BLOCK II: MEMORY AND INPUT OUTPUT ORGANIZATIONS

UNIT 1: MEMORY ORGANIZATION

Space for learners:

Unit Structure

1.1 INTRODUCTION

A computer consists of three primary building blocks as input /output unit, control unit, and memory unit. It is used as a storage device in a system to store programs or a set of instructions, data, and the intermediate results of arithmetical and logical computations. Depending on storing strategy the memories are classified into two prime categories – main memory or primary memory and auxiliary or secondary memory. Memory can be classified into different categories based on some key characteristics such as:

a. Depending on location, memories are classified as CPU-based, internal memory, and external memory.
b. Depending on media used for manufacturing memory i.e. physical type memory is classified as semi-conductor based and magnetic surface-based.
c. Depending on physical characteristics volatile / non-volatile and erasable / non-erasable.
d. Depending on the access method memories are classified as sequential access, direct access, and random access memory.

Space for learners:

1.2 UNIT OBJECTIVES

After completing this unit, you will be able to learn:

- Functions of the memory unit.
- Memory operations
- Representation of memory location in terms of byte
- Big-endian and little-endian assignment
- Composition of a memory chip.
- Learn about the memory hierarchy.
- Learn about the key factors that affect memory performance.
- Know about the different types of RAM and ROM
- Mapping of a memory chip and required amount of memory.

- Learn about the memory access methods.
- Learn about the concepts of cache and virtual memory.
- Learn about the hardware used in memory management.
- Concepts of secondary memory
- Functions of MMU.

Space for learners:

1.3 MEMORY OPERATIONS

Computer memory is used to store both program instructions and data operands. To execute an instruction the processor should load or transfer the instruction or set of instructions into the processor from the primary memory. During the processing of instructions, the operands or the results are also transferred between the memory and the CPU. Thus the basic operations involving in the memory can be classified into two categories such as Load or Read / Fetch and store or Write.

The READ operation transfers a copy of the contents from the memory location specified by the CPU. During the transfer, the memory contents remain unchanged. The READ operation is initiated by the CPU sending a request to the memory with a specific memory location in the address bus. The memory unit will read the contents from the specified address by the CPU and send them to the processor by loading the data into the data bus.

The WRITE operation transfers data from the processor to a specific memory location specified by the instruction. This operation will overwrite the contents in that memory location. The processor sends the data along with the memory location where it has to be stored or written. In a single operation, one word or 1 byte of data can transfer between the memory and the processor.

STOP TO CONSIDER

- The two main memory operations are READ and WRITE
- READ operation perform to fetch data from memory to processor
- WRITE operation perform to store data from processor to memory

Space for learners:

1.4MEMORY CHIP

An integrated circuit (IC) consists of several capacitors and transistors with the capacity of storing information can be defined as a memory chip. Memory chips can be used for process code also. Memory chips can hold data either temporarily or permanently through RAM and ROM respectively. The size or shape and storage capacity of the memory chip can vary.

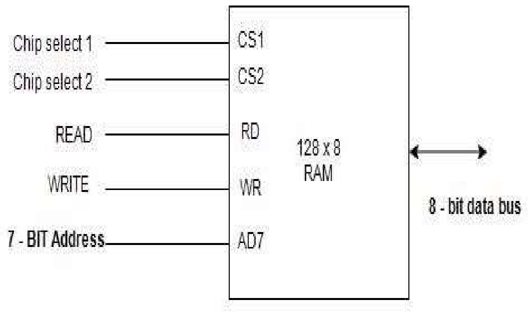


Figure: 1.1 Block diagram of a RAM Chip

A RAM chip is used to communicate with the CPU through control lines. Through a bidirectional data bus, RAM chips are allowed to communicate either from memory to CPU during a read operation or from CPU to RAM during a write operation. Following figure Fig.1.1 shows a typical block diagram of a RAM chip. The chip capacity is 128 words of 8 bits per word. This 128 x 8 chip required a 7-bit address and an 8-bit bidirectional data bus. The signal RD and WR are used to specify memory operations Read/Write respectively during communication. The chip select (CS) line is a control line through which the microprocessor can select and enable a particular chip. The functions of a RAM chip can be depicted as shown in Table 1.

Table 1: Functions table for RAM Chip

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High impedance
0	1	x	x	Inhibit	High impedance
1	0	0	0	Inhibit	High impedance
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High impedance

The RAM chip is in operation only when the value of CS1 = 1 and $\overline{CS2}$= 0. The barin $\overline{CS2}$ indicates that the input is enabled for its complements i.e. for the value 0. If the select controls are not enabled or if it is enabled, but Read and Write lines are not enabled, then the memory is inhibited and the data bus is in high impedance. The high impedance is a state where it behaves like an open circuit i.e. the output does not carry any signal .It leads to very high resistance and hence no current flows. When the CS1 = 1 and $\overline{CS2}$ = 0 the memory can be in reading or write mode. When the WR input line is enabled, then a byte of information will be transferred from the data bus into the memory location specified by the address lines. When the read input line is enabled, a byte of information from the memory specified by the address line is transferred into the data bus.

The ROM chip is also organized the same as that of the RAM chip. In the ROM chip there is no need for reading and writing input control because the unit can only read. Thus if the chip is selected the bytes as specified by the address line will be appeared in the data bus.

Space for learners:

STOP TO CONSIDER

- Memory chips hold data temporarily or permanently through RAM and ROM.
- A RAM chip is used to communicate with CPU through control lines
- Depending on the requirement of memory the number of memory chip may vary.

Space for learners:

Generally, the size of RAM and ROM varies from machine to machine. If a system required more memory storage than the capacity of a chip then many chips are required to get the necessary memory size. If the required size of memory is M x N and the chip capacity is m x n then the required number of chips can be calculated as

$$\mathbf{k} = \frac{\mathbf{M*N}}{\mathbf{m*n}}$$

1.5 MEMORY LOCATIONS AND ADDRESSES

Program instructions, operands, and results of arithmetical logical operations are stored in computer memory. The computer memory is composed of millions of storage cells. Each storage cell can store one bit of information in the form of 0 or 1. To perform basic memory operations the cells are grouped into a fixed number of cells. Each group with n-bit is referred to as a "**word**" of information and the "**n**" will be known as "**word length**". It can be depicted in figure 1.2. Now a day's modern computers are typically ranging between 16 – 64 bits of word length.

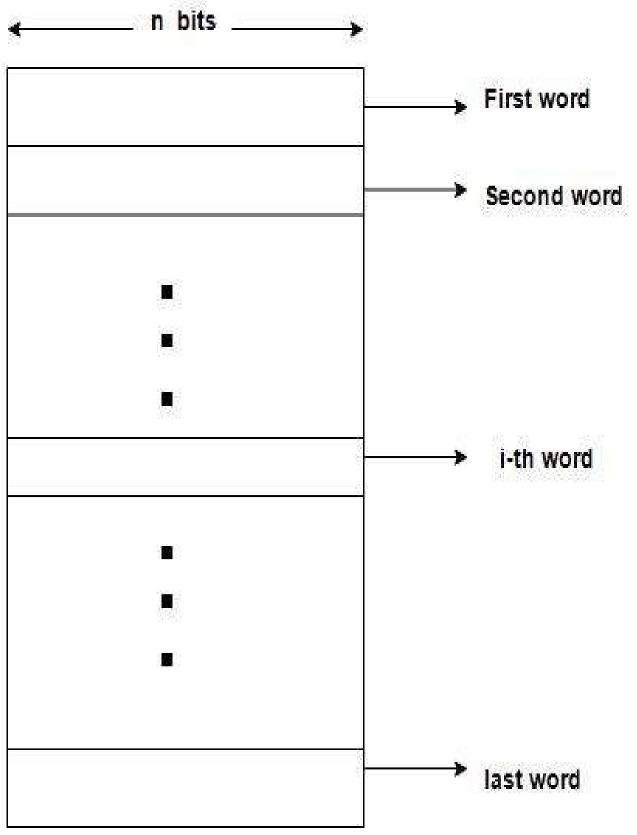


Figure 1.2 Memory words

A computer with 64 bit means that the address bus can carry an address of 64 bit for a specified memory location in computer memory. The address of memory locations is represented by 64-bit numbers. A computer with 32bit can represent 2^{32} = 4294967296

Thus the addressing scheme of a system determined the maximum size of computer memory or address space. For example, a system with a 16-bit computer i.e. with an addressing scheme of 16-bit addresses can address up to 2^{16} = 64 K number of memory locations. Similarly, a machine with 32-bit addresses can generate 2^{32} = 4GB memory locations. The memory location of a system determines the address space. Thus the addresses of each memory location are represented with k bits and using k address bit, 2^k nos.

Space for learners:

of locations or addresses can be represented. The address bit and number of locations is depicted in Table 2. Most computer systems are byte-addressable and memory is usually designed to store or access data in word-length quantities. For a computer, the word length can be defined as the number of bits that store or are retrieved in one access. The processor reads the memory data by loading the address of the required memory location into the Memory Address Register (MAR). Similarly, during a write operation, the processor writes data into a memory location by loading the address of that location into MAR. To perform the read/write operation on a set of consecutive memory locations in the main memory, then a block transfer operation may perform by sending the first address of the memory locations.

Space for learners:

Table 2: Address bit and number of locations

K	Number of Locations
10	2^{10} = 1024 = 1 K
16	2^{16} = 65,536 = 64 K
20	2^{20} = 1,048,576 = 1 M
24	2^{24} = 16,777,216 = 16 M

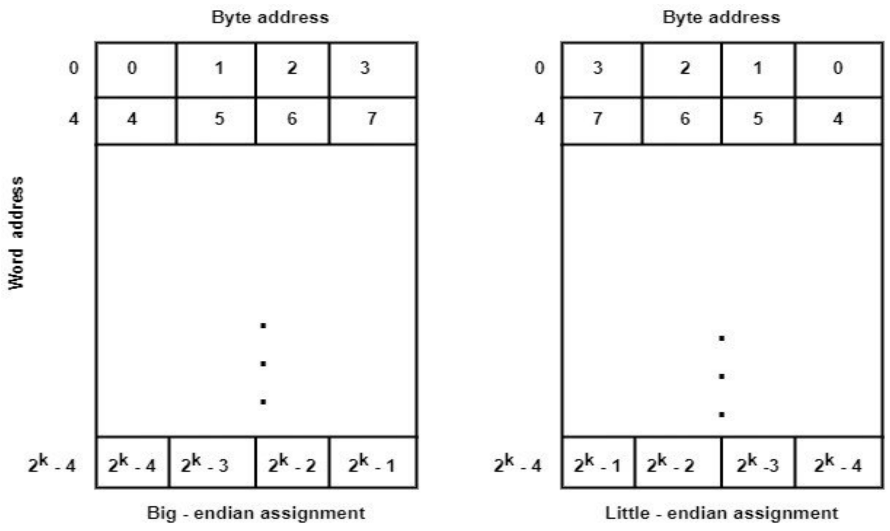
1.5.1 Byte addressability

A nibble is always 4 bits and a byte is 8 bits. The word length of a computer system can range between 16 – 64 bits. To assign an individual address for each of the bit locations in memory will increase the complexity of memory organization. In modern practices, each successive address refers to successive byte locations in memory. For a computer system with 32 bits, successive words will be located at addresses 0000, 0004, 0008,……… with each word of four bytes.

1.5.2 Big – Endian and Little – Endian assignments

Space for learners:

To assign the addresses across the words, there are two ways known as big-endian and little-endian assignments. The big-endian is used when the lower order byte addresses are used for the more significant bytes (MSB) or the leftmost bytes as shown in Figure 1.3. The little-endian is used when the lower order byte addresses are used for the less significant bytes (LSB) or rightmost bytes of the word as shown in Figure 1.3. Commercial machines are used both ways of assignment. To specify the address ordering of bytes within a word it is mandatory to specify the labelling of bits within a byte or a word as shown in Figure 1.4.



Word address	Byte address			
0	0	1	2	3
4	4	5	6	7
⋮		⋮		
$2^k - 4$	$2^k - 4$	$2^k - 3$	$2^k - 2$	$2^k - 1$

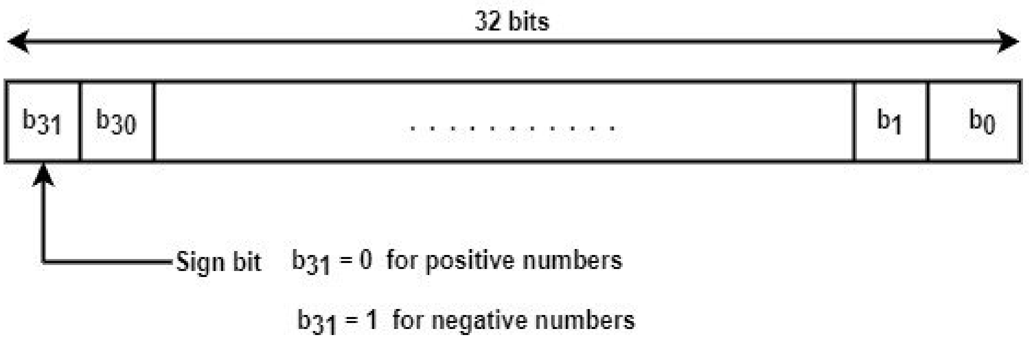
Word address	Byte address			
0	3	2	1	0
4	7	6	5	4
⋮		⋮		
$2^k - 4$	$2^k - 1$	$2^k - 2$	$2^k - 3$	$2^k - 4$

Figure 1.3 Big-endian and little-endian assignment

STOP TO CONSIDER

- The processor reads the memory data by loading the address of the required memory location into the Memory Address Register (MAR).
- A computer with 64 bit means that the address bus can carry an address of 64 bit for a specified memory location in computer memory.
- To assign the addresses across the words, there are two ways known as Big – endian and little – endian assignments.
- In modern practices each successive addresses refers to successive byte locations in memory.

