Binary floating-point format is also called binary real form. The 68881/68882 supports binary floating-point form which contains three fields. These are a sign, biased exponent and a significand. The 68881/68882 operates on these sizes, namely 32-bit single-precision, 64-bit double precision, and 96-bit extended precision.

For single precision (bit 31: sign bit, bits 23-30: 8-bit exponent, bits 0-22: 23-bit significand), 64-bit double precision (bit 63: sign bit, bits 62-52: 11-bit exponent, bits 0-51: 52-bit significand), and 96-bit extended-precision (bit 95: sign bit, bits 80-94: 15-bit exponent, bits 64-79: zero; sixteen unused bits, bits 0-63: 64-bit significand).

The biased exponent is used. The Single Precision adds a bias of 127_{10} ($7F_{16}$), double Precision uses a bias of 1023_{10} ($3FF_{16}$), and the extended-Precision uses a bias of 16383_{10} ($3FFFF_{16}$). The bias is added to the exponent before it is stored in this format and subtracted to convert to a true exponent when the number is interpreted.

A few special cases that do not conform to the floating-point form are also handled by the 68881/68882. For example, a zero is represented with all bits of the exponent and significand as zeros. The sign bit may be a one or a zero representing +0 or -0. The infinity, on the other hand is represented by all bits in the exponent and significand set to ones. The sign bit may be zero or one representing positive or negative infinity. The 68881/68882 supports BCD floating point form which represent each number as normalized significand raised to a power of 10. This format stores a number as 96 bits. The least significant 64 bits (8 bytes) contain the 16-digit BCD fraction. The next byte contains the whole number portion of the significand (0-9). The most significant bit (bit 95) contains the sign of significand while bit 94 includes the sign of the exponent. The exponent is represented by three digit BCD packed exponent (000-999) in bits 80–91. Similar to the 80387, the 68881/68882 also represents NaN's and also provides exceptions for signaling NaN's.

IEEE has established the standard for floating-point arithmetic specified by ANSI-IEEE754-1985. Typical 32-bit microprocessors use this standard.

1.3 Microcomputer Hardware

In this section, some unique features associated with various microcomputer components will be described.

The microcomputer contains a microprocessor, a memory unit, and an input/output unit. These elements are explained in the following in detail. Figure 1.1 shows a simplified block diagram of a microcomputer.

1.3.1 The System Bus

The system bus contains three buses. These are the address bus, the data bus, and the control bus. These buses connect the microprocessor to each of the memory and I/O elements so that information transfer between the microprocessor and any of the other elements can take place.

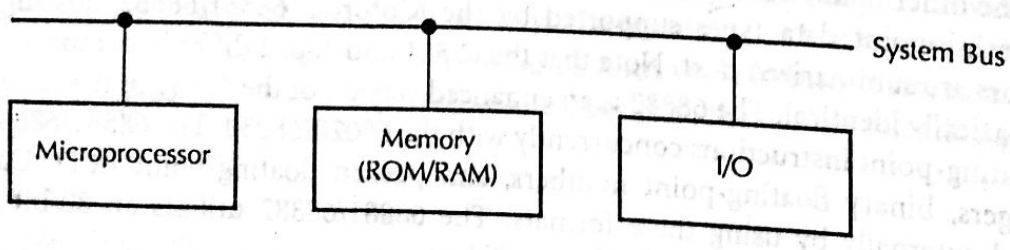


FIGURE 1.1 Simplified block diagram of a microprocessor.

On the address bus, information transfer normally takes place only in one direction, from the microprocessor to the memory or I/O elements. Therefore, this is called a unidirectional bus. This bus is usually 16 to 32 bits wide. The number of unique addresses that the microprocessor can generate on this bus depends on the width of this bus. For example, for a 16-bit address bus, the microprocessor can generate $2^{16} = 65{,}536$ different possible addresses. A different memory location or an I/O element can be represented by each one of these addresses.

The data bus is a bidirectional bus, that is, information can flow in both directions, to or from the microprocessor. This bus is normally 8, 16, or 32 bits wide.

The control bus is used to transmit signals that are used to synchronize the operation of the individual microcomputer elements. Typical control signals include READ, WRITE, and RESET. Some signals on the control bus such as interrupt signals are unidirectional, while some others such as RESET may be bidirectional.

1.3.2 The Microprocessor

The commercial microprocessor, fabricated using the MOS technology, is normally contained in a single chip. The microprocessor is comprised of a register section, one or more ALUs (Arithmetic Logic Units), and a control unit. Depending on the register section, the microprocessor can be classified either as an accumulator-based or a general-purpose register-based machine.

In an accumulator-based microprocessor such as the Intel 8085 and Motorola 6809, one of the operands is assumed to be held in a special register called the "accumulator". All arithmetic and logic operations are performed using this register as one of the data sources. The result after the operation is stored in the accumulator. One-address instructions are very predominant in this organization. Eight-bit microprocessors are usually accumulator-based.

The general-purpose register-based microprocessor is usually popular with 16- and 32-bit microprocessors, such as Intel 8086/80386/80486 and Motorola 68000/68020/68030/68040, and is called general-purpose, since its registers can be used to hold data, memory addresses, or the results of arithmetic or logic operations. The number, size, and types of registers vary from one microprocessor to another. Most registers are general-purpose registers, while some are provided with dedicated functions.

Typical dedicated registers include the Program Counter (PC), the Instruction Register (IR), Status Register (SR), the Stack Pointer (SP) and the Index Register. The 32-bit microprocessors include special on-chip combinational network called the Barrel Shifter.

The PC normally contains the address of the next instruction to be executed. Upon activating the microprocessor chip's RESET input, the PC is normally initialized with the address of the first instruction. For example, the 80486, upon hardware reset, reads the first instruction from the 32-bit address $FFFFFFF0_{16}$. In order to execute the instruction, the microprocessor normally places the PC contents on the address bus and reads (fetches) the first instruction from external memory. The program counter contents are then automatically incremented by the ALU. The microcomputer thus executes a program sequentially unless it encounters a jump or branch instruction. The size of the PC varies from one microprocessor to another depending on the address size. For example, the 68000 has a 24-bit PC, while the 68040 contains a 32-bit PC.

The instruction register (IR) contains the instruction to be executed. After fetching an instruction from memory, the microprocessor places it in the IR for translation.

The status register contains individual bits with each bit having a special meaning. The bits in the status register are called flags. Each flag is usually set or reset by an ALU operation. The flags are used by the Conditional Branch instructions. Typical flags include carry, sign, zero, and overflow.

The carry (C) flag is used to reflect whether or not an arithmetic operation such as ADD generates a carry. The carry is generated out of the 8th bit (bit 7) for byte operations, 16th bit (bit 15) for 16-bit, or 32nd bit (bit 31) for 32-bit operations. The carry is used as the borrow flag for subtraction. In multiple word arithmetic operations, any carry from a low-order word must be reflected in the high-order word for correct results.

The zero (Z) flag is used to indicate whether the result of an arithmetic or logic operation is zero. Z = 1 for a zero result and Z = 0 for a non-zero result. The sign flag (sometimes also called the negative flag) indicates whether a number is positive or negative. S = 1 indicates a negative number if the most significant bit of the number is one; S = 0 indicates a positive number if the most significant bit of the number is zero.

The overflow (V) flag is set to one if the result of an arithmetic operation on signed (two's complement) numbers is too large for the microprocessor's maximum word size; the C flag is overflow for unsigned numbers. The overflow flag for signed 8-bit numbers can be shown as $V = C7 \oplus C6$, where C7 is the final carry and C6 is the previous bit's carry. The $\oplus$ symbol indicates exclusive-OR operation. This can be illustrated by the numerical examples shown below:

```
          0000 0100          04(10)
        + 0000 0010        + 02(10)
        ---------------    ---------
C7= 0 <-0000 0110          06(10)
        |
C6= 0 <-0   6(16)
```

From the above, the result is correct when C6 and C7 have the same values (0 in this case). When C6 and C7 are different, an overflow occurs. For example, consider the following:

```
          1011 1110          - 66(10)
          1011 1111          - 65(10)
          ---------          ---------
          0111 1101          + 125(10(?))
                             result is
C7 = 1 <-                    incorrect
        |
C6 = 0 <-
```

The result is incorrect. Since $V = C6 \oplus C7 = 0 \oplus 1 = 1$, the overflow flag is set. Note that this applies to signed two's complemented numbers only.

The stack pointer (SP) register addresses the stack. A stack is Last-In First-Out (LIFO) read/write memory in the sense that items that go in last will come out first. This is because stacks perform all read (POP) and write (PUSH) operations from one end.

The stack is addressed by a register called the stack pointer (SP). The size of the SP is dependent on the microprocessor's address size. The stack is normally used by subroutines or interrupts for saving certain registers such as the program counter.

Two instructions, PUSH (stack write) and POP (stack read), can usually be performed by the programmer to manipulate the stack. If the stack is accessed from the top, the stack pointer is decremented before a PUSH and incremented after a POP. On the other hand, if the stack is accessed from the bottom, the SP is incremented before a PUSH and decremented after a POP. Typical microprocessors access the stack from the top. Depending on the microproces-

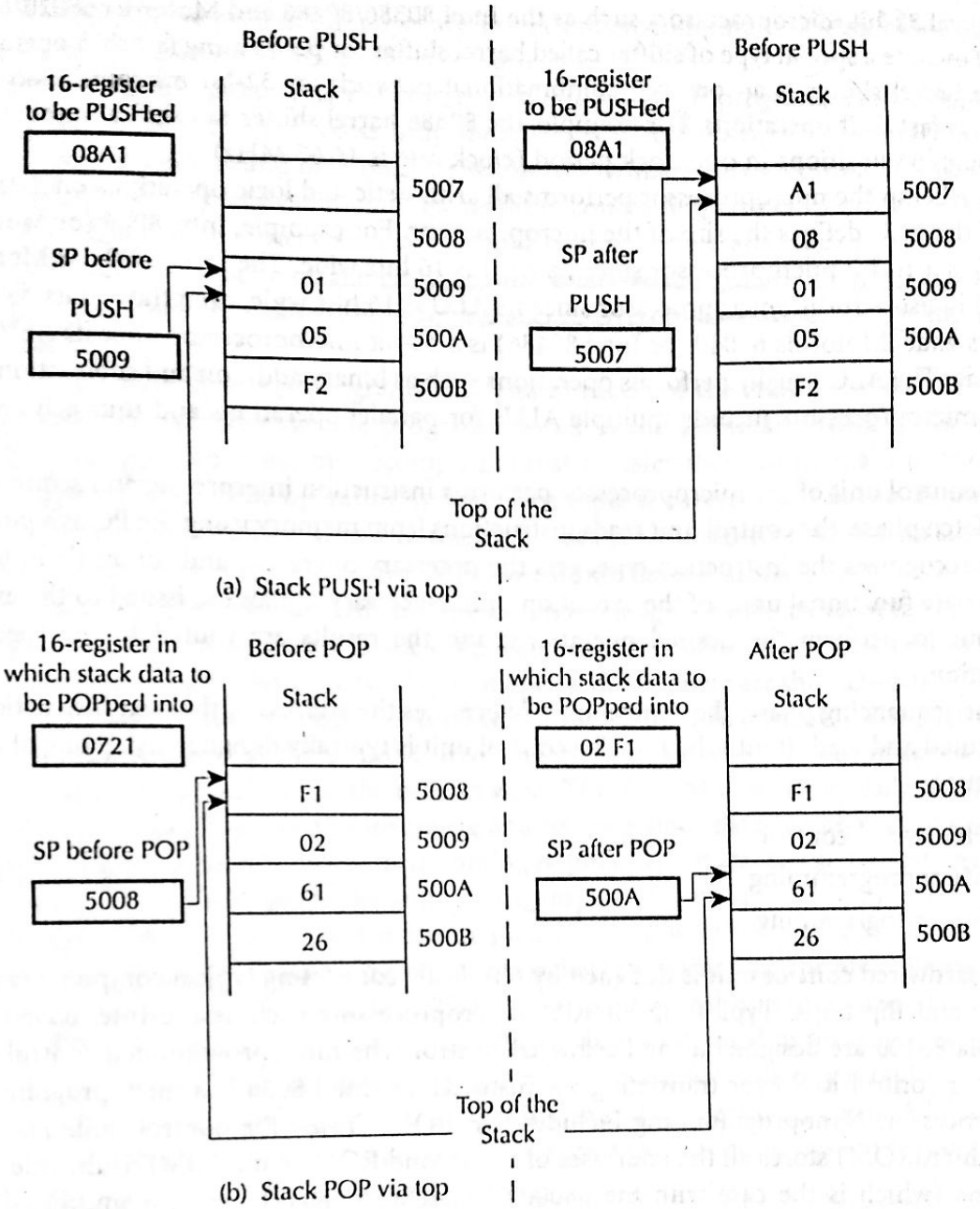


FIGURE 1.2 PUSH and POP operations via top.

sor, an 8-, 16-, or 32-bit register can be pushed onto or popped from the stack. The value by which the SP is incremented or decremented after POP or PUSH operations depends on the register size. For example, values of one for an 8-bit register, two for 16-bit registers, and four for 32-bit registers are used. Figure 1.2 shows the stack data when accessed from the top before and after PUSHing a 16-bit register onto the stack or POPping 16 bits from the stack into the 16-bit register. Note that stack items PUSHed must be POPped in reverse order. The item pushed last must be popped first.

