flag Register
(FC_H; Read/Write)



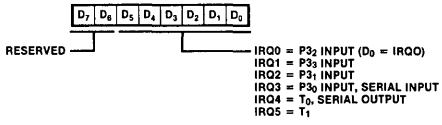
R249 IPR
Interrupt Priority Register
($F9_H$; Write Only)



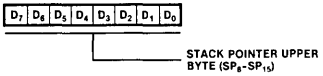
R253 RP
Register Pointer
(FD_H; Read/Write)



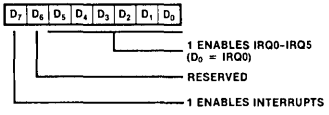
R250 IRQ
Interrupt Request Register
(FA_H; Read/Write)



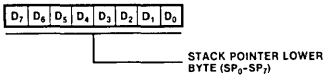
R254 SPH
Stack Pointer
(FE_H; Read/Write)



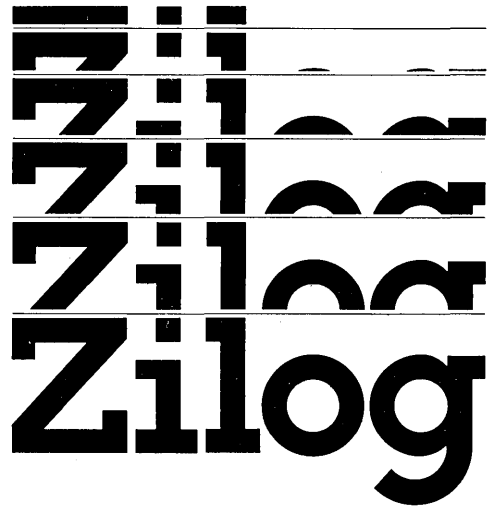
R251 IMR
Interrupt Mask Register
(FB_H; Read/Write)



R255 SPL
Stack Pointer
(FF_H; Read/Write)



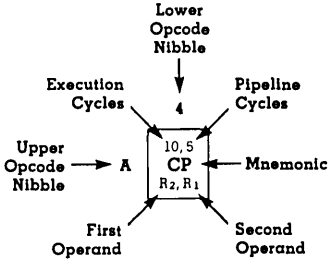
c



Opcode Map

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1, r_2	6,5 **ADD** r_1, Ir_2	10,5 **ADD** R_2, R_1	10,5 **ADD** IR_2, R_1	10,5 **ADD** R_1, IM	10,5 **ADD** IR_1, IM	6,5 **LD** r_1, R_2	6,5 **LD** r_2, R_1	12/10,5 **DJNZ** r_1, RA	12/10,0 **JR** cc, RA	6,5 **LD** r_1, IM	12/10,0 **JP** cc, DA	6,5 **INC** r_1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1, r_2	6,5 **ADC** r_1, Ir_2	10,5 **ADC** R_2, R_1	10,5 **ADC** IR_2, R_1	10,5 **ADC** R_1, IM	10,5 **ADC** IR_1, IM								
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1, r_2	6,5 **SUB** r_1, Ir_2	10,5 **SUB** R_2, R_1	10,5 **SUB** IR_2, R_1	10,5 **SUB** R_1, IM	10,5 **SUB** IR_1, IM								
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1, r_2	6,5 **SBC** r_1, Ir_2	10,5 **SBC** R_2, R_1	10,5 **SBC** IR_2, R_1	10,5 **SBC** R_1, IM	10,5 **SBC** IR_1, IM								
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1, r_2	6,5 **OR** r_1, Ir_2	10,5 **OR** R_2, R_1	10,5 **OR** IR_2, R_1	10,5 **OR** R_1, IM	10,5 **OR** IR_1, IM								
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1, r_2	6,5 **AND** r_1, Ir_2	10,5 **AND** R_2, R_1	10,5 **AND** IR_2, R_1	10,5 **AND** R_1, IM	10,5 **AND** IR_1, IM								
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1, r_2	6,5 **TCM** r_1, Ir_2	10,5 **TCM** R_2, R_1	10,5 **TCM** IR_2, R_1	10,5 **TCM** R_1, IM	10,5 **TCM** IR_1, IM								
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1, r_2	6,5 **TM** r_1, Ir_2	10,5 **TM** R_2, R_1	10,5 **TM** IR_2, R_1	10,5 **TM** R_1, IM	10,5 **TM** IR_1, IM								
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1, Irr_2	18,0 **LDEI** Ir_1, Irr_2												6,1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** r_2, Irr_1	18,0 **LDEI** Ir_2, Irr_1												6,1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1, r_2	6,5 **CP** r_1, Ir_2	10,5 **CP** R_2, R_1	10,5 **CP** IR_2, R_1	10,5 **CP** R_1, IM	10,5 **CP** IR_1, IM								14,0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1, r_2	6,5 **XOR** r_1, Ir_2	10,5 **XOR** R_2, R_1	10,5 **XOR** IR_2, R_1	10,5 **XOR** R_1, IM	10,5 **XOR** IR_1, IM								16,0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1, Irr_2	18,0 **LDCI** Ir_1, Irr_2				10,5 **LD** r_1, x, R_2								6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2, Irr_1	18,0 **LDCI** Ir_2, Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2, x, R_1								6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1, Ir_2	10,5 **LD** R_2, R_1	10,5 **LD** IR_2, R_1	10,5 **LD** R_1, IM	10,5 **LD** IR_1, IM								6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1, r_2		10,5 **LD** R_2, IR_1										6,0 **NOP**
Bytes per Instruction	2				3				2					3	1	