Space for learners:



32 bits
b_{31}
b_{30}
. . . .
b_1
b_0
Sign bit $b_{31} = 0$ for positive numbers
$b_{31} = 1$ for negative numbers

Figure 1.4 labelling of bits within a byte or a word

1.6 MEMORY HIERARCHY

Memory performance mainly depends on some key parameters –

a. **Memory access time** –It is defined as the total time requirement from submission of a request for the required piece of information by the CPU for getting or availability of the information in the CPU. CPU registers are local memory for the CPU and the access time is few nanoseconds. Cache memory takes small multiple access times of CPU registers. Cache memory is portions of memory made up of very high-speed static RAM (SRAM). Primary memory access time is few tens of nanoseconds. For secondary memory, the access time is at least 10 msec. and it may measure in seconds if the data is to be fetched/write from or to a drive.

b. **Storage capacity:** The storage capacity of memory has a greater role in performance. As the capacity increase, the access time of the memory is also increased. CPU registers are good for almost 128 bytes. Cache memory can be range as for L1 cache – 8 KB to 64 KB, for L2 cache – 256 KB to 512 KB, and for L3 cache 8 MB to 32 MB. Primary memory storage capacity ranges from 512 MB to 32 GB. The storage capacity of Secondary memory can vary from few gigabytes to terabytes or more than that.

The memory hierarchy shows the organization of different types of memories depending on their performance. It can be explained with a block diagram as shown in the following figure Fig.1.5. At the top of the memory hierarchy, CPU registers are located which are compact and accessible at full CPU speed. The next high-speed and high-cost memory is cache memory. CPU collects the required piece of information from cache memory. From the peak to the bottom of the hierarchy, memory access time and size of the memory are gradually increase and the costs of memory

Space for learners:

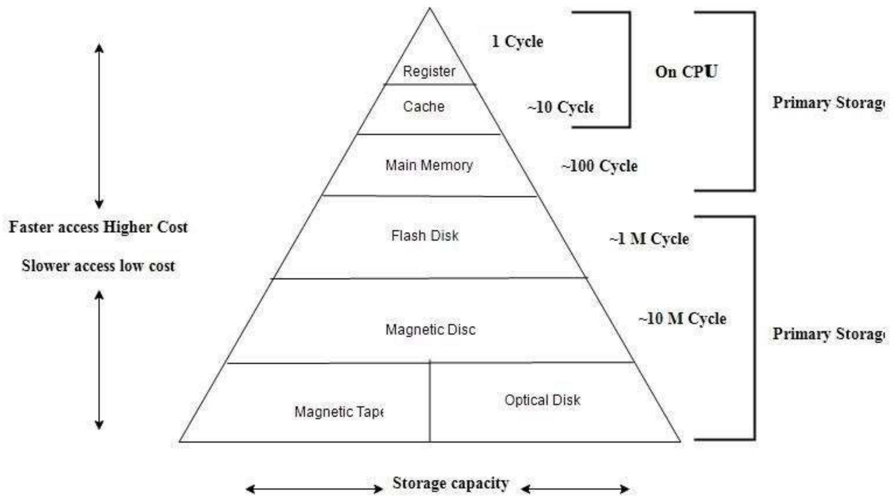


Figure 1.5 The memory hierarchy

decrease. The memory hierarchy primarily depends on some key parameters such as access time and storage capacity of the memory.

Space for learners:

STOP TO CONSIDER

- Depending on the key factors such as storage capacity, accessibility, average access time memory can be organized in a pyramid structure known as memory hierarchy.
- Fastest and smallest memory lies on top or peak of the pyramid structure.
- Slower and bigger storage capacity memories lie in bottom side of the pyramid.

1.7 SECONDARY MEMORY

Memory devices where data are kept permanently for a long time can call secondary memory. Secondary memories are non-volatile i.e. can store data permanently during a power cut or off mode. Some of the secondary memory devices are computer hard drive disk, pen drive, floppy disk, CD, etc. In comparison to the main memory size of the secondary or auxiliary memory is very large. The memory access rate of auxiliary memory is comparatively very less than main memory. Hence the cost is also relatively inexpensive. Thus we can say that cost is directly proportional to the storage capacity of the memory.

1.8 MAIN MEMORY

The CPU directly communicated with a memory unit known as the main memory. The storage capacity of this type of memory is very large in comparison to cache memory and very small in comparison to secondary memory. The main memory can be classified into two different categories such as RAM and ROM.

1.8.1 Random Access Memory (RAM)

Space for learners:

RAM can be defined as a read/write memory. Users can read the memory contents from RAM and also can write into RAM. It is volatile, i.e. it loses all the data when the power goes down. As the power supply goes down the memory contents or stored information in RAM are lost. In RAM any memory location can be accessed randomly without going through any other memory location. The access time for each location is the same. RAM can be classified into two categories as static or SRAM and Dynamic memory or DRAM.

1.8.1.1 Static RAM (SRAM):

SRAM consists of CMOS technology and uses transistors. For storing binary data it is used two cross-coupled inverters which is similar to flip-flops along with two other transistors for access control. The binary information exits in SRAM as long as the power supply is on. Figure1.6 illustrates how an SRAM cell is implemented. Two inverters are cross-coupled to form a latch. Two transistors T1 and T2 are used to connect the latch with the two-bit lines. Using the word line the transistors can be open or closed. When the word line is at ground level, the transistors are turned off and the latch retains its state.

To read the state of the SRAM cell, the word line is activated to close the switches T1 and T2.The signal on bit line b will be high and $b^{/}$will be low for cell state 1. Similarly, for cell state 0, the signal on bit line b is low and the signal in $b^{/}$ is high. Thus b and $b^{/}$ are complements of each other. The state of the cell is set by activating the word line and putting the appropriate value for the bit line b and its complement $b^{/}$. Required signals on the bit line are generated by the sense/write circuit.

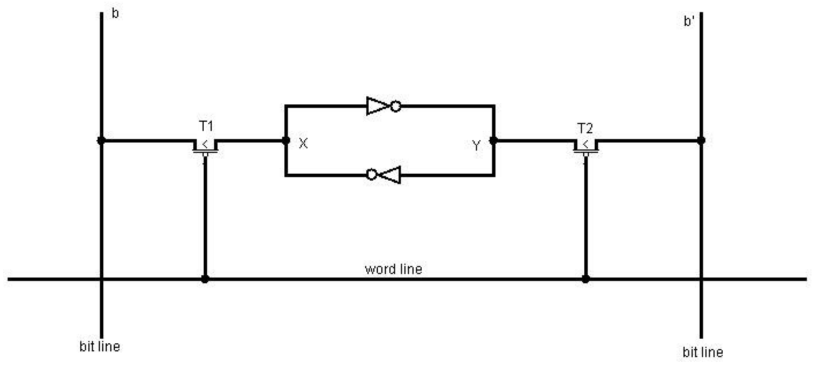


Figure 1.6 a static RAM cell

1.8.1.2 Dynamic RAM (DRAM):

DRAM is constructed using capacitors and few transistors. The term dynamic in DRAM indicates that the charges are continuously discharging even in presence of an uninterrupted power supply and hence the capacitors must refresh periodically through refreshing the DRAM. DRAM is available in the market as it is less expensive. SRAM is an on-chip memory with very little access time whereas DRAM is off-chip memory with a large access time in comparison to SRAM. So SRAM is faster than DRAM. The storage capacity of SRAM is less than DRAM. Cache memories are comprised of SRAM whereas the main memory is comprised of DRAM. Power consumption in DRAM is more in comparison to SRAM.

STOP TO CONSIDER

- Memory can be volatile or non-volatile in nature.
- RAM is volatile and ROM is non-volatile memory.
- SRAM and DRAM are the two categories of random access memory (RAM).

Space for learners:

Space for learners:

1.8.2 Read Only Memory (ROM)

ROMis a non-volatile memory, i.e. contents of this type of memory remain the same or permanently in the memory and not erased due to power cut. Contents of ROM can be read or accessed during operation and nothing can be written into it by the user or programmer. The manufacturing company decides and writes permanently into the ROM during manufacture. Different types of ROMs are PROM, EPROM, and EEPROM. Programs or sets of instructions that are required for starting a computer i.e. bootstrap programs are stores in ROM. ROM is used in some electronic gadgets such as fridges, refrigerators, washing machines, microwaves, etc.

1.8.2.1 Programmable read-only memory (PROM)

PROM was first developed in 1956 by Wen Tsing Chow. It is a memory chip that can be programmed once after is created. Once the memory chip is programmed, the information written on it becomes permanent and cannot be erased or deleted. PROM was used in computer BIOS in early day's computers and now it is replaced by EEPROM.

1.8.2.2 Erasable Programmable Read-Only Memory (EPROM)

EPROM is a memory chip that can store data even after a power cut also. Data from EPROM can be erased using ultraviolet light and makes it re-writable or programmable. It was first developed by Dov Frohman in 1971 at Intel. The contents of EPROM can be erased limitedly. Too much deletion can make the memory unit unreliable by destroying the silicon dioxide layer. It is not possible to erase the EPROM contents partially. The whole data from EPROM is erased. For erasing and reprogramming the EPROM, the chip has to remove from the computer system and it consumes lots of time to erase data. The process of programming on EPROM is known as Burning and it is not a reversible process. It was developed to overcome the drawbacks of ROM and PROM. EPROM is successfully used in some microcontrollers such as Intel 8048, bootstrap loader, video-game, personal computers, etc.

1.8.2.3 Electrically Erasable Programmable Read-Only Memory (EEPROM)

Itis a memory chip that can be erased by exposing electrical charge. Like other ROM, EEPROM retains its stored data even power is turned off. In the year 1978, George Perlegos developed the concept of EEPROM at INTEL. To erase the contents of EEPROM consumes approximately 5 milliseconds. In EEPROM erasing and reprogramming can be done without switching off the electrical circuit of the system.

1.9 CACHE MEMORY

The processing speed of the CPU in comparison to the access time of primary memory is very high. Due to this bottlenecking CPU cannot be utilized at its utmost level and remains idle. To remove this barrier a smaller memory is used in the system such that the average access time got increases and makes the computer memory more efficient. This chip-based smaller and faster memory is known as cache memory. It is a temporary storage area from which the

CPU can retrieve data easily during processing. Sometimes it is called the CPU Memory as it is typically integrated directly with the CPU chip or placed on a separate chip that has a direct connection with the CPU through a separate bus. As the cache memory is smaller in size and faster access time in comparison to primary memory, it increases the average access time and efficiency of the processor. The access time of cache memory is 10 to 100 times faster than the primary memory of a system. Cache consumes only a few nanoseconds to respond to a CPU request. Cache memory built with high-speed SRAM. It can be categorized into three different levels such as L1, L2, and L3 cache. L1 cache is extremely fast and usually embedded in the processor chip. L2 or secondary cache can be implanted on the CPU with a system bus. L3 cache is used to increase the performance of L1 and L2. The speed of L3 is comparatively slower than L1 and L2 but two times faster than DRAM.

Space for learners:

1.10VIRTUAL MEMORY

A computer has a limited amount of memory space in primary memory or RAM. During programming, a user or developer has to concern about the limited amount of free memory address in RAM which increases the complexity of programming. To overcome this difficulty a technique called virtual memory has arisen. Virtual memory allows using more addresses than that the amount physically exists in the system. The main advantage of this memory is that the program may be larger than the size of the primary memory that physically exists. Using the concept of virtual memory the logical and the physical memory can be separated. This separation allows using of a large virtual memory for the developers over the actual physical memory in the system. It gives an illusion to the programmer that large memory locations are available at their end even though the system has a smaller main memory.

The address generated by the user program is called a virtual address. A set of virtual addresses makes the *virtual address space*. The set of main memory addresses or locations are called *memory space* or *physical address space*. Usually, the virtual address space is larger than the physical address space. To map between virtual addresses with physical addresses, memory mapping techniques are used by the memory controller of the system.