Consider the PUSH operation in Figure 1.2a when the stack is accessed from the top. The SP is decremented by 2 after the PUSH. The SP is decremented since it is accessed from the top. A decrement value of 2 is used since the register to be pushed is 16 bits wide.

The POP operation shown in Figure 1.2b is the reverse of the PUSH. The SP is incremented after POP. The contents of locations 5008_{16} and 5009_{16} are assumed to be empty.

An index register is typically used as a counter for an instruction or for general storage functions. The index register is useful with instructions where tables or arrays of data are accessed. The general-purpose register-based microprocessor can use any general-purpose register as the index register.

Typical 32-bit microprocessors such as the Intel 80386/80486 and Motorola 68020/68030/68040 include a special type of shifter called barrel shifter for performing fast shift operations. The barrel shifter is an on-chip combinational network for 32-bit microprocessors and provides fast shift operations. For example, the 80386 barrel shifter can shift a number from 0 through 64 positions in one clock period (clock rate is 16.67 MHz).

The ALU in the microprocessor performs all arithmetic and logic operations on data. The size of the ALU defines the size of the microprocessor. For example, Intel 8086 (or Motorola 68000) is a 16-bit microprocessor since its ALU is 16 bits wide. The Intel 8088 (or Motorola 68008) is also a 16-bit microprocessor since its ALU is 16 bits wide, even though its data bus is 8 bits wide. Motorola 68040 (or Intel 80486) is a 32-bit microprocessor since its ALU is 32 bits wide. The ALU usually performs operations such as binary addition and subtraction. The 32-bit microprocessors include multiple ALUs for parallel operations and thus achieve fast speed.

The control unit of the microprocessor performs instruction interpreting and sequencing. In the fetch phase, the control unit reads instructions from memory using the PC as a pointer. It then recognizes the instruction type, gets the necessary operands, and routes them to the appropriate functional units of the execution unit. Necessary signals are issued to the execution unit to perform the desired operations, and the results are routed to the specified destination.

In the sequencing phase, the control unit determines the address of the next instruction to be executed and loads it into the PC. The control unit is typically designed using one of three techniques:

- Hardwired control
- Microprogramming
- Nanoprogramming

The hardwired control unit is designed by physically connecting typical components such as gates and flip-flops. Typical 32-bit RISC microprocessors such as the Intel 80960 and Motorola 88100 are designed using hardwired control. The microprogrammed control unit includes a control ROM for translating the instructions. Intel 8086 is a microprogrammed microprocessor. Nanoprogramming includes two ROMs inside the control unit. The first ROM (microROM) stores all the addresses of the second ROM (nanoROM). If the microinstructions (which is the case with the 68000/68020/68030/68040) repeat many times in a microprogram, use of two-level ROMs provides tremendous memory savings. This is the reason that the control units of the 68000, 68020, 68030, and 68040 are nanoprogrammed.

1.3.3 Memory Organization

1.3.3.a Introduction

A memory unit is an integral part of any microcomputer system and its primary purpose is to hold programs and data.

In a broad sense, a microcomputer memory system can be logically divided into three groups:

- Processor memory
- Primary or main memory
- Secondary memory

Processor memory refers to the microprocessor registers. These registers are used to hold temporary results when a computation is in progress. Also, there is no speed disparity between these registers and the microprocessor because they are fabricated using the same technology.

However, the cost involved in this approach forces a microcomputer architect to include only a few registers (usually 8 or 16) in the microprocessor.

Primary or main memory is the storage area in which all programs are executed. The microprocessor can directly access only those items that are stored in primary memory. Therefore, all programs and data must be within the primary memory prior to execution.

Secondary memory refers to the storage medium comprising slow devices such as magnetic tapes and disks. These devices are used to hold large data files and huge programs such as compilers and data base management systems which are not needed by the processor frequently. Sometimes secondary memories are also referred to as auxiliary or backup store or virtual memory.

Secondary memory stores programs and data in excess of the main memory. The microcomputer cannot directly execute programs stored in the secondary memory. In order to execute these programs, the microcomputer must transfer them to its main memory by a system program called the operating system. This topic is covered later in the chapter.

Data in disk memories are stored in tracks. A track is a concentric ring of data stored on a surface of a disk. Each track is further subdivided into several sectors. Each sector typically stores 512 or 1024 bytes of data. All disk memories use magnetic media except the optical disk memory which stores data on a plastic disk. Data is read or sometimes written on the optical disk with a laser beam. There are two types of optical disks. These are the CD-ROM (Compact Disk Read Only Memory) and the WORM (Write Once Read Many). The CD-ROM is inexpensive compared to the WORM drive. However it suffers from lack of speed and has limited software applications at the present time. The WORM drive is typically used in huge data storing applications such as insurance and banking since data can be written only once. The optical disk memory is currently becoming popular with microcomputer systems. One of the commonly used disk memories with microcomputer systems is the floppy or flexible disk. The floppy disk is a flat, round piece of plastic coated with magnetically sensitive oxide material. The disk is provided with a protective jacket to prevent fingerprints or foreign matter from contaminating the disk's surface. The floppy disk is available in three sizes. These are the 8 inch, 5.25 inch, and 3.5 inch. The 8 inch floppy disk is not used in present systems. These days, the 5.25 inch and 3.5 inch are very popular. Also, the 3.5 inch floppy is replacing the 5.25 inch floppy in newer systems since it is smaller in size and does not bend easily. All floppy disks are provided with an off-center index hole that allows the electronic system reading the disk to find the start of a track and the first sector.

Hard disk memory is also frequently used with microcomputer systems. The hard disk, also known as the fixed disk, is not removable like the floppy disk.

A comparison of the some of the features associated with the hard disk and floppy disk is provided below:

Characteristic	Hard Disk	Floppy Disk
Size	5 Mbytes to several Gbytes	1.2 Mbytes typical for 5.25 inch floppy. 1.44 Mbytes typical for 3.5 inch floppy.
Rotational Speed	3600 rpm	300 rpm
Number of heads	May have up to 8 disk surfaces with up to two heads per surface	Two heads; One head for the upper surface and the other head for the lower surface

Primary memory normally includes ROM (Read-only Memory) and RAM (Random Access Memory). As the name implies, a ROM permits only a read access. Some ROMs are custom made, that is, their contents are programmed by the manufacturer. Such ROMs are called mask programmable ROMs. Sometimes a user may have to program a ROM in the field. For instance, in a fusible-link ROM, programmable read-only memory (PROM) is available. The main disadvantage of a PROM is that it cannot be reprogrammed.

Some ROMs can be reprogrammed, these are called Erasable Programmable Read-Only Memories (EPROMs).

In an EPROM, programs are entered using electrical impulses and the stored information is erased by using ultraviolet rays. Usually an EPROM is programmed by inserting the EPROM chip into the socket of a PROM programmer and providing program addresses and voltage pulses at the appropriate pins of the chip. Typical erase times vary between 10 and 30 minutes.

With advances in IC technology, it is possible to achieve an electrical means of erasure. These new ROMs are called Electrically Alterable ROMs (EAROMs) or Electrically Erasable PROMs (EEPROMs or E²PROMs) and these ROM chips can be programmed even when they are in the circuit board. These memories are also called Read Mostly Memories (RMMs), since they have much slower write times than read times. Random Access Memories (RAMs) are read/write memories.

Information stored in random access memories will be lost if the power is turned off. This property is known as volatility and, hence, RAMs are usually called volatile memories. RAMs can be backed up by batteries for a certain period of time and are sometimes called nonvolatile RAMs. Stored information in a magnetic tape or magnetic disk is not lost when the power is turned off. Therefore, these storage devices are called nonvolatile memories. Note that a ROM is a nonvolatile memory.

Some RAMs are constructed using bipolar transistors, and the information is stored in the form of voltage levels in flip-flops. These voltage levels do not usually drift away, or decay. Such memories are called static RAMs because the stored information remains constant for some period of time.

On the other hand, in RAMs that are designed using MOS transistors, the information is held in the form of electrical charges in capacitors. Here, the stored charge has the tendency to decay. Therefore, a stored 1 would become a 0 if no precautions were taken. These memories are referred to as dynamic RAMs. In order to prevent any information loss, dynamic RAMs have to be refreshed at regular intervals. Refreshing means boosting the signal level and writing it back. This activity is performed by a hardware unit called "refresh logic" which can either be a separate chip or is contained in the microprocessor chip.

Since the static RAM maintains information in active circuits, power is required even when the chip is inactive or in standby mode. Therefore, static RAMs require large power supplies. Also, each static RAM cell is about four times larger in area than an equivalent dynamic cell; a dynamic RAM chip contains about four times as many bits as a static RAM chip using the same or comparable semiconductor technology. Figure 1.3 shows the subcategories of ROMs, RAMs, and their associated technologies.

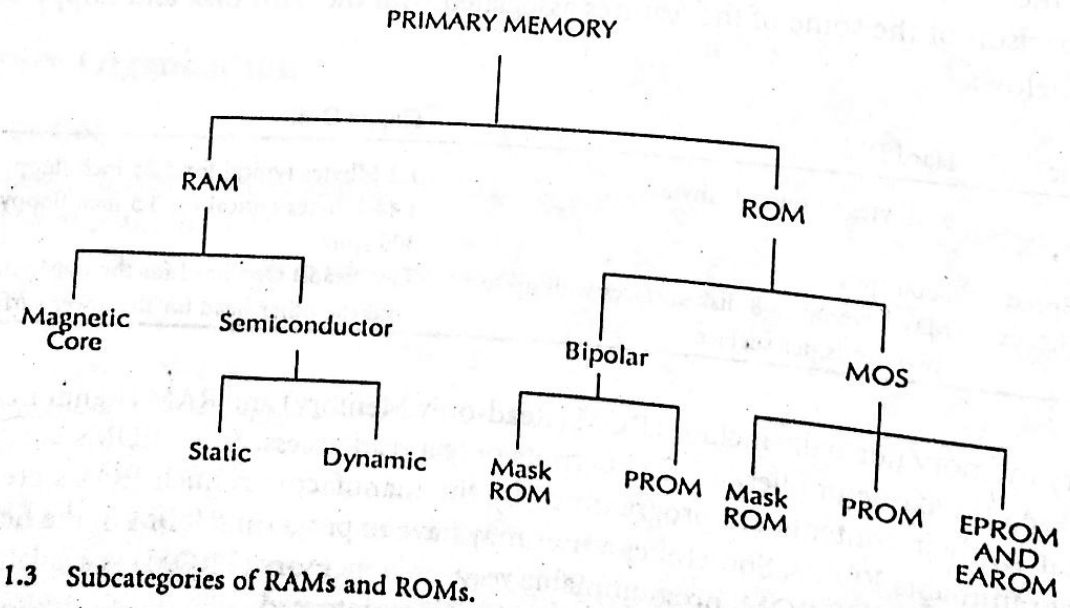


FIGURE 1.3 Subcategories of RAMs and ROMs.