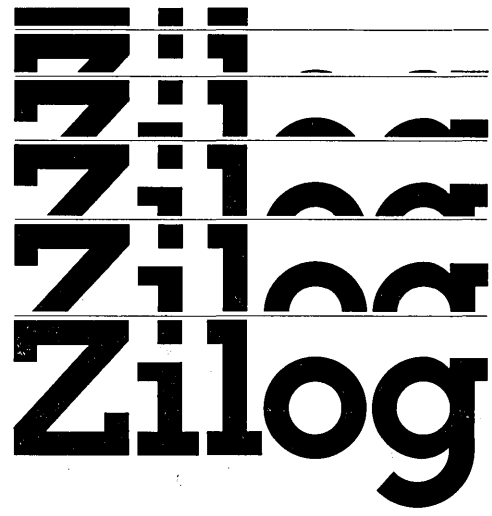
Legend:
R = 8-Bit Address
r = 4-Bit Address
R_1 or r_1 = Dst Address
R_2 or r_2 = Src Address

Sequence:
Opcode, First Operand, Second Operand

Note: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

I



Index

Zilog

READER COMMENTS

Your comments concerning this publication are important to us. Please take the time to complete this questionnaire and return it to Zilog.

Title of Publication: _____

Document Number: _____

Your Hardware Model and Memory Size: _____

Describe your likes/dislikes concerning this document:

Technical Information: _____

Supporting Diagrams: _____

Ease of Use: _____

Your Name: _____

Company and Address: _____

Your Position/Department: _____

No Postage
Necessary If
Mailed In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 35, CAMPBELL, CA.

POSTAGE WILL BE PAID BY:

Zilog

1315 Dell Ave.
Campbell, California 95008
ATTENTION: Corporate Publications

Zilog Sales Offices and Technical Centers

West

Sales & Technical Center
Zilog, Incorporated
1315 Dell Avenue
Campbell, CA 95008
Phone: (408) 370-8120
TWX: 910-338-7621

Sales & Technical Center
Zilog, Incorporated
18023 Sky Park Circle
Suite J
Irvine, CA 92714
Phone: (714) 549-2891
TWX: 910-595-2803

Sales & Technical Center
Zilog, Incorporated
15643 Sherman Way
Suite 430
Van Nuys, CA 91406
Phone: (213) 989-7485
TWX: 910-495-1765

Sales & Technical Center
Zilog, Incorporated
1750 112th Ave. N.E.
Suite D161
Bellevue, WA 98004
Phone: (206) 454-5597

Midwest

Sales & Technical Center
Zilog, Incorporated
951 North Plum Grove Road
Suite F
Schaumburg, IL 60195
Phone: (312) 885-8080
TWX: 910-291-1064

Sales & Technical Center
Zilog, Incorporated
28349 Chagrin Blvd.
Suite 109
Woodmere, OH 44122
Phone: (216) 831-7040
FAX: 216-831-2957

South

Sales & Technical Center
Zilog, Incorporated
4851 Keller Springs Road,
Suite 211
Dallas, TX 75248
Phone: (214) 931-9090
TWX: 910-860-5850

Zilog, Incorporated
7113 Burnet Rd.
Suite 207
Austin, TX 78757
Phone: (512) 453-3216

East

Sales & Technical Center
Zilog, Incorporated
Corporate Place
99 South Bedford St.
Burlington, MA 01803
Phone: (617) 273-4222
TWX: 710-332-1726

Sales & Technical Center
Zilog, Incorporated
240 Cedar Knolls Rd.
Cedar Knolls, NJ 07927
Phone: (201) 540-1671

Technical Center
Zilog, Incorporated
3300 Buckeye Rd.
Suite 401
Atlanta, GA 30341
Phone: (404) 451-8425

Sales & Technical Center
Zilog, Incorporated
1442 U.S. Hwy 19 South
Suite 135
Clearwater, FL 33516
Phone: (813) 535-5571

Zilog, Incorporated
613-B Pitt St.
Cornwall, Ontario
Canada K6J 3R8
Phone: (613) 938-1121

United Kingdom

Zilog (U.K.) Limited
Zilog House
43-53 Moorbridge Road
Maidenhead
Berkshire, SL6 8PL England
Phone: 0628-39200
Telex: 848609

France

Zilog, Incorporated
Cedex 31
92098 Paris La Defense
France
Phone: (1) 334-60-09
TWX: 611445F

West Germany

Zilog GmbH
Eschenstrasse 8
D-8028 TAUFKIRCHEN
Munich, West Germany
Phone: 89-612-6046
Telex: 529110 Zilog d.

Japan

Zilog, Japan K.K.
Konparu Bldg. 5F
2-8 Akasaka 4-Chome
Minato-Ku, Tokyo 107
Japan
Phone: (81) (03) 587-0528
Telex: 2422024 A/B: Zilog J