Consider a computer system with RAM of a storage capacity of 32K words. To specify a location in RAM with 32K physical address space (32K = 2^{15}) 15 bit is required. Consider that the system has 2^{20}= 1024K storage capacity auxiliary memory. Assume the memory space is M and the P is the address space. Hence M=32K and P=1024K. Thus the address bit of the instruction code will have 20 bits whereas the memory address must be specified by 15bit only. CPU will ask for reference instructions with the address of 20bit, but at this point, the reference address must be taken from the primary memory of the address with 15 bit rather than auxiliary memory. Thus there is a requirement of mapping of virtual addresses of 20 bit to physical 15 bit.

STOP TO CONSIDER

- Cache memory is typically integrated into CPU chip or placed on a separate chip that has direct connection with the CPU through a separate bus.
- To specify a location in RAM with 32K physical address space (32K = 2^{15}) 15 bit is required.
- Speed of CPU is very high then the average access rate of primary memory. Cache is used to remove this bottleneck between CPU and RAM.
- This bottlenecking problem can decrease the performance of the computer system.

Space for learners:

1.11 CLASSIFICATION OF MEMORY BASED ON THE ACCESS METHOD

Space for learners:

There are three types of memory access methods as Sequential access, Random access, and direct access.

1.11.1 Sequential access:

In this system, the stored data are accessed in affixed ordered manner i.e. in a specific linear specific manner. Here the access time depends on the location where the data exist. To go from memory location 1001 to 1010 in sequential access, it has to pass through all intervening memory locations. No one can jump from 1001 to 1010 directly as shown in figure 1.7. Examples of sequential media access memory devices are magnetic tape, magnetic disk, optical memories, DVDs, CDs, hard drives, etc.

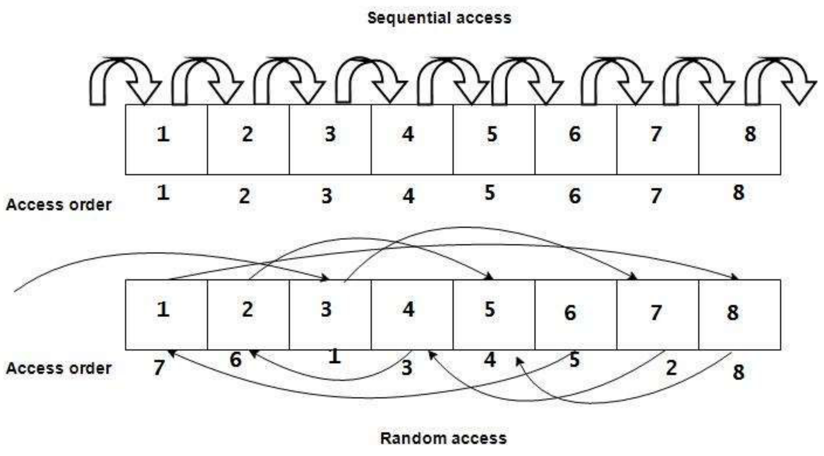


Figure 1.7 Sequential and random access method

1.11.2 Random access:

It refers to access data randomly from the storage device. In the random access method, one can jump from memory location 1001 to 1010 directly without passing through all the intervening locations i.e. 1002, 1003,…. etc. Examples of random access memory devices are disk, RAM, ROM.

1.11.3 Direct access:

In this method, a unique address has been assigned for each block or record based on physical location. It can be seen as a hybrid of random and sequential access methods. The direct access method is used in magnetic hard disks as it contains a huge number of rotating storage tracks. Each track is associated with its own read/write head. Magnetic tracks are accessed randomly, but within the track, the data are accessed sequentially. Magnetic hard disk is a good example of using a direct access method for accessing memory contents.

STOP TO CONSIDER

- Data stored in memory can be accessed in three different ways.
- Sequential access READ or Write data sequentially where as in Random access READ or WRITE operation are performed randomly on memory locations.
- Magnetic hard disk is good example of using direct access method for accessing memory contents.

Space for learners:

1.12 MEMORY MANAGEMENT HARDWARE

Space for learners:

To manage the operations performed by memory dedicated hardware is placed in between the processor and main memory called Memory Management Unit (MMU). If the processor does not have an on-chip MMU, then use an external MMU. The operations done by MMU are performed by the operating system. But to reduce the load on the operating system MMU is used. The logical address can be defined as the memory address which is being used by a program. A logical address represents or specifies the location of an instruction or data in a program relative to the starting address of the program. During the compilation of a program statement, the logical addresses are represented by a memory pointer consisting of two parts namely segment selector and offset. For a page-oriented system, the memory pointer has a page address and page offset. The physical address will be represented in terms of page number and page offset. The virtual memory concept is also performed by the MMU to provide a very large memory space to users. The concept of virtual memory allows the users to use more memory than that a system has in reality. A computer processor can access the data from the main memory during the execution of an instruction. For the execution of a program statement, it has to store or load into the main memory. The MMU allows users to store the program instructions into the secondary memory and it transfers a block of instructions to the main memory which is currently required by the processor. Similarly, the parts of the program statements are sending back to the secondary memories which are not being used by the processor currently. This to and fro data transferring process between main memory and secondary memory is known as swapping.

When a request for data or instruction is sent by the processor to the MMU by specifying a logical address, the MMU checks the segment containing that logical address in the main memory. If it is available in the physical memory then the MMU calculates the physical address corresponding to the logical address specified by the processor. If the required segment is not available in the physical memory then MMU interrupts the CPU. On receiving an interrupt signal from the MMU, the CPU access or read the desired segment from the secondary memory.

Space for learners:

1.13 SOLVED EXAMPLES

1. 16K x 8 RAM chips are used to construct 64K x 16 RAM. Calculate the required number of chips for construction.

   Solution: Number of chips required $=\frac{64\,K*16}{16\,K*8}$ = 8 chips

2. 1K x 4 RAM chips are used to construct 1K x 8 RAM. Calculate the required number of chips for construction.

   Solution: Number of chips required $=\frac{1\,K*8}{1\,K*4}$ = 2 chips

3. **Direct Mapping Question:** Assume a computer has 32-bit addresses. Each block stores 16 words. A direct-mapped cache has 256 blocks. In which block (line) of the cache would we look for each of the following addresses? Addresses are given in hexadecimal for convenience.

a. 1A2BC012

b. FFFF00FF

c. 12345678

d. C109D532

Solution: Of the 32-bit address, the last four bits denote the word on the line. Since four bits are used for one hex digit, the last digit of the address is the word on the line. With 256 blocks in the cache, we need 8 bits to denote the block number. This would be the third to last and second to last hex digit.

a. this would be blocking 01, which is block 1

b. this would be 0F which is block 15

c. this would be 67 which is block 103 (remember, 67 is a hex value)

d. this would be 53 which is block 83.

CHECK YOUR PROGRESS

1. Choose the correct options from the following: (Multiple choice questions)

   i. What is true about the memory unit?
      A. Memory is the collection of storage units or devices together.
      B. The memory unit stores the binary information in the form of bits.
      C. Both A and B
      D. None of the above

   ii. When the power of a computer system shuts down, then which type of memory loses its data?
      A. Non-volatile memory
      B. Volatile memory
      C. Both A and B
      D. None of the above

   iii. The fastest data access is provided using______________
      A. Cache memory
      B. DRAM
      C. SRAM
      D. Registers

   iv. The minimum time delay between two successive memory read operations is_________.
      A. Cycle time
      B. Latency
      C. Delay
      D. None of the above

Space for learners:

v. The effectiveness of the cache memory is based on the property of____________.
   A. Locality of reference
   B. Memory localization
   C. Memory size
   D. None of the above

vi. The drawback of building a large memory with DRAM is
   A. Large cost factor
   B. Inefficient memory organization
   C. Slow speed of operation
   D. All of the above

vii. The memory which is used to store the copy of data or instructions stored in larger memories, inside the CPU is called ____________.
   A. L1 cache
   B. L2 cache
   C. Registers
   D. TLB

viii. Four memory chips of 16 x 4 sizes have their address bases connected. The system will be of size
   A. 64 x 64
   B. 16 x 16
   C. 32 x 16
   D. 256 x 1

ix. In a virtual memory system, the address space specified by the address lines of the CPU must be ____________ than the physical memory size and __________ then the secondary storage size.
   A. Smaller, smaller
   B. Smaller, larger
   C. Larger, smaller
   D. Larger, larger

x. For the synchronisation of the read head we make use of a ____________.
   A. Framing bit
   B. Synchronization bit
   C. Clock
   D. Dirty bit

xi. The BOOT sector files of the system are stored in __________.
   A. RAM

Space for learners:

B. ROM
C. Hard disk
D. Fast solid-state chip in the motherboard.

xii. The technique where the controller is given complete access to the main memory is
A. Cycle stealing
B. Memory stealing
C. Memory con
D. Burst mode

xiii. How many address bits are required to represent a 32K memory?
A. 10bits
B. 12 bits
C. 14 bits
D. 16 bits

xiv. Which of the following memories stores the most number of bits?
A. 64 K x 8 memory
B. 1 M x 8 memory
C. 32 M x 8 memory
D. 64 x 6 memory

xv. In a virtual memory system, the addresses used by the programmer belongs to
A. Memory space
B. Physical address
C. Address space
D. Main memory address

Space for learners:

1.14 SUMMING UP

- The memory unit is a key part of a computer system.
- Computer memory can be divided into two categories namely primary and secondary memory.
- The central processing unit of a system directly communicates with the primary memory.
- The processor can access the secondary memory through primary memory.

- Primary memory can be categorized as RAM and ROM.
- To increase the throughput of a system cache memory is used in between the primary memory and CPU.
- Different types of ROMS are available such as ROM, PROM, EPROM, and EEPROM.
- If the required chip size is M x N and if the chip capacity is m x n, then the number of the required chip is k = $\frac{M*N}{m*n}$
- A set of physical addresses is known as memory space.
- The address space can be divided into groups of equal size known as a page.
- The memory space is broken into groups of equal size called blocks.
- A computer processing speed can represent using the address bit.
- Big-endian and Little-endian assignments are used in byte addressing.
- MMU is used to control the communication between the CPU and memory.
- The concept of virtual memory allows users to use memory space during programming which does not exist physically.
- Lower order bytes are used to represent MSB in big-endian assignments.
- Lower order bytes are used to represent LSB in little-endian assignments.
- In modern practices, each successive address refers to successive byte locations in memory
- Cache memory can be categorized into three different levels such as L1, L2, and L3 cache.
- SRAM is an on-chip memory with very little access time whereas DRAM is off-chip memory with a large access time in comparison to SRAM.

Space for learners:

1.15 ANSWERS TO CHECK YOUR PROGRESS

Space for learners:

Answers to question no. 1:

i. C	ii. B	iii. D	iv. A
v. A	vi. C	vii. A	viii. B
ix. C	x. C	xi.B	xii.D
xiii.D	xiv.C	xv. C	

1.16 POSSIBLE QUESTIONS

1) How many 128 x 8 RAM chips are needed to provide a memory capacity of 2048 bytes?
2) How many 128 x 8 RAM chips are needed to provide a memory capacity of 4096 x 16?
3) What is the bit storage capacity of a ROM with a 1024 x 8 organization?
4) Find the total number of cells in a 64k x 8 memory chip.
5) What is virtual memory?
6) Differentiate between ROM and EEPROM.
7) What are the key factors for memory efficiency?
8) What is memory access time?
9) What is clock cycle and CPU burst time?
10) Differentiate between sequential and random access?
11) What are SRAM and DRAM?
12) What is a memory chip? How the number of chips is calculated for the required number of memory?
13) What is memory hierarchy?
14) Why DRAM is slower than SRAM?
15) What is cache memory?
16) Explain the memory hierarchy with a block diagram.
17) What are the classifications of memory depending on access method, Explain?

18) Explain the organization of a RAM chip.
19) What are the different types of ROM? Explain the differences between them.
20) Explain some data structures which are using sequential access and random access.
21) How direct memory is different from random access memory?
22) Discuss static and dynamic RAM.
23) Explain the functions of the memory management unit (MMU) of a computer system.

Space for learners:

1.17 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.

---×---

UNIT 2 : CACHE MEMORY

Space for learners:

Unit Structure:

2.1 INTRODUCTION

To compensate for the speed of primary memory access time and CPU, a high speed memory is used called cache memory. Cache memory increases the processing speed of the CPU by making the required data available to it. Thus the cache memory has a great role in increasing the throughput of a system. It is placed in between the processor and main memory. The memory access time of cache memory is very high in comparison to main memory and compatible with the speed of the processor. The cache is used to store the program segment currently executed by the CPU and the data used by the CPU frequently. Since the memory space of the cache is much smaller than the main memory, mapping is required to identify the location in the main memory as specified by the CPU. Using cache replacements algorithms the contents of the cache can be changed by new program segments as required by the processor.