Today, one megabit of data can be stored in an ordinary dynamic RAM chip. The data can be accessed in 80 nanoseconds or less. The RAM chip costs $5. In contrast, it takes 150 nanoseconds to access a one-megabit EEPROM which costs $150. Sixteen megabit DRAMs are very popular these days at a price of approximately 0.3 millicent per bit. Recently, IBM, Hitachi, Toshiba and others have introduced 64 mega-bit DRAMs. It is expected that giga-bit DRAMs will not be introduced until the next century.

In the mid 1980s, Toshiba Semiconductor invented flash memory. About the same time, Intel and Seeq Semiconductor were also working on flash memories. While each manufacturer implemented its flash memory differently, they operate in a similar way.

Like EPROMs and EEPROMs or EAROMs, flash memory is nonvolatile and reprogrammable. Flash memory is fabricated by using ETOX II (EPROM Tunnel Oxide) technology which is a combination of EPROM and EEPROM technologies. Flash memory is relatively inexpensive compared to EEPROM. A one megabit flash memory costs about $15. Flash memory can be reprogrammed electrically while embedded in the board. However, one can only change a sector or a block (consisting of multiple bytes) at a time.

Flash memory cells contain a single transistor like the EPROM cell. In contrast, a DRAM cell typically contains a transistor and a capacitor, an EEPROM cell contains two transistors while a static RAM cell requires four or six transistors.

The non-volatility and DRAM-like speed of flash memory are ideal for solid-state "disk" drives. Flash based disks do not have any disks or moving parts. Flash disks are very fast compared to most available disk drives.

Data can be accessed in 120 nanoseconds in flash memories while it takes 15 to 30 milliseconds to access data stored in today's typical hard disk. However, flash disks are limited to up to 40 megabytes in capacity whereas hard disk drives can store from 5 megabytes to several gigabytes.

A flash disk can be built from one or more flash-memory IC chips and some controlling logic devices. For example, to build a 512Kbyte flash disk, four one-megabit flash memory chips can be connected on a small card. An example of such a flash memory system is the Intel iMC004FLKA 4 Megabyte flash memory card. In addition to flash-disk hardware, software to manage files on a flash disk is required. The file system software handles creating and deleting files, changing the file sizes and formatting the flash disk. Microsoft offers flash file system software for the MS-DOS operating system.

The most severe limitation of flash disks has been its cost.However, the cost of flash ROM is significantly decreasing. In the future, high density flash memory is expected to be available at an inexpensive cost.

Flash memory can be programmed using either 5V or 12V. The 5V feature becomes more desirable for portable equipment wh[illegible]e no 12V power is available. The speed, rugged construction, and lower power consumption of flash disks is ideal for laptop and notebook computers.

In summary, due to the high cost of flash disks, desktop computers will continue to use hard disk drives. Since flash memory combines the advantages of an EPROM's low cost with an EEPROM's ease of reprogramming, flash memories are being extensively used these days as a microcomputer's main non-volatile memory. An example of flash memory is the Intel 28F020 256K x 8 flash memory. By 1997, the cost of a megabyte of flash memory is expected to move from its current level of $120 to about $5. At that time, flash disks will be able to replace hard disks in many applications.

1.3.3.b Main Memory Array Design

In many applications, a memory of large capacity is often realized by interconnecting several small-size memory blocks. In this section, design of a large main memory using small-size memories as building blocks is presented. The memory map defining all memory addresses is determined. Note that the microprocessor's reset vector must be included in the memory map.

There are three types of techniques used for designing the main memory. These are linear decoding, full decoding/partial decoding and memory decoding using PALs. We will illustrate the concepts associated with these techniques in the following.

First, consider the block diagram of a typical static RAM chip shown in Figure 1.4.

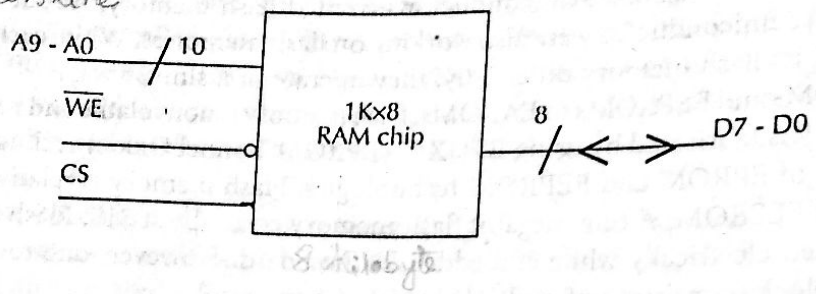


FIGURE 1.4 Typical Static RAM Chip.

The capacity of this chip is 8192 bits and these bits are organized as 1024 words with 8 bits/word. Each word has a unique address and this is specified on 10-bit address lines A9—A0 (note that $2^{10} = 1024$). The inputs and outputs are routed through the 8-bit bidirectional data lines D7 through D0. The operation of this chip is governed by the two control inputs: $\overline{WE}$ (write Enable) and CS (chip select). The truth table that describes the operation of this chip is shown in Table 1.1.

TABLE 1.1 Truth Table for 1K × 8 Static RAM

CS	$\overline{WE}$	MODE	Status of D7—D0	Power
L	X	Not selected	High impedance	Standby
H	L	Write	Acts as an input bus	Active
H	H	Read	Acts as an output bus	Active

Note: H — high, L — low, X — don't care.

From this table, it is easy to see that when CS input is low, the chip is not selected and thus the lines D7 through D0 are driven to the high impedance state. When CS = 1 and $\overline{WE}$ is LOW, data on lines D7—D0 are written into the word addressed by A0 through A9. Similarly, when CS = 1 and WE is high, the contents of the memory word (whose address is specified on address lines A9 through A0) will appear on lines D7 through D0. Note that when the chip select input CS goes to low, the device is disabled and the chip automatically reduces its power requirements and remains in this low-power standby mode as long as CS remains low. This feature results in system power savings as high as 85% in larger systems, where the majority of devices are disabled.

1.3.3.b.i Linear Decoding. This technique uses the unused address lines of the microprocessor as chip selects for the memory chips. This method is used for small systems.

A simple way to connect an 8-bit microprocessor to a 6K RAM system using linear decoding is shown in Figure 1.5. In this approach, the address lines A9 through A0 of the microprocessor are used as a common input to each 1K × 8 RAM chip. The remaining 6 high-order lines are used to select one of the 6 RAM chips. For example, if A15A14A13A12A11A10 = 000010, then the RAM chip 1 is selected. The address map realized by this arrangement is summarized in Figure 1.6. This method is known as the linear select decoding technique. The principal advantage of this method is that it does not require any decoding hardware. However, this approach has some disadvantages:

- Although with a 16-bit address bus we have 64K bytes of RAM space, we are able to interface only 6K bytes of RAM. This means that this idea wastes address space.
- The address map is not contiguous; rather, it is sparsely distributed.
- If both A11 and A10 are high at the same time, both RAM chips 0 and 1 are selected and thus a bus conflict occurs. This can be avoided by proper programming to select the desired memory chip and deselect the others.
- Also, if all unused address lines are not utilized as chip selects for memory, then these unused pins become don't cares (can be 0 or 1). This results in foldback, meaning that a memory location will have its image in the memory map. For example, if A15 is don't care in design and if A14 to A0 address lines are used, then address 0000_{16} and address 8000_{16} are the same locations. This is called foldback and it wastes memory space.

1.3.3.b.ii Full/Partial Decoding. Difficulties such as the bus conflict and sparse address distribution are eliminated by the use of the full/partial decoded addressing technique. To see this, consider the organization shown in Figure 1.7. In this setup, we use a 2-to-4 decoder and interface the 8-bit microprocessor with 4K bytes of RAM. In particular, the four combinations of the lines A11 and A10 select the RAM chips as follows:

A11	A10	Device Selected
0	0	RAM chip 0
0	1	RAM chip 1
1	0	RAM chip 2
1	1	RAM chip 3

Also observe that this hardware makes sure that the memory system is enabled only when the lines A15 through A12 are zero. The complete address map corresponding to this organization is summarized in Figure 1.8.

1.3.3.b.iii Memory Decoding by using Programmable Array Logic (PAL). A Programmable Array Logic (PAL) is similar to a ROM in concept except that it does not provide full decoding of the input lines. Instead, a PAL provides a partial sum of products which can be obtained via programming and saves a lot of space on the board. The PAL chip contains a fused programmable AND array and a fixed OR array. Note that in PLA (Programmable Logic Array) both AND and OR arrays are programmable. The AND and OR gates are fabricated inside the PAL without interconnections. The specific functions desired are implemented during programming via software. Programming of the PAL provides connections of the inputs of the AND gates and the outputs of the AND gates to the inputs of the OR gates. Therefore, the PAL implements sum of products of the inputs. PALs are used extensively these days with 32-bit microprocessors such as the Intel 80386/80486 and Motorola 68030/68040 for performing the memory decode function. PALs connect these microprocessors to memory, I/O and other chips without the use of any additional logic gates or circuits.

Each input has both true and complemented forms. A look at the NOR gate output $\varnothing_1$ indicates that there are two X connections at the inputs of this NOR gate. The three X inputs are wire-ANDed together with programming the PAL so that $\varnothing_1 = \left(I_0 \cdot \overline{I_1}\right) + I_2$.

The PAL chips are usually identified by a two-digit number followed by a letter and then a digit. The two-digit number specifies the number of input lines while the last digit defines the number of output lines. The fixed number of AND gates are connected to either an OR or a NOR gate. The letter 'H' indicates that the output gates are OR gates. The letter 'L' is used when the outputs are NOR gates.

As an example, the 10H8 provides eight OR gate outputs driven by ten AND gate inputs. The 10L8, on the other hand, is the same as the 10H8 except that the eight output gates are NOR gates.

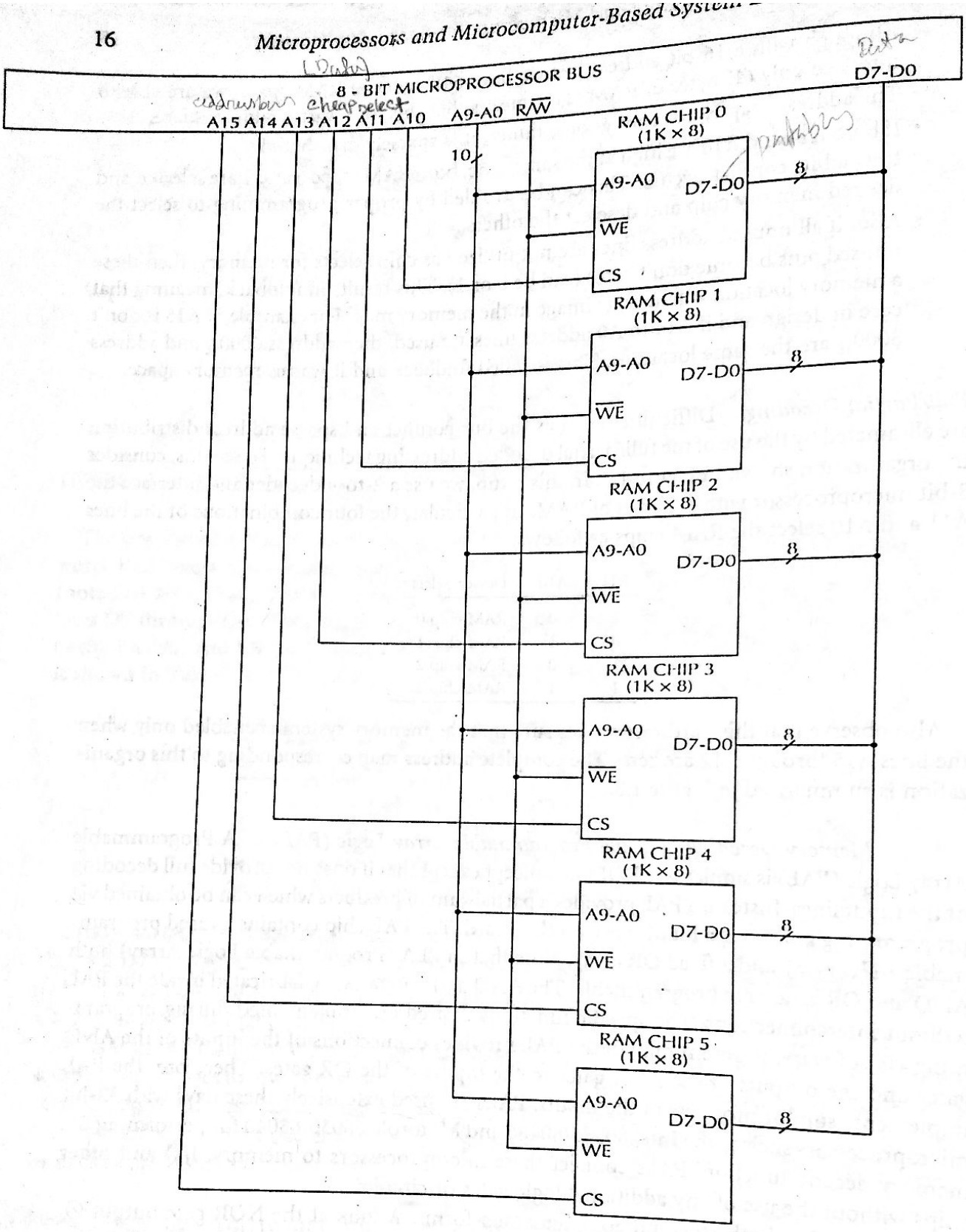


FIGURE 1.5 An 8-bit microprocessor interfaced to a 6K RAM system using the linear select decoding technique.