Space for learners:

2.2 UNIT OBJECTIVES

The primary objectives of the chapter are

- to know about cache memory and its use
- to understand how to measure the performance of cache memory
- to explore what are the operations of cache memory?
- to learn about the mapping process of cache memory.
- to find the different mapping processes
- to visualize cache replacement policies.
- to understand the cache optimization techniques.
- to know how to write into the cache.
- to discuss the different levels of cache memory.
- to learn about cache coherency

2.3 BASIC OPERATIONS

Space for learners:

When the processor needs a particular data during its execution, at first it searches the data in cache memory. If it is found then the content is extracted from the memory location of cache as specified by the processor. If the word addressed by the CPU is not available in the cache memory, the main memory is accessed to read the word. The program segment or a block of the word containing the desired one will be transferred from the main memory to the cache memory. In multilevel cache, it can be categorized into two; internal cache, typically located inside the CPU chip and external cache, normally placed in the system board. Internal caches are known as primary or L1 cache and the range may have within 1– 32 KB. External caches are known as secondary or L2 cache and the range may vary between the range 64 KB – 1 MB.

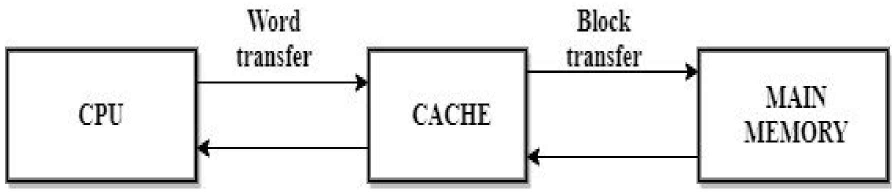


Figure 2.1 Cache memory

STOP TO CONSIDER

- In multilevel cache, it can be categorized into two; internal cache, typically located inside the CPU chip and external cache, normally placed in the system board.
- Internal caches are known as primary or L1 cache and the range may have within 1kb – 32kb.
- External caches are known as secondary or L2 cache and the range may vary between the range 64kb – 1mb.

Space for learners:

2.4 PERFORMANCE

The performance of cache memory can be determined by the ratio of finding the required data by the processor. If the required data is available in cache memory then it can be defined as a HIT, if it is not available in the cache then it is called a MISS. Three different types of cache miss may exist namely – compulsory miss, conflict miss and capacity miss. Compulsory miss may occur when a memory location is accessed for the first time. Conflict miss can occur due to insufficient space when two blocks are mapped on the same location. Capacity miss may take place due to smaller space in cache memory. The time taken to check the presence of data in the cache is called hit latency. For every hit, the CPU accesses the data from the cache directly but for a miss, the CPU has to wait for responding from the main memory. The block of data will be transferred from the main memory to cache memory and then the required word will transfer from the cache to the CPU. The ratio between the hit and the total number of references by the CPU (the summation of hit and miss) can be defined as the hit ratio. The hit ratio "h" always lies between the range 0 and 1. Let us consider that the

h is the hit ratio, $\mathbf{t_m}$ is the memory access time, $\mathbf{t_c}$ is the cache access time

$\bar{t}$ is the average access time.

Then the average access time $\bar{t}$ can be calculated by the relation:

$\bar{t} = ht_c + (1 - h)(t_c + t_m)$ …………………………….. (1)

The relation (1) is derived using the fact that for a cache hit, the main memory will not be accessed by the processor. For a miss, both the cache and main memory will be accessed by the CPU. Consider that the ratio between cache and main memory access time is $\gamma = \frac{t_m}{t_c}$ then the efficiency (Λ) of a system using cache memory can be derived as:

$$\begin{aligned}
\Lambda &= \frac{t_c}{\bar{t}} \\
&= \frac{t_c}{ht_c + (\mathbf{1} - h)(t_c + t_m)} \\
&= \frac{t_c}{t_c\left[h + (\mathbf{1} - h)\left(\mathbf{1} + \frac{t_m}{t_c}\right)\right]} \\
&= \frac{\mathbf{1}}{h + (\mathbf{1} - h)(\mathbf{1} + \gamma)} \\
&= \frac{\mathbf{1}}{h + \mathbf{1} - h + \gamma(\mathbf{1} - h)} \\
&= \frac{\mathbf{1}}{\mathbf{1} + \gamma(\mathbf{1} - h)}
\end{aligned}$$

For the value of h = 1, the efficiency Λ = 1, i.e. efficiency will be maximum for h = 1 or all the CPU references are confined to the cache.

Space for learners:

STOP TO CONSIDER

- An INTEL motherboard of 100MHz consume 180ns to retrieve information from main memory, whereas 45ns from the cache memory.
- Static RAM (SRAM) is typically used to build cache memory.
- Systems with Multi-core CPUs are generally used a separate L1 and L2 cache for each core and L3 is shared by each core.

Space for learners:

Example 2.1: Calculate average access time ($\bar{t}$), the ratio between main memory access time and cache access time (γ),and efficiency (Λ) of a memory system whose parameters are indicated as: t_c=150 ns, t_m=950 ns, and h=0.90.

Solution: Since the average access time $\bar{t}=ht_c+(1-h)(t_c+t_m)$

$$=0.90*150+(1-0.90)(150+950)$$
$$=135+0.1(1100)$$
$$=245\text{ns}$$

And since the $\gamma = \frac{t_m}{t_c} = \frac{\mathbf{950}}{\mathbf{150}} = \mathbf{6.33}$

And efficiency $\Lambda = \frac{\mathbf{1}}{\mathbf{1}+\gamma(\mathbf{1}-h)}$

$$= \frac{\mathbf{1}}{\mathbf{1+6.33(1-0.9)}} = \frac{\mathbf{1}}{\mathbf{1+0.633}} = \frac{\mathbf{1}}{\mathbf{1+0.633}} = \mathbf{0.612}$$

Example 2.2: The access time of cache memory is 50 ns. And the access time for the main memory is 500 ns. It is estimated that 80% of the main memory requests are for reading operation and the remaining are for the write operation. The hit ratio for reading operation is 0.09 and a write-through policy is used.

Space for learners:

a. Compute the average access time for the memory read cycles only?

b. Calculate the average access time for both read and write requests?

c. What is the hit ratio regarding the write cycle?

Solution:

a. Since the average access time $\bar{t}=ht_c+(1-h)(t_c+t_m)$

$=0.90*50+(1-0.90)(50+500)$

$=45+55$

$=100ns$

b. For both read and write cycle

Average access time = P_r * average access time for read + $(1 - P_r)$ * t_m

$= 0.8 * 100 + 0.2 * 500$

$= 80 + 100$

$= 180$ ns.

c. Hit ratio when write cycle is also considered is

$h = P_r * h_r + (1 - P_r) * h_w$ [h_w is the hit ratio for write cycle]

$= 0.8 * 0.9 + 0.2 * 0$

$= 0.72$

2.5 MAPPING PROCESS

There are three different types of mapping techniques in cache organization such as

a. Associative mapping

b. Direct mapping

c. Set – associative mapping

2.5.1 Associative mapping

In the case of the associative mapping procedure, each of the cache memory contents is associated with an address. But during the execution of a program statement, the data is not read or fetch by referring to or specifying any memory address. Instead of it, the data are searched by matching with the contents. In associative mapping, the cache memory contains the data along with the main memory address of the corresponding data as shown in Figure 2.2. The address bit sent by the CPU for searching the required data in the cache memory is matched with the stored addresses in the cache. If any address is matched, the corresponding word

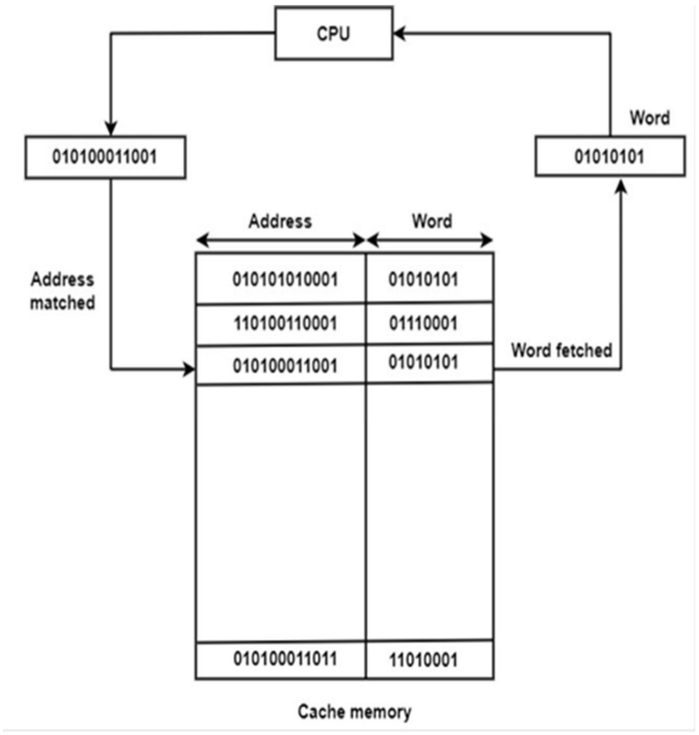


Figure 2.2 Associative mapping in cache memory

from that memory location will be fetched by the CPU. For a miss or if no match is found for the required word, then it will be searched in the main memory. Then the word from the main memory along with the address will be transferred into the cache memory. If the cache is full, using any replacement technique must make room for the new word.

Associative mapping is a very fast access method. But the manufacturing difficulties and cost are more in comparison to other mapping methods.

2.5.2 Direct mapping

Consider a computer system with the main memory storage capacity is 4K, i.e. 4 x 1024 = 2^{12} bytes. Then the required number of bits to address the main memory location will be 12. Consider a cache memory of 1K = 2^{10} bytes, i.e. 10 bits are required to address a cache memory location. Thus the main memory required a 12-bit address line whereas the cache memory required only 10 bits of the address. In the direct mapping method, the address sent by the CPU is divided into two parts namely tag field and index field. The index field contains an equal number of bits that are required to address a word in cache

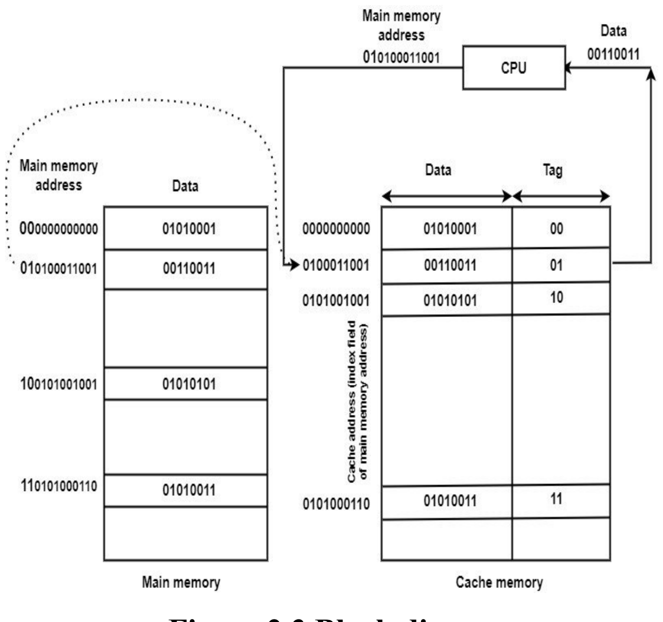


Figure 2.3 Block diagram

Space for learners:

memory. The remaining bits are used in the tag field. If a system contains the main memory of capacity **2^m** and cache of capacity **2^n**, then the bits in the index field will be n bits and in the tag field is an **(m-n)** bit. In the example cited above, the index field and the tag field are consist of 10 bits and 2 bits respectively.

In direct mapping, the cache memory stores the data as well as the tag field as shown in Figure 2.3. In the cache, the words are stored in a memory location as the index field defined. When an address is requested by the CPU, the index part of the address is used to get the location in the cache memory. If the tag of the cache is matched with the tag of the requested address, the word will be fetched by the CPU. Else there will be a miss and the data will be searched in the main memory. For a miss, the block of data from the main memory has to be transferred into the cache memory by dividing the main memory address into index and tag fields. The main disadvantage of direct mapping is that, if the index field is the same for more than one word in cache memory with a different tag value, the hit ratio may drop considerably.

2.5.3 Set Associative Mapping

In direct mapping, two words or data of a similar index field cannot be stored at the same time. To overcome this drawback of direct mapping, the third method of mapping is used which is known as set-associative mapping. In this method, cache memories are allowed to store more than one word with a similar index in the same word location along with a different tag. The number of tags–data pair in the one-word location of the cache is said to be as a set. An example of set-associative mapping has been depicted in Figure 2.4. As shown in the figure, the word stored in the memory addresses 001010011001 and 011010011001 of main memory is stored in cache memory at index address 1010011001. Similarly, the word stored at address 101010000111and 111010000111 of main memory is stored in cache memory index address 1010000111.

INDEX	TAG	DATA	TAG	DATA
1010011001	00	10000111	01	00111010
1010000111	10	10100110	11	11100001

Figure 2.5 Block diagram of set associative

STOP TO CONSIDER

- If a system contains main memory of capacity **2^m** and cache of capacity **2^n**, then the bits in index field will be n bits and in tag field is an **(m-n)** bit.
- In direct mapping when an address is request by the CPU, the index part of the address are used to get the location in the cache memory.

2.6 CACHE REPLACEMENT POLICIES

When the cache memory of a system is full, then cache replacement policies are used to make a decision about which page or data has to be replaced from the cache to make room for new data. The main problem in cache is that how to identify the page or data to be removed from cache memory. Lots of algorithms for cache replacement are being developed. The efficiency of those algorithms depends on the factors such as time, number of misses and balancing cost, etc. An efficient algorithm takes less time, lower number of miss rate and balancing cost. Some of the cache replacement algorithms are discussed as follows.

Space for learners:

Space for learners:

2.6.1 Least Recently Used (LRU) Algorithm

This algorithm discards the least recently used data from the cache to make space for new data. A variable known as the **aging bit** is used to keep the record of all data items such as which data is used when or kept at what time in the cache. It is one of the most popular algorithms among all others as it provides better performance. The implementation policy of the LRU algorithm is also very simple and time and space overhead are constant.

Example 2.3: Let us consider a set of data items 7 0 1 2 0 3 0 4 2 3 0 3 2, which have to load into the cache memory of size 4 word. How many misses will occur if the LRU technique is being used as a cache replacement policy?

Solution: Initially the cache was empty, so for the first four data items namely 7, 0, 1, and 2, there will be **4 MISS** as shown in figure 2.6.

For the 5th element 0, does already exist in the cache so **0 MISS**.

For the 6th element 3, which does not exist in the cache, it will replace the least recently used 7 from the cache – **1 MISS**

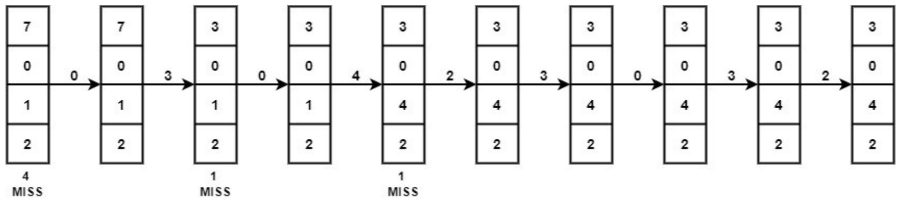


Figure 2.6 Example of Cache least recently used replacements

For the 7th element 0, does already exist in the cache so **0 MISS**.

For the 8th element 4, which does not exist in the cache, it will replace the least recently used 1 from the cache – **1 MISS**

For further referencing all the data exist in the cache, so no more replacement is required for any one of them. The total number of MISS is 6 and the number of hits is 7.

Space for learners:

2.6.2 Least frequently used (LFU) algorithm

The LFU counts the number of uses of a particular data item or it counts how frequently the data item has been used. The data which is used very few will be identified and removed from the cache first. If all the data items in the cache have the same count then randomly any one of the items has been chosen and deleted. The min-heap data structure is a suitable one to implement this algorithm.

2.6.3 First in first out (FIFO) algorithm

FIFO algorithm removes the data which has come first into the cache and has not been used for a long time. It is the simplest algorithm to implement. Here the system keeps track of all the blocks or words in memory in a queue. The oldest page is in the front of the queue. When a replacement is required, the data from the front of the queue will be selected for removal.

Belady's anomaly – proves that it is possible to have more MISS for an increasing number of frames while using the FIFO replacement algorithm. For example consider a set of data items such as 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 and 3 slots frame. The number of a miss for 3 slots frame will be 9, whereas the MISS is 10 for 4 numbers of slots in a frame.

Example 2.4: Let us consider a block referencing strings 1, 3, 0, 3, 5, 6, 3 with 4 block frames. Find the number of misses.

Solution: Initially the frame is empty, so for the first three elements 1, 3, and 0, there will be three miss consecutively. Further referencing has been shown in the following figure 2.7.

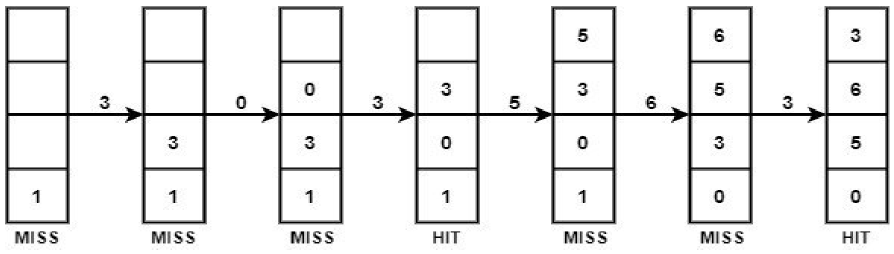


Figure 2.7 Example of FIFO replacement algorithms

A third iteration when 3 comes, is already in the queue, so one hit occurred. Again at the 7th iteration when 3 come, one hit occurred. At steps 5th when 6 come, the data do not exist in the queue. The element entered first into the queue i.e. 1 will be replaced by 6 as shown in figure 2.7.

2.6.4 Segmented LRU (SLRU) algorithm

SLRU algorithm divided the cache memory into two parts as protected and unprotected. The protected part is reserved for the most used objects. Once the first request for an object is done by the CPU; it will be transferred into the unprotected section. The least recently used technique is used to manage both the portion.

2.6.5 Optimal block replacement

In this method that block will be replaced from the cache which would not be used for a longer period in the future. Optimal page

Space for learners:

replacement is theoretically perfect, but the operating systems could not know or guess the future request.

Example 2.5 Consider a set of cache block references as 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 slots frame. Find the number of MISS that occurred during cache access.

Solution: Initially there will be four miss for the first four data items 7, 0, 1, and 2 as the slots were empty.

0 is already exit, **0 – MISS or HIT**,

When 3 came, the algorithms identified 7 as the not-used item for the longest period in the future and replace it by 3. – **1 MISS.**

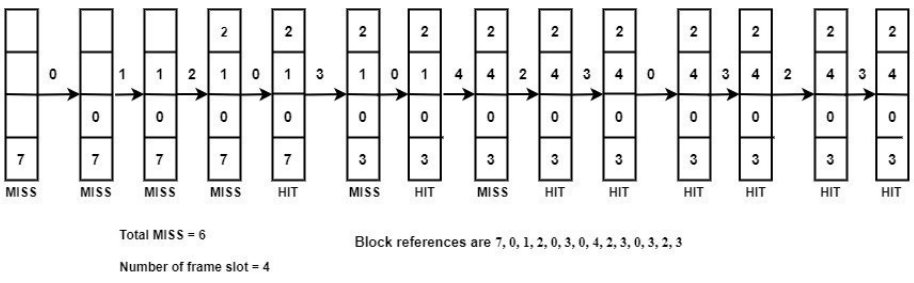


Figure 2.8 Example of Cache optimal block replacements

0 is already exit, **0 – MISS**,

When 4 came, the algorithms identified 1 as the not-used item for the longest period in the future and replace it by 4. – **1 MISS**. Thus there will be 6 misses.

2.6.6 Random replacement (RR) algorithm

The RR algorithm randomly selects any of the data items from the cache memory and replaces it with the required one. It never keeps track of the history of removable data items and does not follow any data structure.

Space for learners:

2.6.7 Pseudo – least recently used (PLRU) algorithm

It is one of the most popular and common block replacement policies for the current generation's cache memory. It is widely used by the industry and a common policy for AMD and INTEL products. It uses the data structure binary tree for saving the status of cache memory and hence it is also known as Tree – LRU. The tree structure is used to identify the block position which is to be replaced in case of a miss.

Space for learners:

2.6.8 Lowest latency first (LLF)

LLF algorithm keeps the average and minimum latency by removing the objects with the lowest latency. It shows the best performance during the execution of database queries in the relational database.

All the algorithms discussed above can be classified into several classes such as –

a. Recency-based algorithms,
b. Frequency-based algorithms
c. Function-based algorithms
d. Randomized algorithm

2.7 CACHE OPTIMIZATION TECHNIQUE

Cache optimization can be achieved by reducing the miss penalty, miss rate, and hit time and increasing the cache bandwidth. These can be obtained using different optimization techniques.
To decrease the gap between CPU cycle and memory latency, a multilevel cache can be used. Generally, the cache memory can be

categorized into three levels such as L1, L2, and L3 cache. L1 is comparatively the smallest and fastest cache memory in comparison to L2 and L3 levels. It is located within the CPU itself and hence it is called on-chip memory. L2 is faster than L3 cache. L3 is larger and slower in comparison to other levels of cache memory. In a multiprocessor system, each processor has own L1 and L2 cache memories and the L3 cache is shared by all processor. Miss rate of L1 cache can be reduced by introducing L2 cache.

User-level cache-control (ULCC) is another technique through which space allocation in the cache can be controlled by the user. Less hit rate and minimal cache pollution can be produced using this technique. The implementation is very complex.

Cache memory optimization can be achieved by optimizing the loop in compiler-level implementation. A set of compiler algorithms are being used to predict the data to be reuse in near future. This will help to achieve a better hit ratio in cache access.

The performance of the cache can be improved by producing the next data to be used by the cache. To produce the next data a process called data perfecting can be used in advance.

2.8 WRITING INTO THE CACHE

The cache is a technique to keep a copy of one or more blocks of data from the main memory into the fastest memory storage such that the processor can access it easily. Cache mostly works as a buffer in between the processor and RAM and increases the speed of the processor by making the data available. Whenever the CPU wants to write data or a word, at first it checks the address where the word to be written is available in cache or not. If the address is available in cache memory then it is known as a **write-hit**. During the write operation, if the main memory is not updated

Space for learners:

simultaneously, it may lead to inconsistency of data. That is the content in cache and main memory may be different for the same reference address. If the system memory is shared with more than one device then problems may arise due to this inconsistent data. Hence the two methods write through and write back come into the picture to perform the write operation in cache effectively and efficiently. If the referred address is not available in the cache during a written request a **write-miss** will occur. For a write-miss, two other processes are being used to maintain the data consistency in cache and main memory namely **write allocation** and **write around**.

Space for learners:

2.8.1 Write through

When the number of the write operations in cache is less, then this process is used. It is comparatively simpler and reliable to perform the write operation. In write-through, both the memory cache and main memory are updated simultaneously. During a write operation, a data or word has to write in both the memory locations and due to this, the write-through process experienced delays in the write operation. This process has solved the problem of inconsistent data but raises a question about the use of cache memory during the write operation. Because the access of main memory along with the cache during write operation increases the cost of the write operation and decreases the CPU performance.

2.8.2 Write back

In this process, the cache is updated during the write operation and the main memory is updated later.

Space for learners:

2.8.3 Dirty bit

A status bit is used to indicate whether the data present in the cache memory is modified or not during a write operation. It is known as dirty bit. If the status bit is set to clean-bit, no need to update the main memory later as the data is not modified in the cache. For a dirty bit, the main memory has to be updated as it represents that the cache has been updated during the write operation. But if power fails due to any cause the modified data will be lost in the cache. Lost data from the cache cannot be restored.

2.8.4 Write allocation

In this process, data has to be loaded from the main memory into cache memory and then updated. It works with both write-through and write-back processes.

2.8.5 Write around

Write around process allows for writing or updating the main memory without interrupting the cache memory.

2.9 CACHE COHERENCE

During the write operation of cache memory, data inconsistency may occur among adjacent or within the same level of the memory hierarchy. It is possible to have many copies of one instruction operand in a shared memory multiprocessor system. If any operand value is changed in one memory, then it should reflect in the main memory as well as all the levels of cache memories simultaneously.

Consider a system with three processors P1,P2, and P3. The P1 reads the data X with the value 25 from the main memory and stores it into the cache. The P2 also reads the same data X = 25 from the main memory and stores it into the cache. In the next instruction cycle, P1 writes X as 55 locally into cache but not updated into the RAM. If P3 reads the data from RAM, what value it will get against X? For the main memory and P2, it will be 25, whereas for P1 it will be 55. Thus in caches write operation can create multiple copies of data in different levels of cache and main memory which may lead to the cache coherence problem.

Cache coherence can be defined as a protocol or discipline which ensures that the values of shared operands are propagated throughout the system in a timely fashion.
Generally, cache coherence may occur from three different sources of inconsistency problem –

i. Sharing of writable data
ii. Process migration
iii. Input / Output (I/O) activity.

Space for learners:

2.9.1 Sharing of writable data

When two processors P1 and P2 read the same word X from shared memory into their local cache and P1 writes to the word as X1 using the write-through method, then the shared memory will be updated from X to X1. Now when P2 will read the data from its local cache it will be X which is become outdated as shown in Figure 2.9.