Some PALs provide additional features. The 16L8 includes tristate outputs. Each of the eight NOR gate outputs is driven internally by six AND gates. A seventh AND gate provides the enable signal for the tristate buffers.

The 16L8 is a popular PAL used with 32-bit microprocessors. The 16L8 is a 20-pin PAL manufactured using bipolar transistors. It has ten input pins (labeled I), two outputs (labeled O) and six programmable Input/Output (labeled I/O) lines. Using the programmable I/O lines, the number of input lines can be increased to a maximum of 16 and the number of output lines can be increased to 8.

Binary Address Pattern A15 A14 A13 A12 A11 A10 A9 A7 A6 A5 A4 A3 A2 A1 A0	Device Selected	Address Assignment in Hex
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1	RAM CHIP 0	0400 to 07FF
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1	RAM CHIP 1	0800 to 0BFF
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1	RAM CHIP 2	1000 to 13FF
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1	RAM CHIP 3	2000 to 23FF
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1	RAM CHIP 4	4000 to 43FF
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1	RAM CHIP 5	8000 to 83FF

FIGURE 1.6 Address map realized by the system shown in Figure 1.5.

Programming PALs can be accomplished by first creating a file by using a text editor on a personal computer. The file should include information such as the pin assignments of the PAL and the boolean equation for the outputs. By inserting the PAL into the programming module included with the personal computer, the PAL can then be programmed with the PAL programming software provided with the personal computer.Note that PAL programming hardware and software are sold separately and not usually included with a personal computer.

1.3.3.c Memory Management Concepts

Due to the massive amount of information that must be saved in most systems, the mass storage is often a disk. If each access is to a disk (even a hard disk), then system throughput will be reduced to unacceptable levels.

An obvious solution is to use a large and fast locally accessed semiconductor memory. Unfortunately the storage cost per bit for this solution is very high. A combination of both off-board disk (secondary memory) and on-board semiconductor main memory must be designed into a system. This requires a mechanism to manage the two-way flow of information between the primary (semiconductor) and secondary (disk) media. This mechanism must be able to transfer blocks of data efficiently, keep track of block usage, and replace them in a nonarbitrary way. The primary memory system must therefore be able to dynamically allocate memory space.

An operating system must have resource protection from corruption or abuse by users. Users must be able to protect areas of code from each other, while maintaining the ability to communicate and share other areas of code. All these requirements indicate the need for a device, located between the microprocessor and memory, to control accesses, perform address mappings, and act as an interface between the logical (programmer's memory) and microprocessor physical (memory) address spaces. Since this device must manage memory use, it is appropriately called the memory management unit (MMU). Typical 32-bit microprocessors such as Motorola 68030 and Intel 80386 include on-chip MMU.

The MMU reduces the burden of the memory management function on the operating system.

The basic functions provided by the MMU are address translation and protection. The MMU translates logical program addresses to physical memory addresses. The addresses in a

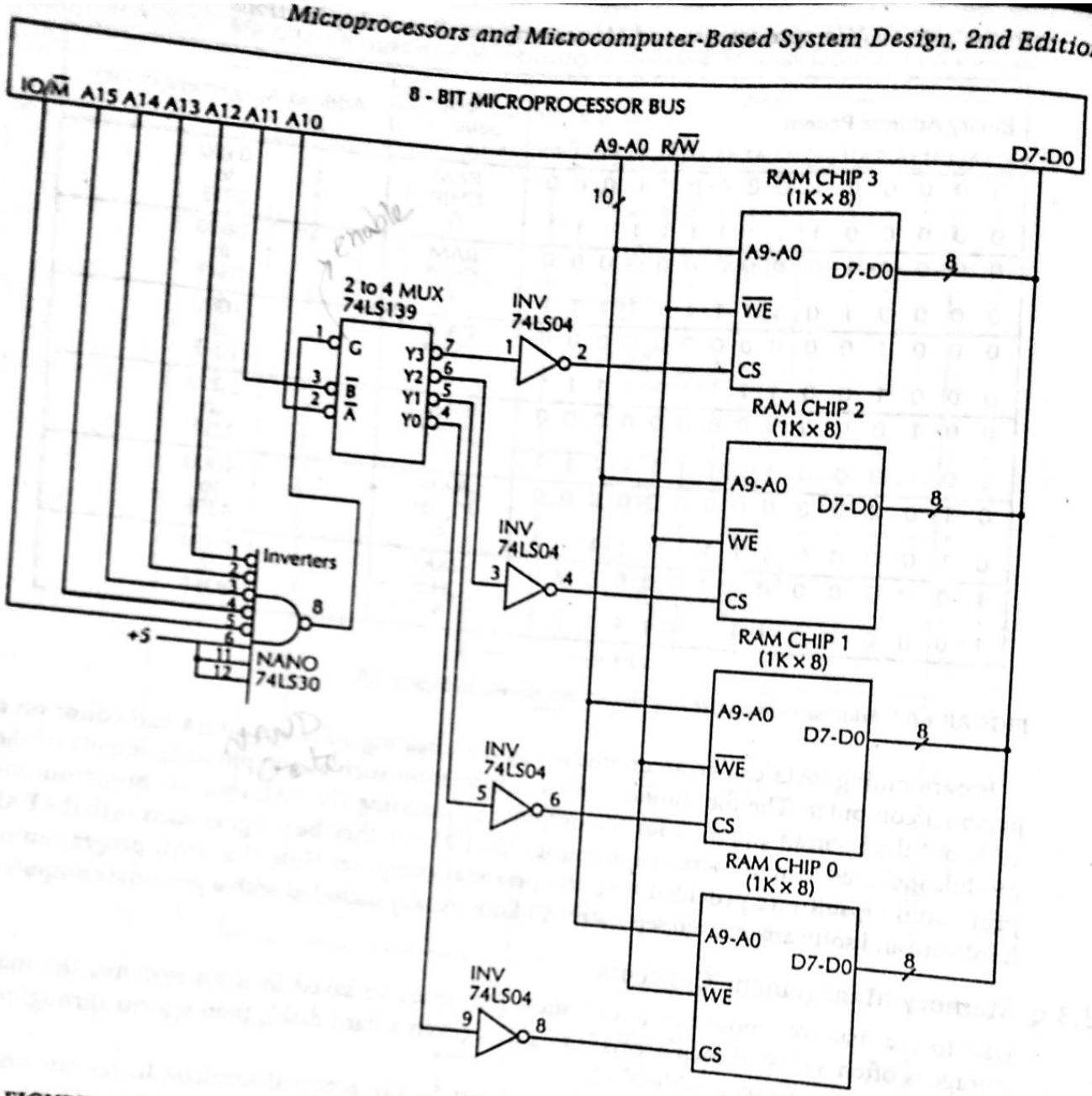


FIGURE 1.7 An 8-bit microprocessor interfaced to a 4K RAM system using a full/partial decoded addressing technique

Binary Address Pattern A15 A14 A13 A12 A11 A10 A9 A7 A6 A5 A4 A3 A2 A1 A0	Device Selected	Address Assignment in Hex
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1	RAM CHIP 0	0000 to 03FF
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1	RAM CHIP 1	0400 to 07FF
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1	RAM CHIP 2	0800 to 0BFF
0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1	RAM CHIP 3	0C00 to 0FFF

FIGURE 1.8 Address map corresponding to the organization shown in Figure 1.7.

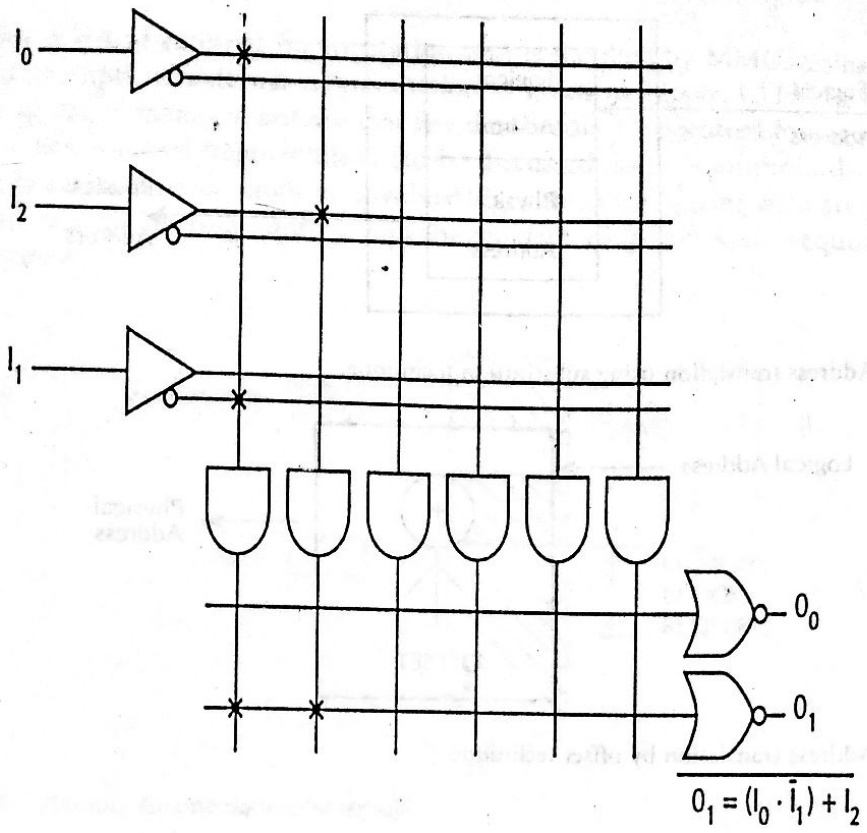


FIGURE 1.9 A typical PAL.

program are called logical addresses since they indicate the logical positions of instructions and data. The MMU translates these logical addresses to physical addresses provided by the memory chips. The MMU can perform address translation in one of two ways:

1. By using the substitution technique as shown in Figure 1.10a
2. By adding an offset to each logical address to obtain the corresponding physical address as shown in Figure 1.10b

Address translation using substitution is faster than the offset method. However, the offset method has the advantage of mapping a logical address to any physical address as determined by the offset value.

Memory is usually divided into small manageable units. The terms "page" and "segment" are frequently used to describe these units. Paging divides the memory into equal-sized pages, while segmentation divides the memory into variable-sized segments.

It is relatively easier to implement the address translation table if the logical and main memory spaces are divided into pages. The term "page" is associated with logical address space, while the term "block" usually refers to a page in main memory space.

There are three ways to map logical addresses to physical addresses. These are paging, segmentation, and combined paging/segmentation.