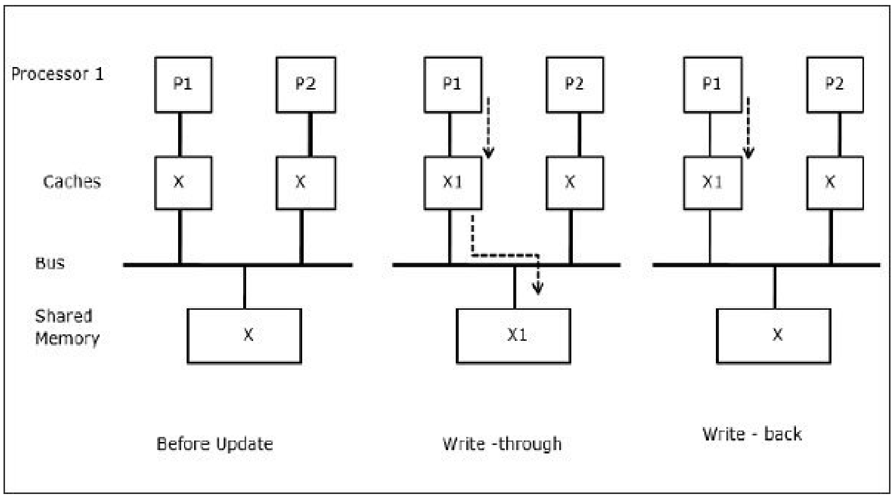


Figure 2.9Sharing of writable data

Space for learners:

2.9.2 Process migration

Consider that the process P1 has a data operand X and P2 does not hold any data in its cache. The process P2 first writes on data operand X as X1 and then migrated to P1. Now the process P1 starts reading outdated data X. So P1 writes the data operand X onto the main memory and migrated to P2 as shown in figure 2.10. After migration P2 will start reading the data element and found X in the main memory which is outdated for P2.

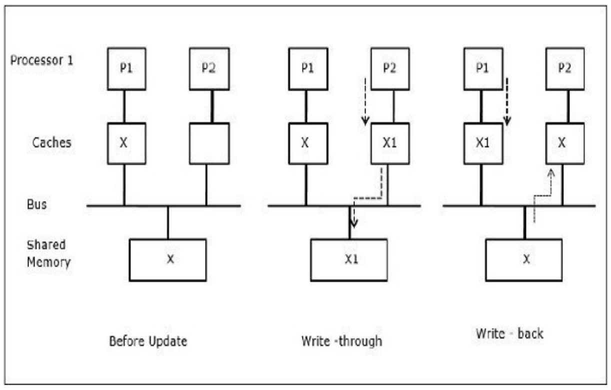


Figure 2.10Process migration

2.9.3 I/O activity

As shown in Figure 2.11, an input/output device is added to the bus in a multi-processor system. As shown in the figure initially both the processor P1 and P2 holds the data operand X. If the I/O peripheral write the data operand as X1 into the main memory, then the processes P1 and P2 will get outdated data in the successive read operation. Then the process P1 will modify the operand as X into the main memory as well in the local cache. Now if the I/O device wants to transfer the data, it will get a copy of outdated data.

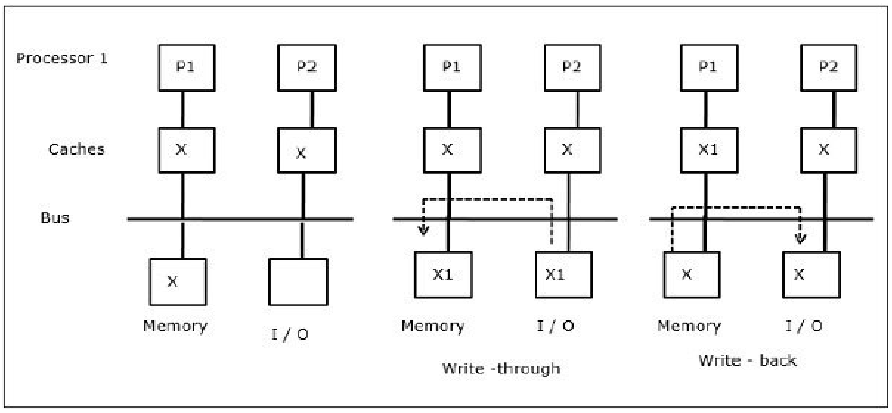


Figure 2.11 I/O activity

Space for learners:

2.10 COHERENCY MECHANISM

Coherency mechanisms are categorized into four categories –

2.10.1 Directory-based

In this method, the data which is to be shared is placed into a common directory, which helps to maintain the cache coherency. Before each read / write operation by the processor from the main memory into the local cache, the common directory have to be checked once. Once the directory is changed by any processor, immediately invalidates or updates the other cache with that entry.

2.10.2 Snooping

It is a process where the individual cache monitors the address lines for checking the memory location where the cache is mapped. When a write operation is observed at that location in the main memory, the cache controller invalidates its copy of the snooped memory location. It is known as the write invalidate protocol.

2.10.3 Snarfing

It is quite similar to snooping. This method is used to monitor both the memory location that has been cached as well as the actual information that is stored in the main memory. During a memory write operation, the cache can be updated by new data.

CHECK YOUR PROGRESS

1. **Choose the correct options from the following for each question:**

a. Assume that there are 3-page frames that are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is__________.
(A) 5
(B) 6
(C) 7
(D) 8

b. Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4, 1 on a demand paged virtual memory system running on a computer that main memory size of 3 pages frames which are initially empty. Let LRU, FIFO, and OPTIMAL denote the number of page faults under the corresponding page replacements policy. Then
(A) OPTIMAL < LRU < FIFO
(B) OPTIMAL < FIFO < LRU
(C) OPTIMAL = LRU
(D) OPTIMAL = FIFO

c. A virtual memory system uses First in First out (FIFO) block replacement policy and allocates a fixed number of frames to a process. Consider the following statements:

P: Increasing the number of page frames allocated to a process sometimes increases the page fault rate.
Q: Some programs do not exhibit locality of reference.

Which one of the following is TRUE?
(A) Both P and Q are true, and Q is the reason for P
(B) Both P and Q are true, but Q is not the reason for P.
(C) P is false, but Q is true
(D) Both P and Q are false

d. A process has been allocated 3-page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1, 2, 1, 3, 7, 4, 5, 6, 3, and 1
If an optimal page replacement policy is used, how many page faults occur for the above reference string?
(A) 7
(B) 8
(C) 9
(D) 10

e. A system uses 3-page frames for storing process pages in the main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below?
4, 7, 6, 1, 7, 6, 1, 2, 7, 2
(A) 4
(B) 5
(C) 6
(D) 7

f. The optimal page replacement algorithm will select the page that
(A) Has not been used for the longest time in the past.
(B) Will not be used for the longest time in the future.
(C) Has been used least number of times.
(D) Has been used most number of times.

g. Consider a virtual memory system with a FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will

Space for learners:

(A) always decrease the number of page faults
(B) always increase the number of page faults
(C) sometimes increase the number of page faults
(D) never affect the number of page faults

h. A system uses a FIFO policy for page replacement. It has 4-page frames with no pages loaded, to begin with. The system first accesses 100 distinct pages in some order and then accesses the same 100 pages but now in the reverse order. How many page faults will occur?

(A) 196
(B) 192
(C) 197
(D) 195

i. Which of the following is not a written policy to avoid cache coherence?
(A) Write through
(B) Write within
(C) Write back
(D) Buffered write

j. The transfer between CPU and cache is ____________________.
(A) Block transfer
(B) Word transfer
(C) Set transfer
(D) Associative transfer

k. Which of the following is a common cache?
(A) DIMM
(B) SIMM
(C) TLB
(D) Cache

l. How many possibilities of mapping does a direct-mapped cache have?
(A) 1
(B) 2
(C) 3
(D) 4

Space for learners:

m. In which writing scheme does all the data writes go through to the main memory and update the system and cache?
(A) Write-through
(B) Write-back
(C) Write-buffering
(D) No caching of writing cycle

n. In which writing scheme does the cache is updated but the main memory is not updated?
(A) Write-through
(B) Write-back
(C) Write-buffering
(D) None of these

o. What is the main idea of the writing scheme in the cache memory?
(A) Debugging
(B) Accessing data
(C) Bus snooping
(D) Write allocate

2. Answer the following questions and fill up the blanks:

(A) Which cache has a separate comparator for each entry?
(B) What is the disadvantage of a fully associative cache?
(C) Which mechanism splits the external memory storage into memory pages?
(D) Which of the following cache mapping can prevent bus thrashing?
(E) Which cache mapping has a sequential execution?
(F) Which address is used for a tag?
(G) What do you mean by locality of reference?
(H) The number of failed attempts to access memory, stated in the form of a fraction is called as ______________.
(I) The extra time needed to bring the data into cache memory in case of a miss is called as __________________.
(J) The counter that keeps track of how many times a block is most likely used is __________________.

Space for learners:

2.11 SUMMING UP

Space for learners:

- Cache memory is smaller in size and one of the faster memory used in a computer system.
- The cache is used to place in between CPU and RAM.
- The memory access time of cache memory is very high in comparison to the main memory
- L1 cache and L2 cache may embed on the CPU chip, hence it is known as an on-chip cache.
- The cache is a very high-speed memory and is used to increase the processing speed by making the data available to the CPU at a rapid rate.
- Cache works as a buffer between the CPU and the RAM.
- Performance of cache memory is measured in terms of hit-ratio.
- If the CPU finds the referred address in the cache then it can be defined as a hit.
- If the CPU does not find the referred address in the cache then it can be defined as a miss.
- The ratio between the hit and the total amount of address referred by the CPU can be defined as hit-ratio.
- Through the mapping process, data can be transfer from main memory to cache memory.
- There are three different types of mapping processes in cache memory such as – associative mapping, direct mapping, and set-associative mapping.
- In associative mapping, the cache memory contains the data along with the address references of that data in the main memory.
- Direct mapping divides the main memory reference done by the CPU into two fields – index and tag field.

- If a system contains the main memory of capacity $\mathbf{2^m}$ and cache of capacity $\mathbf{2^n}$, then the bits in the index field will be n bits and in the tag field is an **(m-n)** bit in the direct mapping.
- Cache replacement policies are used to make room for the new data in cache memory if it is full.
- Some of the cache replacement policies are LRU, LFU, FIFO, RR, etc.
- Belady's anomaly – proves that it is possible to have more MISS for an increasing number of frames while using the FIFO replacement algorithm.
- User-level cache-control (ULCC) is one technique for block replacement, through which space allocation in the cache can be controlled by the user.
- Cache optimization can be achieved by reducing the miss penalty, miss rate, and hit time and increasing the cache bandwidth.
- Generally, the cache memory can be categorized into three levels such as L1, L2, and L3 cache.
- When the CPU wants to write into cache and the CPU referred address is available in cache memory then it is known as a **write-hit**.
- When the CPU wants to write into cache and the CPU referred address is not available in cache memory then it is known as a **write-miss**.
- To write into cache two methods are used – write through and write back.
- In the write-through process, both the memory cache and RAM are writing simultaneously.
- In the write-back process, the cache is used to write first and the main memory is updated later with the help of dirty-bit.

Space for learners:

- Two other processes write-allocation and write around are used, when a write miss has occurred.

Space for learners:

2.12 ANSWERS TO CHECK YOUR PROGRESS

Answers to the question number 1:

a) C
b) C
c) B
d) A
e) C
f) B
g) C
h) A
i) B
j) B
k) C
l) A
m) A
n) B
o) C

Answers to the question number 2:

(A) Fully associative cache.
(B) Hardware
(C) Index mechanism
(D) N-way set associative
(E) Burst fill
(F) Logical address
(G) The surroundings of the recently accessed block are called the locality of reference.

(H) MISS rate

(I) MISS penalty

(J) Reference counter

2.13 POSSIBLE QUESTIONS

1. Consider block reference strings 1, 3, 0, 3, 5, 6, and a block frame size 3 is used. Count the cache block miss when the FIFO replacement algorithm is used.
2. Consider the reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1. Count the number of miss using FIFO page replacement algorithm.
3. Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 block frame. Find number of miss using optimal block replacement technique.
4. Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 block frames. Find the number of misses using the Least recently used method.
5. Consider page reference strings 1, 3, 0, 3, 5, 6 with the frame size of 3. Find the number of misses using the FIFO replacement technique.
6. What is cache memory? Explain the role of cache memory in program statements execution.
7. Explain different cache mapping processes for example.
8. Why block replacement is necessary for cache memory? What are the replacement policies; explain the pros and cons of each.
9. Why cache optimization is required? Discuss any two cache optimization techniques.
10. What types of problems may arise during cache write and how it can be solved? Explain.
11. What is Belady's anomaly? Explain with an example.

Space for learners:

Space for learners:

12. How the multilevel cache is implemented?
13. Discuss the factors on which the cache optimization techniques are dependent.
14. How in compiler level cache memory can be optimized? Explain.
15. Define the terms: cache access time, efficiency, average access time, hit-ratio, miss, memory access time.

2.14 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.

UNIT 3: VIRTUAL MEMORY AND PAGING

Space for learners:

Unit Structure:

3.1 INTRODUCTION

Even though the focus of the subject is computer hardware, there is one area of software that needs to be addressed and that is the operating system. An operating system is a software that acts as an interface between a computer hardware and computer user. The operating system manages computer hardware, software resources and allocates resources and services, such as memory, processors and devices. One of the most important function of operating system is memory management that includes the hardware support in processor for paging, virtual memory and segmentation. Virtual memory allows a program with memory space larger than the size of the main memory to be available in the system. This is possible by allowing only that section of the code that is active at that point of time without the need of having all instructions and data of the process being present in main memory at the same time. The concept of paging and segmentation eliminates the need of allocating main memory to the process in contiguous manner.