In a paged system, a user has access to a larger address space than physical memory provides. The virtual memory system is managed by both hardware and software. Note that memory in excess of the main memory such as floppy disk storage is called virtual memory. The hardware included in the memory management unit handles address translation. The memory management software in the operating system performs all functions including page replacement policies in order to provide efficient memory utilization. The memory management software performs functions such as removal of the desired page from main memory to accommodate a new page, transferring a new page from secondary to main memory at the right instant of time, and placing the page at the right location in memory.

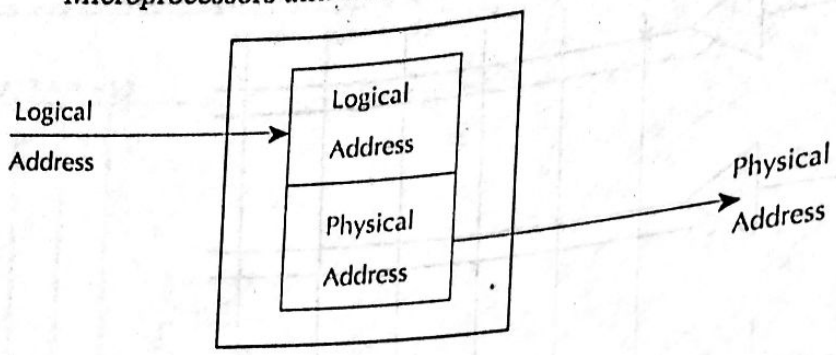


FIGURE 1.10a Address translation using substitution technique.

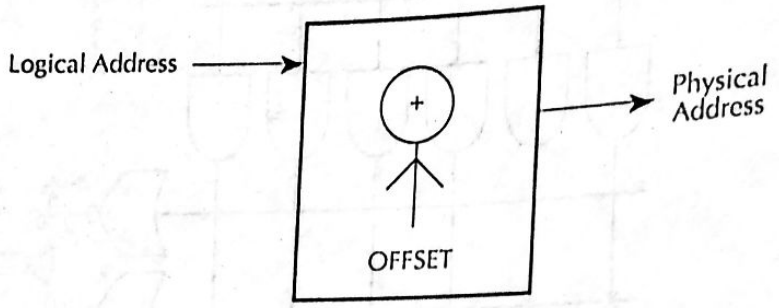


FIGURE 1.10b Address translation by offset technique.

If the main memory is full during transfer from secondary to main memory, it is necessary to remove a page from main memory to accommodate the new page. Two popular page replacement policies are first-in first-out (FIFO) and least recently used (LRU). The FIFO policy removes the page from main memory that has been resident in memory for the longest amount of time. The FIFO replacement policy is easy to implement. One of the main disadvantages of the FIFO policy is that it is likely to replace heavily used pages. Note that heavily used pages are resident in main memory for the longest amount of time. Sometimes this replacement policy might be a poor choice. For example, in a time-shared system, several users normally share a copy of the text editor in order to type and correct programs. The FIFO policy on such a system might replace a heavily used editor program page to make room for a new page. This program page might be recalled to main memory immediately. The FIFO, in this case, would be a poor choice.

The LRU policy, on the other hand, replaces that page which has not been used for the longest amount of time.

In the segmentation method, the MMU utilizes the segment selector to obtain a descriptor from a table in memory containing several descriptors. A descriptor contains the physical base address for a segment, the segment's privilege level, and some control bits. When the MMU obtains a logical address from the microprocessor, it first determines whether the segment is already in the physical memory. If it is, the MMU adds an offset component to the segment base component of the address obtained from the segment descriptor table to provide the physical address. The MMU then generates the physical address on the address bus for accessing the memory. On the other hand, if the MMU does not find the logical address in physical memory, it interrupts the microprocessor. The microprocessor executes a service routine to bring the desired program from a secondary memory such as disk to the physical memory. The MMU determines the physical address using the segment offset and descriptor as above and then generates the physical address on the address bus for memory. A segment will usually consist of an integral number of pages, say, each 256 bytes long. With different-sized segments being swapped in and out, areas of valuable primary memory can become unusable. Memory is unusable for segmentation when it is sandwiched between already allocated segments and if it is not large enough to hold the latest segment

loaded. This is called external fragmentation and is handled by MMUs using special techniques. An example of external fragmentation is given in Figure 1.11. The advantages of segmented memory management are that few descriptors are required for large programs or data spaces, and internal fragmentation (to be discussed later) is minimized. The disadvantages include external fragmentation, involved algorithms for placing data are required, possible restrictions on starting address, and longer data swap times are required to support virtual memory.

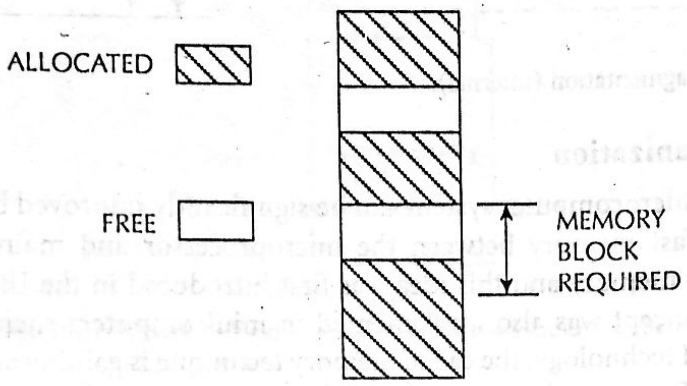


FIGURE 1.11 Memory fragmentation (external).

Address translation using descriptor tables offers a protection feature. A segment or a page can be protected from access by a program section of a lower privilege level. For example, the selector component of each logical address includes one or two bits indicating the privilege level of the program requesting access to a segment. Each segment descriptor also includes one or two bits providing the privilege level of that segment. When an executing program tries to access a segment, the MMU can compare the selector privilege level with the descriptor privilege level. If the segment selector has the same or higher privilege level, then the MMU permits the access. If the privilege level of the selector is lower than the descriptor, the MMU can interrupt the microprocessor informing of a privilege level violation. Therefore, the indirect technique of generating physical address provides a mechanism of protecting critical program sections in the operating system.

Paging divides the memory into equal-sized pages, it avoids the major problem of segmentation-external fragmentation. Since the pages are of the same size, when a new page is requested and an old one swapped out, the new one will always fit into the vacated space. However, a problem common to both techniques remains — internal fragmentation. Internal fragmentation is a condition where memory is unused but allocated due to memory block size implementation restrictions. This occurs when a module needs, say, 300 bytes and page is 1K bytes, as shown in Figure 1.12.

In the paged-segmentation method, each segment contains a number of pages. The logical address is divided into three components: segment, page, and word. The segment component defines a segment number, the page component defines the page within the segment, and the word component provides the particular word within the page. A page component of n bits can provide up to 2^n pages. A segment can be assigned with one or more pages up to a maximum of 2^n pages; therefore, a segment size depends on the number of pages assigned to it.

Protection mechanisms can operate either on physical address or logical address. Physical memory protection can be accomplished by using one or more protection bits with each block to define the access type permitted on the block. This means that each time a page is transferred from one block to another, the block protection bits must be updated. A more efficient approach is to provide a protection feature in logical address space by including protection bits in the descriptors of the segment table in the MMU.

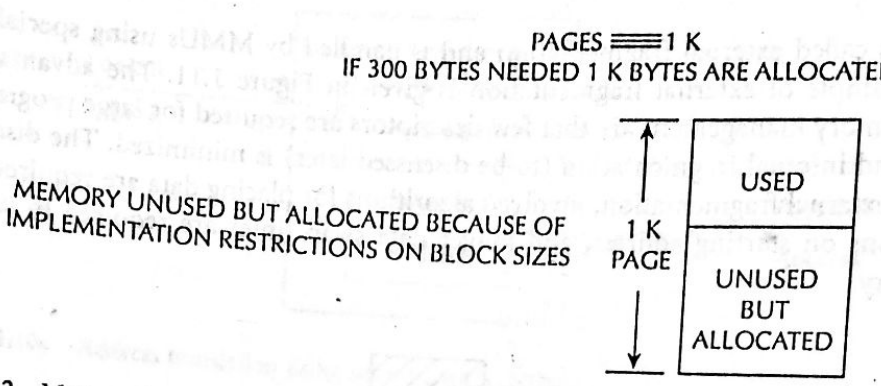


FIGURE 1.12 Memory fragmentation (internal).

1.3.3.d Cache Memory Organization

The performance of a microcomputer system can be significantly improved by introducing a small, expensive, but fast memory between the microprocessor and main memory. This memory is called cache memory and this idea was first introduced in the IBM 360/85 computer. Later on, this concept was also implemented in minicomputers such as PDP-11/70. With the advent of VLSI technology, the cache memory technique is gaining acceptance in the microprocessor world. For example, an on-chip cache memory is implemented in Intel's 32-bit microprocessor, the 80486, and Motorola's 32-bit microprocessors, the MC 68020/68030/68040. The 80386 does not have on-chip cache but external cache memory can be interfaced to it. Studies have shown that typical programs spend most of their execution times in loops. This means that the addresses generated by a microprocessor have a tendency to cluster around a small region in the main memory. This phenomenon is known as locality of reference. The 32-bit microprocessor can execute the same instructions in a loop from the on-chip cache rather than reading them repeatedly from the external main memory. Thus the performance offered by 32-bit microprocessors is greatly improved.

The block diagram representation of a microprocessor system that employs an on-chip cache memory is shown in Figure 1.13. Usually, a cache memory is very small in size and its access time is less than that of the main memory by a factor of 5.

Cache hit means that the reference is found in the cache and the data pertaining to the microprocessor reference is transferred to the microprocessor from the cache. However, if the reference is not found in the cache, we call it a cache miss. When there is a cache miss, the main memory is accessed by the microprocessor and the data are then transferred to the microprocessor from the main memory. At the same time, a block of data containing the desired data needed by the microprocessor is transferred from the main memory to the cache. The block normally contains 4 to 16 bytes, and this block is placed in the cache using the standard replacement policies such as FIFO (First-In First-Out) or LRU (Least Recently Used). This block transfer is done with a hope that all future references made by the microprocessor will be confined to the fast cache.

The relationship between the cache and main memory blocks is established using mapping techniques. Three widely used mapping techniques are

- Direct mapping
- Fully-associative mapping
- Set-associative mapping

In direct mapping, the main memory address is divided into two fields: an index field and a tag field. The number of bits in the index field is equal to the number of address bits required to access the cache memory.

Assume that the main memory address is m bits wide and the cache memory address is n bits wide. Then the index field requires n bits and the tag field is (m — n) bits wide. The n-

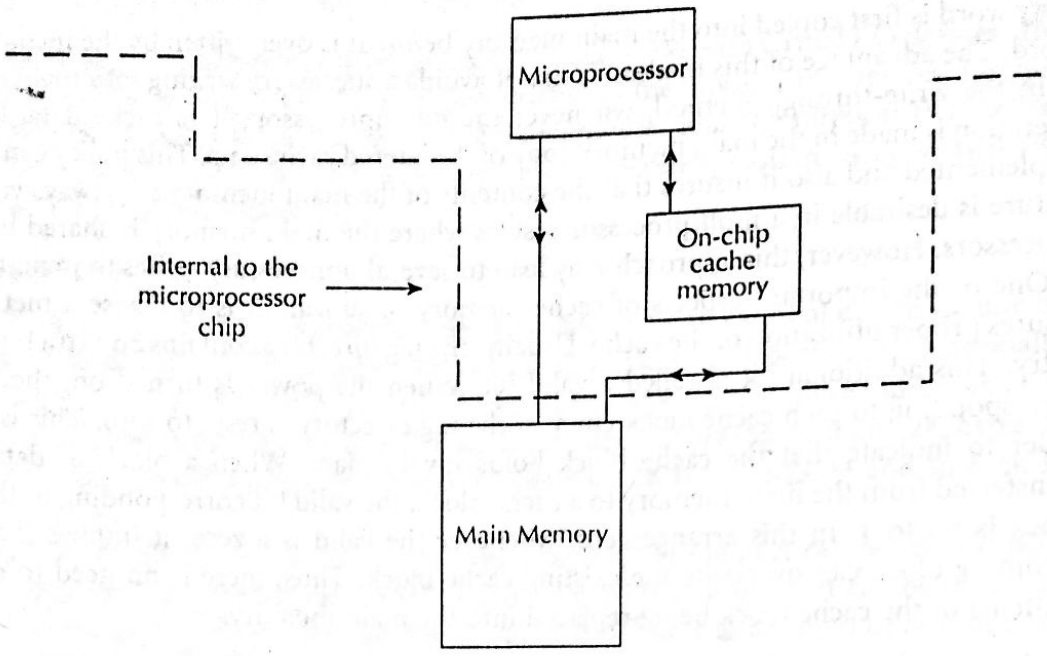


FIGURE 1.13 Memory organization of a computer system that employs a cache memory.

bit address accesses the code. Each word in the cache includes the data word and its associated tag. When the microprocessor generates an address for main memory, the index field is used as the address to access the cache. The tag field of the main memory is compared with the tag field in the word read from cache. A hit occurs if the tags match. This means that the desired data word is in cache. A miss occurs if there is no match, and then the required word is read from main memory. It is written in the cache along with the new tag. A random access memory is used as the cache memory.