Space for learners:

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Explain the mapping of logical address to physical address using paging memory management scheme.
- Analyze and solve problems on paging.
- Explain the working of paging hardware.
- Explain the mapping of logical address to physical address using segmentation.
- Explain the working of segmentation hardware.
- Explain Virtual memory management scheme.

3.3 PAGING

To understand the concept of paging we have to go through the following concepts:

- Process: It is a program in execution or a program placed in main memory for execution.
- Logical Address: It is the address that is generated by the CPU for a program while it is running. As the address does not exist physically it is also called virtual address. The hardware unit of memory known as memory management unit (MMU) maps logical address to physical address.
- Physical Address: A physical address is the actual address in the main memory.

Paging is a memory management scheme that is used to map CPU generated logical address of a process to physical address in main memory. A process consists of fixed size blocks; Figure 3.1 shows an example of a process with 4 blocks each of size 1 kilobyte. Size of a block depends upon architecture of the computer and varies between 512 bytes to 16 megabytes.

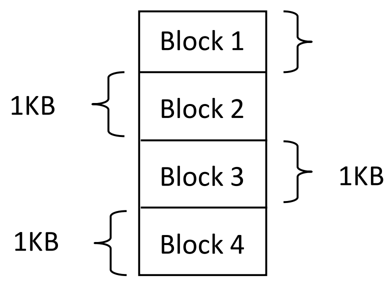


Figure 3.1: A Process with 4 blocks each of size 1 kilobyte.

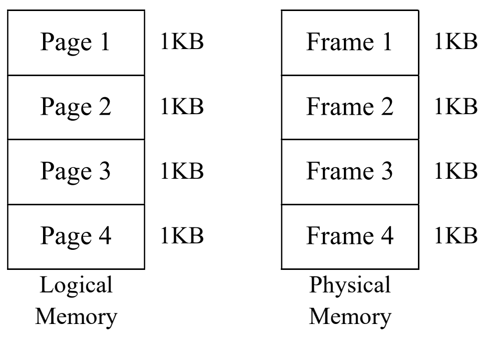


Figure 3.2: A Process with 1KB block size in logical and physical memory.

The paging technique divides the logical memory to blocks of the fixed size known as pages and divides physical memory into blocks of fixed-size known as Frames. Figure 3.2 shows an example of pages and frames in logical and physical memory respectively.

Paging scheme allows a process to be stored in the main memory in noncontiguous manner. It also solves the problem of searching and fitting blocks of different sizes in main memory by having all block of same size. One more advantage of the paging scheme is that it prevents from external fragmentation that is if the main memory

Space for learners:

blocks are of varying sizes and the size of the free blocks are smaller than the size of the pages, then the operating will be required to merge two or more blocks into a single block large enough to fit a page. By keeping block of equal sizes for both pages and frames, such problems are resolved. Figure 3.3shows paging model of physical and logical memory. A page table is used for mapping between logical addresses and physical addresses. A page table resides in the main memory. Figure 3.3 shows noncontiguous allocation of a process in main memory. The mapping of logical address to physical address is achieved using the page table.

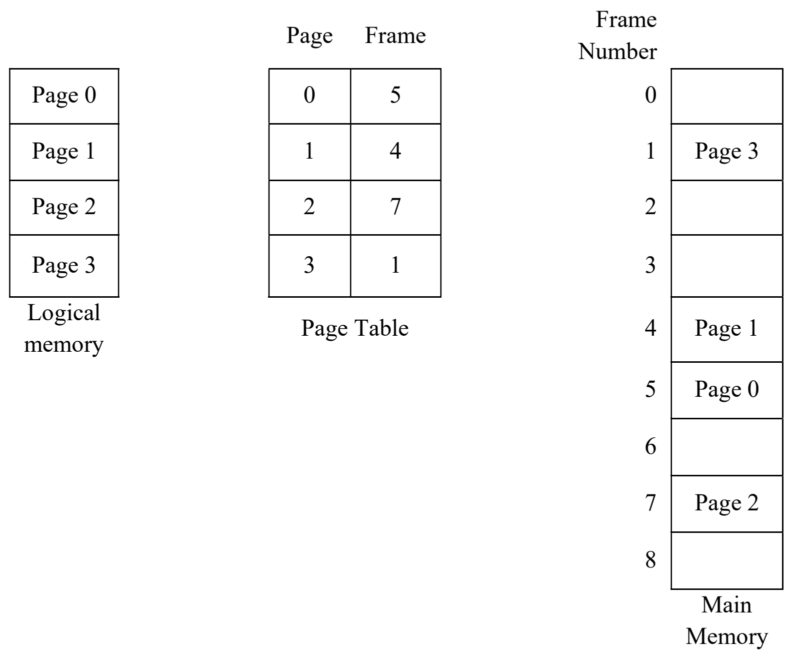


Figure 3.3: Paging model of physical and logical memory.

The hardware support for paging is demonstrated using an example in Figure 3.4. The logical address generated by the CPU is divided into two parts namely *page number* and *displacement* within the page. The page number is used as an index in the page table to search for the corresponding frame number. The displacement is combined with frame number to get the physical address. In the Figure 3.4, the logical address having *page number 3* is searched for the corresponding frame number in the page table which is *frame number 15*. The *frame number 15* is combined with the *displacement 7* to form the physical address.

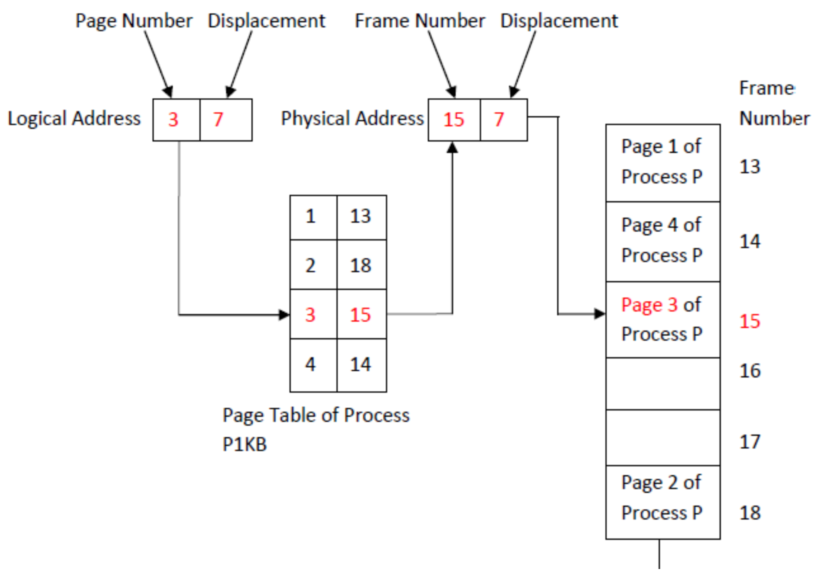


Figure 3.4: Paging hardware support

If the size of the logical address space is 2^m and size of a page is 2^n bytes/words, then "*m-n*" bits of a logical address designate the page number, the "*n*" bits designate the displacement or offset. Therefore, the logical address is:

Page Number	**Displacement**
p	d
m - n	n

Paging Example -1:
Assume a page size of 1K and a 15-bit logical address space. How many pages are in the system?

Solution:
Page size = 1K = 2^{10}i.e. displacement, n=10 bits
No. of bits in logical address = 15, i.e. m=15 bits.
Therefore, no. of bits used for page number is, m - n = 5 bits
Total no. of pages in the system is 2^5 =32.

Paging Example -2:
Assume that a CPU has a 15-bit logical address space with 8 logical pages. How large are the pages?
Solution:
There are 8 logical pages, that means3 bits are required to address 8 logical pages (2^3 = 8).
Therefore, m - n=3 bits
Logical address is 15 bits, m=15 bits
Displacement = 15 -3 = 12 bits.
So, the pages are of size 2^{12} = 4096 = 4K bytes

3.3.1 Paging Hardware Support

Operating system provides support for storing page table of a process. Generally, a page table can be stored in following ways:

- Set of dedicated registers
- In main memory
- Translation lookaside buffer (TLB)

The feasibility of the first approach using a set of dedicated registers is that the page table should be reasonably smaller in size like 256 entries. With the second approach page table can be very large like millions of entries can be stored in the main memory with a pointer to the starting address of the page table for referencing. However, in this case the time required to access the page table is slower by a factor of two as it involves first accessing memory for the page table to locate the frame number which is combined with the displacement to get the physical address and then a second memory access to read the byte.

Space for learners:

The solution to the disadvantages of the first two approaches is resolved using a fast lookup hardware support called Translation Lookaside Buffer (TLB). TLB is a small, expensive but very fast associative memory.

It can store entries in the range of 64 to 1024. Associative memory has two parts: a tag and a value. When a page/key needs to be searched, the key is compared simultaneously with all the tags of the associative memory.

There are possibly two cases for a page search in TLB. Figure 3.5 illustrates the paging hardware with TLB for these two cases:

- If the search key/page is found, it is called as a TLB hit and corresponding value/frame is returned from the TLB. Displacement is combined with frame number and the physical address is accessed.
- If the search key/page is not found, it is called as a TLB miss and the page is searched in the page table stored in main memory. The frame number corresponding to the search page is combined with the displacement to access the address in the physical memory. Also the page number and frame number is added to the TLB so that if the same page is referred next time it is found quickly. In case the TLB is full, operating system selects a page replacement algorithm to replace an existing page with the new entry.

The percentage of times that a particular page number is found in the TLB is called the *hit ratio*. If the hit ratio is 60% that means 60 times out of 100 references, the page will be found in TLB and remaining 40 times the page is found in the page table.

Space for learners:

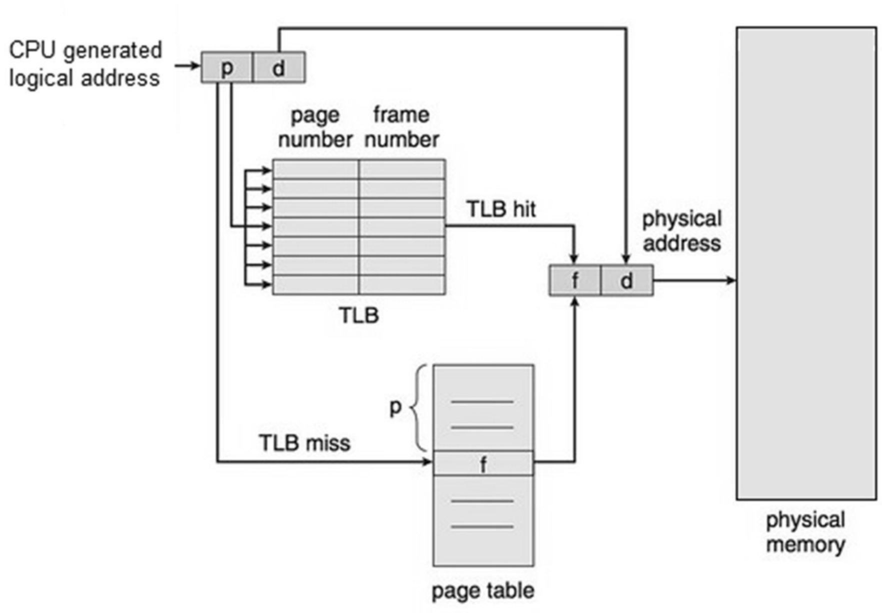


Figure 3.5: Paging hardware with Translation Lookaside Buffer [1].

Space for learners:

Paging Example -3:

If it takes 25 nanoseconds to search the TLB and 75 nanoseconds to access memory. If the hit ratio is 70%, calculate effective memory access time.

Solution:

If the page is in the TLB, time taken to access the physical address

= Time taken to search the TLB + Time taken to access memory

= 25 +75 =100 nanoseconds

If the page is not in the TLB, time taken to access the physical address

= Time taken to search the TLB + Time taken to access page table

+ Time taken to access memory

= 25 +75 +75

= 175 nanoseconds

Hit ratio is 70%, therefore

Effective access time = 0.70 X 100 + 0.30 X 175 =122.5 nanoseconds.

3.4 SEGMENTATION

Segmentation is a memory management scheme similar to paging that allows a process to be stored in the main memory in noncontiguous manner. Unlike paging where all the pages or frames are of fixed size, segmentation allows blocks or segments of variable size. Segmentation maps the user's view of a program onto the physical memory. Looking at the user's view in Figure 3.6, a program contains several variable size segments, such as the main program, subroutine, symbol table, methods etc. It also includes data structures like arrays, objects, variables, stacks etc. These segments and data structures are referred by their name without concerning about the address these segments are stored in memory. Users are not concerned about the order in which the segments are stored in the memory.