One of the main drawbacks of direct mapping is that numerous misses may occur if two or more words with addresses having the same index but different tags are accessed several times. This can be minimized by incorporating a larger cache.

The fastest cache memory utilizes an associative memory. The method is known as fully associative mapping. Each associative memory content contains main memory address and its content (data). When the microprocessor generates a main memory address, it is compared associatively (simultaneously) with all addresses in the associative memory. If there is a match, the corresponding data word is read from the associative cache memory and sent to the microprocessor. If a miss occurs, the main memory is accessed, and the address and its corresponding data are written to the associative cache memory. If the cache is full, certain policies such as FIFO (first-in first-out) are used as replacement algorithm for the cache. The associative cache is expensive but provides fast operation.

The set-associative mapping is a combination of direct and associative mapping. Each cache word stores two or more main memory words using the same index address. Each main memory word consists of a tag and its data word. An index with two or more tags and data words forms a set. When the microprocessor generates a memory request, the index of the main memory address is used as the cache address. The tag field of the main memory address is then compared associatively (simultaneously) with all tags stored under the index. If a match occurs, the desired data word is read. If a miss occurs, the data word, along with its tag, is read from main memory and also written into the cache. The hit ratio improves as the set size increases. This is because more words with the same index but different tags can be stored in cache.

There are two ways of writing into cache: the write-back and write-through methods. In the write-back method, whenever the microprocessor writes something into a cache word, a dirty bit is assigned to the cache word. When a dirty word is to be replaced with a new word, the

dirty word is first copied into the main memory before it is overwritten by the incoming new word. The advantage of this method is that it avoids unnecessary writing into main memory.

In the write-through method, whenever the microprocessor alters cache data, the same alteration is made in the main memory copy of the altered cache data. This policy can be easily implemented and also it insures that the contents of the main memory are always valid. This feature is desirable in a multiprocessor system where the main memory is shared by several processors. However, this approach may lead to several unnecessary writes to main memory.

One of the important aspects of cache memory organization is to devise a method that insures proper utilization of the cache. Usually, the tag directory contains an extra bit for each entry. This additional bit is called a valid bit. When the power is turned on, the valid bit corresponding to each cache block entry of the tag directory is reset to zero. This is done in order to indicate that the cache block holds invalid data. When a block of data is first transferred from the main memory to a cache block, the valid bit corresponding to this cache block is set to 1. In this arrangement, whenever the valid is a zero, it implies that a new incoming block can overwrite the existing cache block. Thus, there is no need to copy the contents of the cache block being replaced into the main memory.

1.3.4 Input/Output (I/O)

This section describes the basic input and output techniques used by microcomputers to transfer data between the microcomputer and external devices. The general characteristics of I/O are described. One communicates with a microcomputer system via the I/O devices interfaced to it. The user can enter programs and data using the keyboard on a terminal and execute the programs to obtain results. Therefore, the I/O devices connected to a microcomputer system provide an efficient means of communication between the computer and the outside world. These I/O devices are commonly called peripherals and include keyboards, CRT displays, printers, and disks.

The characteristics of the I/O devices are normally different from those of the microcomputer. For example, the speed of operation of the peripherals is usually slower compared to the microcomputer, and the word length of the microcomputer may be different from the data format of the peripheral device. To make the characteristics of the I/O devices compatible with those of the microcomputer, interface hardware circuitry between the microcomputer and I/O devices is necessary.

In a typical microcomputer system, the user gets involved with two types of I/O devices: physical I/O and logical I/O. When the microcomputer has no operating system, the user must work directly with physical I/O devices and perform detailed I/O design.

There are three ways of transferring data between the microcomputer and a physical I/O device:

- Programmed I/O
- Interrupt driven I/O
- Direct memory access (DMA)

The microcomputer executes a program to communicate with an external device via a register called the I/O port for programmed I/O.

An external device requests the microcomputer to transfer data by activating a signal on the microcomputer's interrupt line during interrupt I/O. In response, the microcomputer executes a program called the interrupt-service routine to carry out the function desired by the external device, again by way of one or more I/O ports.

Data transfer between the microcomputer's memory and an external device occurs without microprocessor involvement with direct memory access.

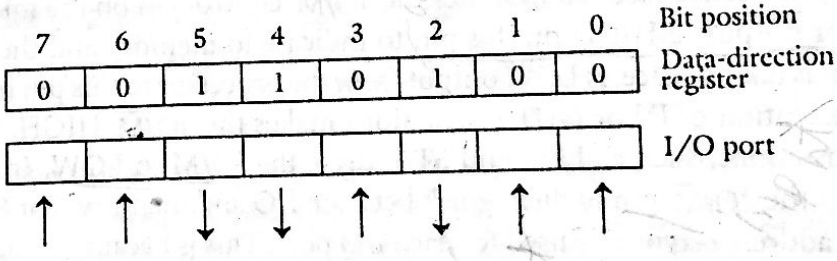
For a microcomputer with an operating system, the user works with virtual I/O devices. The user does not have to be familiar with the characteristics of the physical I/O devices. Instead, the user performs data transfers between the microcomputer and the physical I/O devices indirectly by calling the I/O routines provided by the operating system using virtual I/O instructions. This is called logical I/O.

1.3.4.a Programmed I/O

As described earlier, the microcomputer communicates with an external device via one or more registers called I/O ports using programmed I/O. These I/O ports are occasionally fabricated by the manufacturer in the same chip as the memory chip to achieve minimum chip count for small system applications. For example, the Intel 8355/8755 contains 2K bytes of ROM/EPROM with two I/O ports. The Motorola 6846 has 2K bytes of ROM and an 8-bit I/O port.

I/O ports are usually of two types. For one type, each bit in the port can be individually configured as either input or output. For the other type, all bits in the port can be set up as either all parallel input or output bits. Each port can be configured as an input or output port by another register called the command, or data-direction register. The port data register contains the actual input or output data. The data-direction register is an output register and can be used to configure the bits in the port as inputs or outputs.

Each bit in the port can usually be set up as an input or output by respectively writing a 0 or a 1 in the corresponding bit of the data-direction register (DDR). A bidirectional buffer (one input buffer and one output buffer) is connected at each bit of the port. A '1' written to a particular bit in DDR enables the output buffer while a '0' enables the input buffer connected at the corresponding bit of the port. As an example, if an 8-bit data-direction register contains 34_{16}, then the corresponding port is defined as follows:

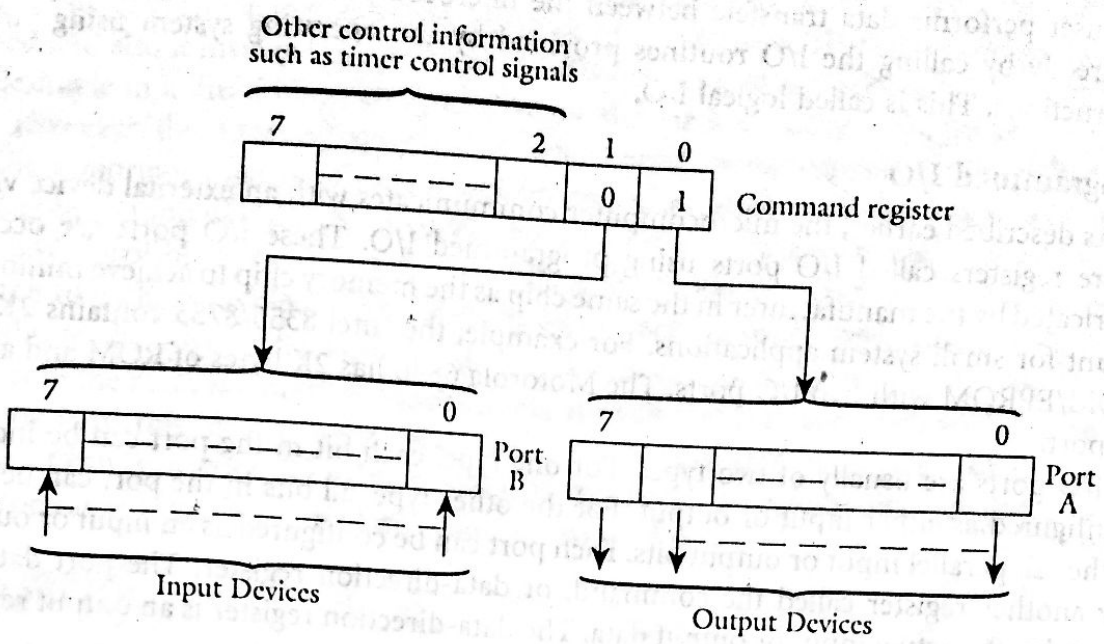


In the preceding example, since 34_{16} ($0011\ 0100_2$) is sent as an output into the data-direction register, bits 0, 1, 3, 6, and 7 of the port are set up as inputs, and bits 2, 4, and 5 of the port are defined as outputs. The microcomputer can then send outputs to external devices, such as LEDs, connected to bits 2, 4, and 5 through a proper interface. Similarly, the microcomputer can input the status of external devices, such as switches, through bits 0, 1, 3, 6, and 7. To input data from the input switches, the 8-bit microcomputer assumed here inputs the complete byte, including the bits to which LEDs are connected. While receiving input data from an I/O port, however, the microcomputer places a value, probably 0, at the bits configured as outputs and the program must interpret them as "don't cares". At the same time, the microcomputer's outputs to bits configured as inputs are disregarded.

For parallel I/O, there is only one register, known as the command register, for all ports. A particular bit in the command register configures all bits in a port as either inputs or outputs.

Consider two I/O ports in an I/O chip along with one command register. Assume that a 0 or a 1 in a particular bit position defines all bits of ports A or B as inputs or outputs.

For example,



Some I/O ports are called handshake ports. Data transfer occurs via these ports through exchanging of control signals between the I/O controller and an external device.

1.3.4.b Standard I/O Versus Memory-Mapped I/O

I/O ports are addressed using either standard I/O or memory-mapped I/O techniques. The standard I/O, also called isolated I/O, uses the IO/$\overline{M}$ control pin on the microprocessor chip. The processor outputs a HIGH on this pin to indicate to memory and the I/O chips that an I/O operation is taking place. A LOW output from the processor to this pin indicates a memory operation. Execution of IN or OUT instructions makes the IO/$\overline{M}$ HIGH, whereas memory-oriented instructions, such as LDA and STA, drive the IO/M to LOW. In standard I/O, the processor uses the IO/$\overline{M}$ pin to distinguish between I/O and memory. For 8-bit microprocessors, an 8-bit address is typically used for each I/O port. This is because 8 bits are the basic data unit for these processors. Eight-bit processors are usually capable of directly addressing 64K bytes of memory using 16 address lines. With an 8-bit I/O port address, these processors are capable of addressing 256 ports. However, in a typical application, there are usually four or five I/O ports required. Some of the address bits of the microprocessor are normally decoded to obtain the I/O port addresses. With memory-mapped I/O, the processor does not differentiate between I/O and memory and, therefore, does not use the microprocessor's IO/$\overline{M}$ control pin. The microprocessor uses the memory addresses (which may not exist in the microcomputer's physical memory) to represent I/O ports. The I/O ports are mapped into the microprocessor's main memory and, hence, are called *memory-mapped I/O.*

In memory-mapped I/O, the most significant bit (MSB) of the address may be used to distinguish between I/O and memory. If the MSB of address is 1, an I/O port is selected. If the MSB of address is 0, a memory location is accessed. This reduces the microprocessor's addressing memory (main memory) by 50%. Sixteen and thirty-two bit microprocessors provide special control signals for performing memory-mapped I/O. Thus, these processors do not use MSB of the address lines. Intel microprocessors can use either standard or memory-mapped I/O while Motorola microprocessors use only memory-mapped I/O. For example, with standard I/O, Intel 8086 uses IN AL, PortA, and OUT PortA, AL for inputing and outputing data. On the other hand, the 8086 uses memory-oriented instructions such as MOV AL, START, and MOV START, AL for inputing and outputing data. Note that START is an

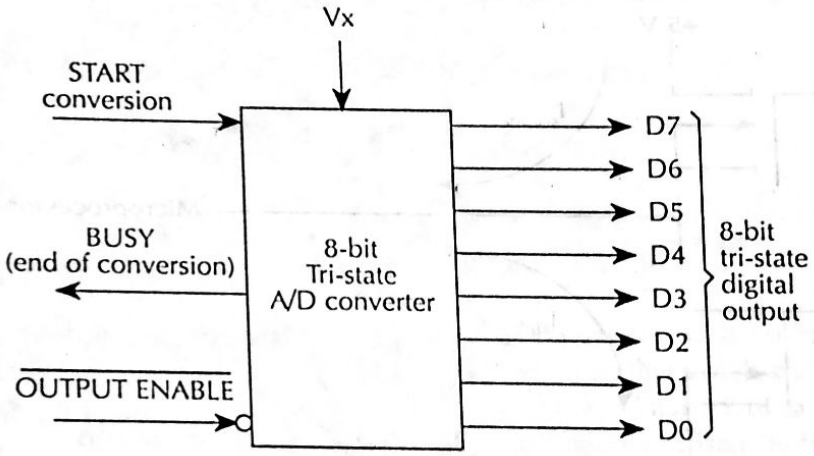
I/O port mapped as a memory address. Motorola, on the other hand, does not have any IN or OUT instructions and uses memory-oriented instructions for I/O operation.

1.3.4.c Unconditional and Conditional Programmed I/O

The microprocessor can send data to an external device at any time during unconditional I/O. The external device must always be ready for data transfer. A typical example is when the processor outputs a 7-bit code through an I/O port to drive a seven-segment display connected to this port.

In conditional I/O, the microprocessor outputs data to an external device via handshaking. Data transfer occurs by the exchanging of control signals between the microprocessor and an external device. The microprocessor inputs the status of the external device to determine whether the device is ready for data transfer. Data transfer takes place when the device is ready.

The concept of conditional I/O will now be demonstrated by means of data transfer between a microprocessor and an analog-to-digital (A/D) converter. Consider, for example, the A/D converter shown in the accompanying figure.



The A/D converter just shown transforms an analog voltage Vx into an 8-bit binary output at pins D7—D0. A pulse at the START conversion pin initiates the conversion. This drives the BUSY signal LOW. The signal stays LOW during the conversion process. The BUSY signal goes HIGH as soon as the conversion ends. Since the A/D converter's output is tristated, a LOW on the $(\overline{\text{OUTPUT ENABLE}})$ transfers the converter's output. A HIGH on the $(\overline{\text{OUTPUT ENABLE}})$ drives the converter's output to a high impedance state.

The concept of conditional I/O can be demonstrated by interfacing the A/D converter to an 8-bit processor. Figure 1.14 shows such an interfacing example.

The user writes a program to carry out the conversion process. When this program is executed, the processor sends a pulse to the START pin of the converter via bit 2 of port A. The microprocessor then checks the BUSY signal by bit 1 of port A to determine if the conversion is completed. If the BUSY signal is HIGH (indicating the end of conversion), the microprocessor sends a LOW to the $(\overline{\text{OUTPUT ENABLE}})$ pin of the A/D converter. The microprocessor then inputs the converter's D0—D7 outputs via port B. If the conversion is not completed, the microprocessor waits in a loop checking for the BUSY signal to go HIGH.

1.3.4.d Typical Microcomputer Output Circuit

The microcomputer designer is often concerned with the output circuit because of the microcomputer's small output current drive capability. The tristate output circuit shown in Figure 1.15 typically is utilized by a microcomputer as the output circuit. This circuit uses totem pole-type output called a PUSH-PULL circuit providing low-output currents. Therefore, a current amplifier (buffer) is required to drive devices such as LEDs.

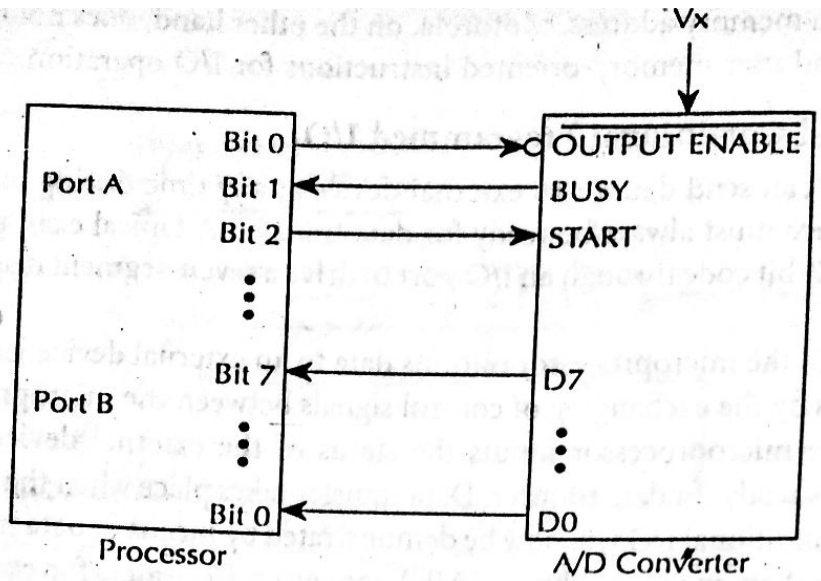


FIGURE 1.14 Interfacing an A/D converter to an 8-bit processor.

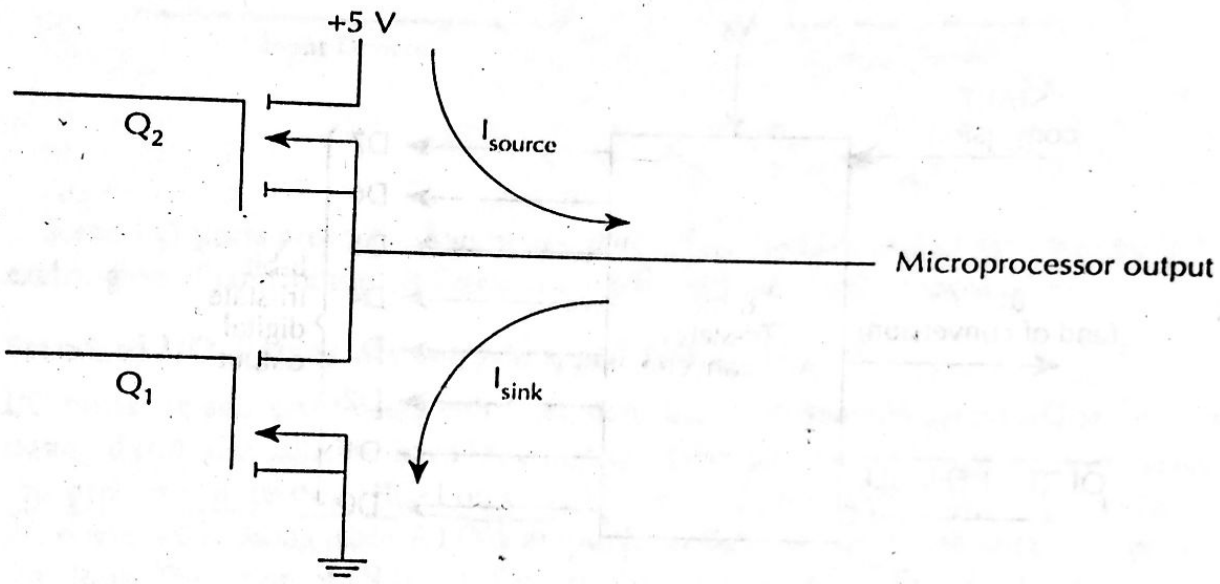


FIGURE 1.15 Digital microprocessor output circuit.

In the preceding figure, when Q_1 is ON, Q_2 is OFF and I_{sink} will flow from the external device into Q_1. Also, when Q_2 is ON, Q_1 is OFF and I_{source} will flow from Q_2 into the output device. Figure 1.16 shows a hardware interface circuit using the push-pull circuit for driving an external device such as LED.

Assume I_{source} to be 400 μA (usually represented by a negative sign such as $I_{OH} = -400$ μA; the negative sign indicates that the chip is losing current) with a minimum voltage V_A of 2.4 at point A, and that the LED requires 10 mA at 1.7 V. Therefore, a buffer such as a transistor is required at the output circuit to increase the current drive capability to drive the LED. In order to design the interface, the values of R_1,R_2 and minimum β of the transistor will be determined in the following:

$$R_1 = \frac{V_A - V_{BE}(Q_3)}{400\ \mu A} = \frac{2.4 - 0.7}{400\ \mu A} = 4.25\ k\Omega$$

Since $I_{LED} = 10$ μA at 1.7 V and assuming that V_{CE} (saturation) = 0 V,

$$R_2 = \frac{5 - 1.7 - V_{CE}(Q_3)}{10\ mA} = \frac{3.3}{10\ mA} = 330\ \Omega$$

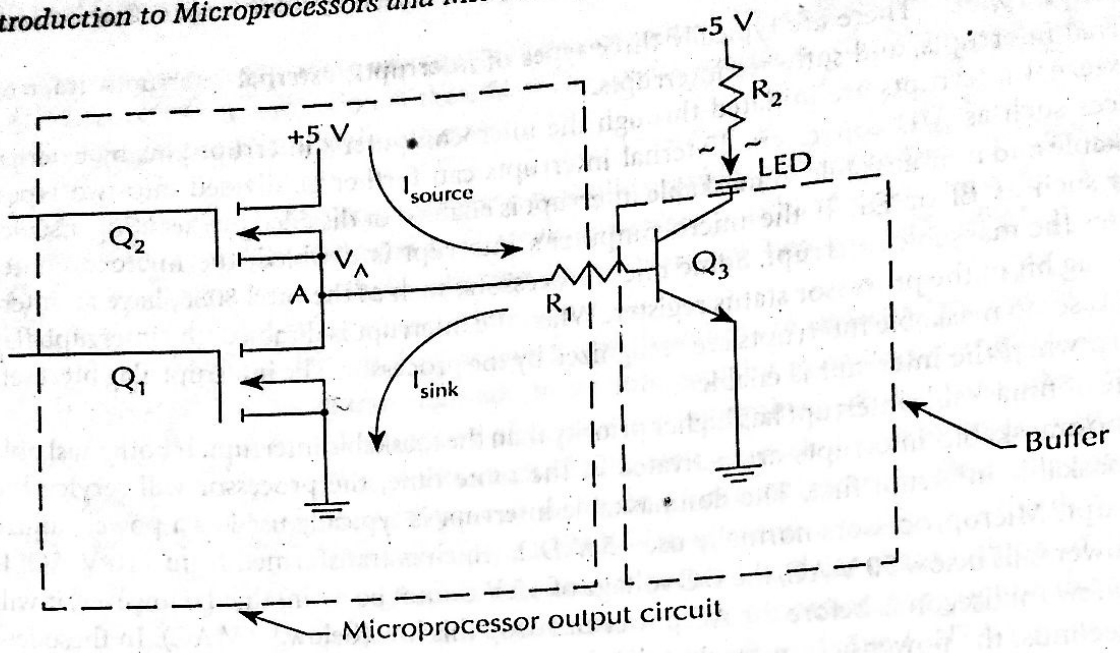


FIGURE 1.16 Microprocessor I/O interfaces for driving an LED.

Since $I_{source} = 400\ \mu A = I_B(Q_3)$, β for transistor Q_3 is

$$\beta = \frac{I_C(Q_3)}{I_B(Q_3)} = \frac{10\ mA}{400\ \mu A} = \frac{10 \times 10^{-3}}{400 \times 10^{-6}} = 25$$

Therefore, the interface design is complete, and a transistor with a mimimum of saturation β of 25 and $R_1 = 4.25\ k\Omega$ and $R_2 = 330\ \Omega$ is required. Note that a MOS outputs more sink current than source current. If sink current is used, the LED in Figure 1.16 can directly be connected to the microcomputer's output through an appropriate resistance. However, if the resistor value is not large enough, it may damage the transistor Q1.

1.3.4.e Interrupt Driven I/O

A disadvantage of conditional programmed I/O is that the microcomputer needs to check the status bit (BUSY signal for the A/D converter) by waiting in a loop. This type of I/O transfer is dependent on the speed of the external device. For a slow device, this waiting may slow down the capability of the microprocessor to process other data. The polled I/O and interrupt I/O techniques are efficient in this type of situation.

Interrupt I/O is a device-initiated I/O transfer. The external device is connected to a pin called the *interrupt* (INT) pin on the microprocessor chip. When the device needs an I/O transfer with the microcomputer, it activates the interrupt pin of the microprocessor chip. The microcomputer usually completes the current instruction and saves at least the contents of the current program counter on the stack.

The microcomputer then automatically loads an address into the program counter to branch to a subroutine-like program called the interrupt-service routine. This program is written by the user. The external device wants the microcomputer to execute this program to transfer data. The last instruction of the service routine is a RETURN, which is typically the same instruction used at the end of a subroutine. This instruction normally loads the return address (saved in the stack before going to the service routine) in the program counter. Then, the microcomputer continues executing the main program. Note that subroutines and interrupts are handled in a similar way: subroutines are initiated via execution of an instruction such as subroutine CALL while interrupts are initiated via activation of the microprocessor's interrupt pin by the interrupting device.

1.3.4.e.i Interrupt Types. There are typically three types of interrupts: external interrupts, traps or internal interrupts, and software interrupts.

External interrupts are initiated through the microcomputer's interrupt pins by external devices such as A/D converters. External interrupts can further be divided into two types: maskable and nonmaskable. A maskable interrupt is enabled or disabled by executing instructions such as EI or DI. If the microcomputer's interrupt is disabled, the microcomputer ignores the maskable interrupt. Some microprocessors, such as the Intel 8086, have an interrupt-flag bit in the processor status register. When the interrupt is disabled, the interrupt-flag bit is 1, so no maskable interrupts are recognized by the processor. The interrupt-flag bit resets to zero when the interrupt is enabled.

The nonmaskable interrupt has higher priority than the maskable interrupt. If both maskable and nonmaskable interrupts are activated at the same time, the processor will service the nonmaskable interrupt first. The nonmaskable interrupt is typically used as a power failure interrupt. Microprocessors normally use +5 V DC, which is transformed from 110 V AC. If the power falls below 90 V AC, the DC voltage of +5 V cannot be maintained. However, it will take a few milliseconds before the AC power can drop this low (below 90 V AC). In these few milliseconds, the power failure-sensing circuitry can interrupt the microprocessor. An interrupt service routine can be written to store critical data in nonvolatile memory such as battery-backed CMOS RAM. The interrupted program can continue without any loss of data when the power returns.

Some microprocessors are provided with a maskable handshake interrupt. This interrupt is usually implemented by using two pins — INTR and $\overline{\text{INTA}}$. When the INTR pin is activated by an external device, the processor completes the current instruction, saves at least the current program counter onto stack, and generates an interrupt acknowledge ($\overline{\text{INTA}}$). In response to the $\overline{\text{INTA}}$, the external device provides an instruction, such as CALL, using external hardware on the data bus of the microcomputer. This instruction is then read and executed by the microcomputer to branch to the desired service routine.

Internal interrupts, or traps, are activated internally by exceptional conditions such as overflow, division by zero, or execution of an illegal op-code. Traps are handled the same way as external interrupts. The user writes a service routine to take corrective measures and provide an indication to inform the user that an exceptional condition has occurred.

Many microprocessors include software interrupts, or system calls. When one of these instructions is executed, the microprocessor is interrupted and serviced similarly to external or internal interrupts. Software interrupt instructions are normally used to call the operating system. These instructions are shorter than subroutine calls, and no calling program is needed to know the operating system's address in memory. Software interrupt instructions allow the user to switch from user to supervisor mode. For some microprocessors, a software interrupt is the only way to call the operating system, since a subroutine call to an address in the operating system is not allowed.

1.3.4.e.ii Interrupt Address Vector. The technique used to find the starting address of the service routine (commonly known as the *interrupt address vector*) varies from one microprocessor to another. With some microprocessors, the manufacturers define the fixed starting address for each interrupt. Other manufacturers use an indirect approach by defining fixed locations where the interrupt address vector is stored.

1.3.4.e.iii Saving the Microprocessor Registers. When a microprocessor is interrupted, it saves at least the program counter on the stack so the microprocessor can return to the main program after executing the service routine. Some microprocessors save only one or two registers, such as the program counter and status register. Some other microprocessors save all microprocessor registers before going to the service routine. The user should know the specific registers the

microprocessor saves prior to executing the service routine. This will enable the user to use the appropriate return instruction at the end of the service routine to restore the original conditions upon return to the main program.

1.3.4.e.ivInterrupt Priorities. A microprocessor is typically provided with one or more interrupt pins on the chip. Therefore, a special mechanism is necessary to handle interrupts from several devices that share one of these interrupt lines. There are two ways of servicing multiple interrupts: polled and daisy chain techniques.

Polled interrupts are handled by software and therefore are slower when compared with daisy chaining. The processor responds to an interrupt by executing one general service routine for all devices. The priorities of devices are determined by the order in which the routine polls each device. The processor checks the status of each device in the general service routine, starting with the highest-priority device to service an interrupt. Once the processor determines the source of the interrupt, it branches to the service routine for the device.

In a daisy chain priority system, devices are connected in a daisy chain fashion to set up a priority system. Suppose one or more devices interrupt the processor. In response, the processor pushes at least the PC and generates an interrupt acknowledge (INTA) signal to the highest-priority device. If this device has generated the interrupt, it will accept the INTA. Otherwise it will pass the INTA onto the next device until INTA is accepted. Once accepted, the device provides a means for the processor to find an interrupt address vector by using external hardware. The daisy chain priority scheme is based on mostly hardware and is therefore faster than the polled interrupt.

1.3.4.f Direct Memory Access (DMA)

Direct Memory Access (DMA) is a technique that transfers data between a microcomputer's memory and an I/O device without involving the microprocessor. DMA is widely used in transferring large blocks of data between a peripheral device and the microcomputer's memory. The DMA technique uses a DMA controller chip for the data-transfer operation. The main functions of a typical DMA controller are summarized as follows:

- The I/O devices request DMA operation via the DMA request lines of the controller chip.
- The controller chip activates the microprocessor HOLD pin, requesting the CPU to release the bus.
- The processor sends HLDA (hold acknowledge) back to the DMA controller, indicating that the bus is disabled. The DMA controller places the current value of its internal registers, such as the address register and counter, on the system bus and sends a DMA acknowledge to the peripheral device. The DMA controller completes the DMA transfer and releases the buses.

There are three basic types of DMA: block transfer, cycle stealing, and interleaved DMA.

For block-transfer DMA, the DMA controller chip takes the bus from the microcomputer to transfer data between the memory and I/O device. The microprocessor has no access to the bus until the transfer is completed. During this time, the microprocessor can perform internal operations that do not need the bus. This method is popular with microprocessors. Using this technique, blocks of data can be transferred.

Data transfer between the microcomputer memory and an I/O device occurs on a word-by-word basis with cycle stealing. Typically, the microprocessor clock is enabled by ANDing an INHIBIT signal with the system clock. The system clock has the same frequency as the microprocessor clock.

The DMA controller controls the INHIBIT line. During normal operation, the INHIBIT line is HIGH, providing the microprocessor clock. When DMA operation is desired, the controller

makes the $\overline{\text{INHIBIT}}$ line LOW for one clock cycle. The microprocessor is then stopped completely for one cycle. Data transfer between the memory and I/O takes place during this cycle. This method is called cycle stealing because the DMA controller takes away or steals a cycle without microprocessor recognition. Data transfer takes place over a period of time.

With interleaved DMA, the DMA controller chip takes over the system bus when the microprocessor is not using it. For example, the microprocessor does not use the bus while incrementing the program counter or performing an ALU operation. The DMA controller chip identifies these cycles and allows transfer of data between the memory and I/O device. Data transfer takes place over a period of time for this method.

The DMA controller chip usually has at least three registers normally selected by the controller's register select (RS) line: an address register, a terminal count register, and a status register. Both the address and terminal count registers are initialized by the microprocessor. The address register contains the starting of the data to be transferred, and the terminal count register contains the desired block to be transferred. The status register contains information such as completion of DMA transfer.

It should be mentioned that while using either block transfer or cycle stealing DMA in systems with dynamic RAMs, circuitry must be included for refreshing the dynamic RAM's during DMA transfer.

1.3.4.g Summary of Microcomputer I/O Methods

Figure 1.17 summarizes the I/O structure (explained so far) of typical microcomputers.

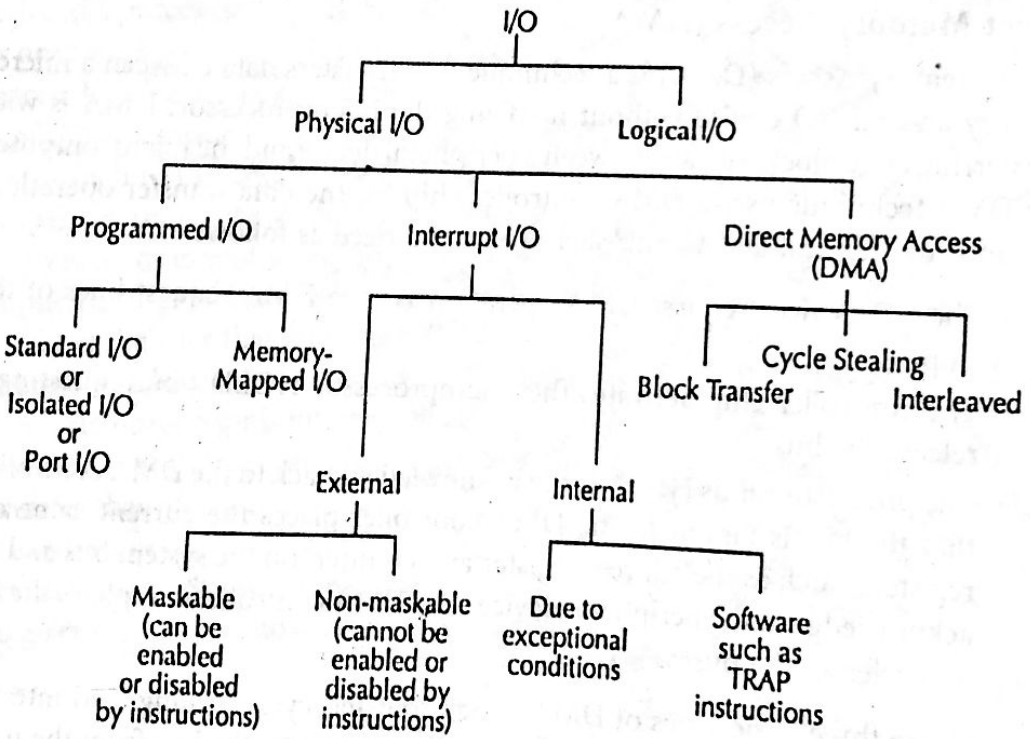


FIGURE 1.17 I/O structure of a typical microcomputer.

1.3.4.h Coprocessors

In typical 8-bit microprocessors such as the Intel 8085, technology places a limit on the chip area. As a consequence, these microprocessors include no hardware or firmware for performing scientific computations such as floating-point arithmetic, matrix manipulation, and graphic-data processing. Therefore, users of these systems must write these programs. Unfortunately