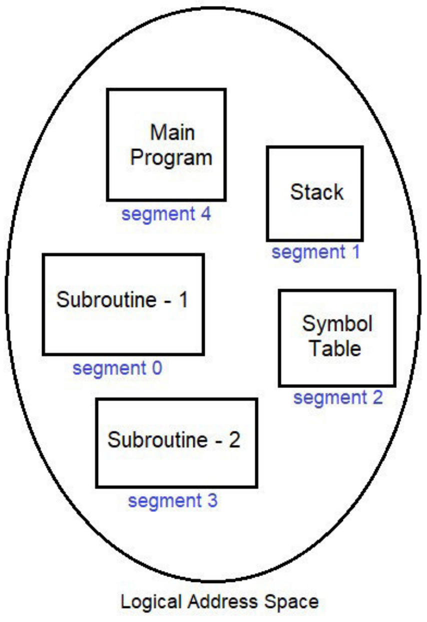


Figure 3.6: User's view of a program

The logical address space is a group of segments. Each segment has a name and a length. From the implementation point of view, segments are numbered instead of using name and the logical address is represented using the *two tuple:*

Segment-number	Displacement

3.4.1 Segmentation Hardware

The mapping of the logical address <segment-number, displacement>to the physical address is achieved with the help of segment table and the segmentation hardware as shown in Figure 3.7. Each entry of the segment table has a segment limit and segment base. The base represents the starting address of the segment in the main memory and the limit specifies the length of the segment. The segment table is indexed on the segment number.

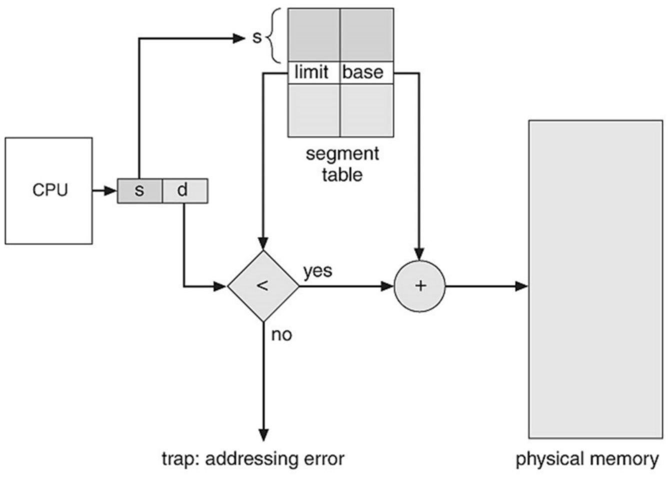


Figure 3.7: Segmentation Hardware[1].

The working of segmentation hardware starts by first identifying the segment number, *s* and the displacement, *d*o f the logical address. The segment number is used to search the segment table, which is indexed on the segment number. The displacement, *d* of the logical address should be between 0 and limit. If the condition is not satisfied, it means that the logical address is going beyond the segment limit and a trap interrupt is initiated which is handled by the operating system.

Space for learners:

Space for learners:

A segmentation example is shown in Figure 3.8. There are 5 segments numbered from 0 through 4. The segments are stored in physical memory in noncontiguous manner. Also, no specific ordering is followed for storing the segments as can be observed in the example. The segment table has an entry for each of the segment, the starting address of the segment mentioned as *base* and the length of the segment mentioned as *limit*. For example, segment 0 begins at address 5100 and length of the segment is limited to 500 bytes. Therefore, a reference to byte 17 of segment 0 is mapped to 5100 (base of segment 0) + 17 = 5117. Similarly, a reference to byte 88 of segment 4 is mapped to 7300 + 88 = 7388.A trap interrupt will be called if byte 1700 of segment 4 is referenced as the limit is 1500.

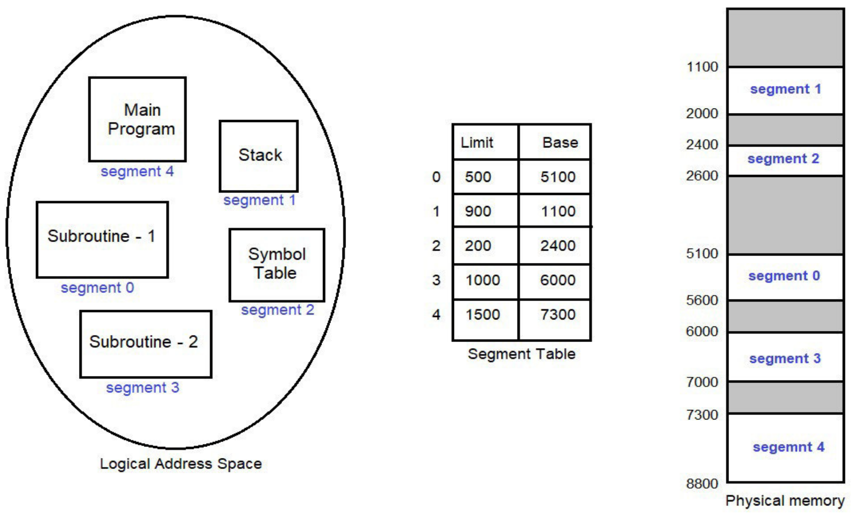


Figure 3.8: Segmentation Example

3.5 VIRTUAL MEMORY

The memory management scheme discussed in previous section requires the entire process to be in the main memory for execution. Most of the times there can be a requirement of many processes to be in the memory simultaneously for execution. This situation can prevent simultaneous execution of multiple processes due to the size of the main memory, which may not be large enough to hold all the processes. So the concept of virtual memory was introduced.

A virtual memory management scheme allows execution of a process even if it is not completely in memory. That is, it requires only that section of the code of the process to be in the memory that will be executed. Generally, a process contains several functions or procedures and not all the functions are required to be in the memory at the same time. So the function or the procedure that will be executed needs to be in the main memory while the other functions or procedures can be placed in the secondary memory and wait for their turn of execution. So whenever a function is not available in the main memory, it is brought from the secondary memory to main memory for execution. The main advantage of this scheme is that a program larger than main memory can still run on a smaller physical memory. This is how a games like *Need for speed* or *Call of Duty* which require respectively 30 GB and 90 GB of memory can still run on a system having 6 GB RAM with sufficient hard disk space. Also, as only a section of the process's code needs to be in memory so many process can be there in memory simultaneously. Thereby increasing CPU utilization and throughput.

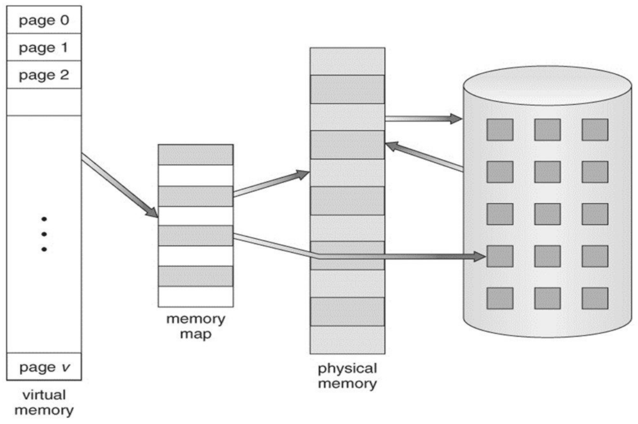


Figure 3.9: Example showing virtual memory larger than physical memory[1].

Space for learners:

Figure 3.9 shows an example of a larger virtual memory than physical memory. The programmer thus need not have to worry about the size of the main memory available, thus can concentrate on the problem to be programmed. As can be seen in the Figure 3.9, pages from the large virtual memory address space is stored in the secondary memory and the pages are brought back to main memory whenever a call to those pages are required. If the main memory does not have any free slot for the pages, then some page replacement algorithms are used to replace the pages in main memory with the pages from secondary memory.

Space for learners:

3.5.1 Demand Paging

Suppose a user wants to run a program, so the entire program is loaded to main memory from the secondary memory. However, if the program runs only one option/case out of the several cases based on the user input, it is impractical to load the code for all the cases, other cases may never be called for execution. So, a virtual memory technique known as demand paging is used to load only those pages of the process when they are required or whenever there is a demand for the page occurs during the program execution.

In Figure 3.10 shows an example of demand paging where pages 4, 5, 6 and 7 of Program A is swapped out of memory and pages 17, 18 and 19 of Program B is moved in to the memory because of the demand for the pages 17, 18 and 19. The method is implemented by a *pager* program responsible for demand paging.

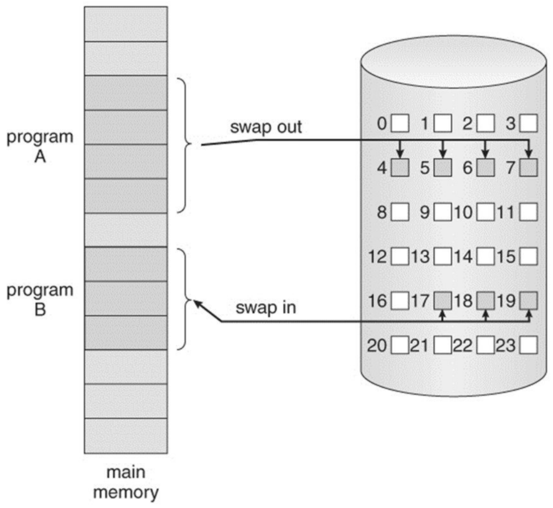


Figure 3.10: Example showing Demand Paging [1].

Space for learners:

CHECK YOUR PROGRESS

i. Fixed-sized blocks in physical memory is called ________
   a) Block
   b) Frame
   c) Pages
   d) Segment

ii. In paging CPU generated logical address has two parts _____________and _____________.
   a) Page offset & Frame bit
   b) Page number & Page offset
   c) Frame offset & Displacement
   d) Frame number& page offset

iii. Fixed-sized blocks in logical memory is called ________
   a) Block
   b) Frame
   c) Pages
   d) Segment

iv. Paging does not suffer from ________.
   a) Internal Fragmentation
   b) External Fragmentation

c) Both a) and b)
d) None of the above

v. If it takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is _________.
a) 120
b) 122
c) 134
d) 124

vi. The displacement 'd' in a logical address must be _____________
a) Greater than segment limit
b) Greater than the segment number
c) Between 0 and the segment number
d) Between 0 and segment limit

vii. In segmentation, each address is specified by _____________
a) A key and value
b) A displacement and value
c) A segment number & displacement
d) A value and segment number

viii. A CPU generated memory larger than main memory is called as
a) Logical Memory
b) Secondary Memory
c) Virtual Memory
d) All of the above

ix. The virtual memory manager loads only those component of a program during execution as a when required is known as _____.
a) Segmentation
b) Swapping
c) Virtual memory
d) Demand Paging

Space for learners:

x. Virtual memory can be implemented with
   a) Swapping
   b) Paging
   c) Segmentation
   d) Both b) and c)

Space for learners:

3.6 SUMMING UP

- Logical address is the address that is generated by the CPU for a running program.
- A physical address is the actual address in the main memory.
- Paging is a memory management scheme that is used to map CPU generated logical address of a process to physical address in main memory.
- The logical address generated by the CPU is divided into two parts namely page number and displacement with the page.
- Translation Lookaside Buffer is a small, expensive but very fast associative memory.
- In a TLB, if the search page is found it is called as a TLB hit if the page is not found it called as TLB miss.
- The percentage of times that a particular page number is found in the TLB is called the hit ratio.
- Segmentation is a memory management scheme similar to paging that allows a process to be stored in the main memory in noncontiguous manner.
- The mapping of the logical address <segment-number, displacement> to the physical address is achieved with the help of segment table and the segmentation hardware.
- A virtual memory management scheme allows execution of a process even if it is not completely in memory.

- A virtual memory technique known as demand paging is used to load only those pages of the process when they are required or whenever there is a demand for the page occurs during the program execution.

3.7 ANSWERS TO CHECK YOUR PROGRESS

i, b	ii, b	iii, c	iv, b	v, b
vi, d	vii, c	viii, c	ix, d	x, d

3.8 POSSIBLE QUESTIONS

1. Differentiate between physical and logical address space.
2. Explain paging memory management scheme.
3. Define a page table. Why it is needed in paging?
4. What is hit ratio? Why page should be replaced in the memory?
5. Explain the working of a paging memory management scheme.
6. Consider a logical address space of 16 pages of 512 words each, mapped on to a physical memory of 64 frames. How many bits are there in the logical address? How many bits are there in the physical address?
7. If it takes 125 nanoseconds to search the TLB and 500 nanoseconds to access memory. If the hit ratio is 90%, calculate effective memory access time.
8. Assume a page size of 4K and an 18-bit logical address space. How many pages are in the system?
9. Assume that a CPU has a 16-bit logical address space with 4 logical pages. How large are the pages?
10. What is segmentation? Explain.
11. Define a virtual memory. With a neat diagram, explain the working of a virtual memory. What are the benefits of a virtual memory?
12. What is demand paging? Explain.

Space for learners:

13. What is the benefit of demand paging?

14. Consider logical address 1025 and the following page table for some process P0. Assume a 15-bit address space with a page size of 1K. What is the physical address to which logical address 1025 will be mapped?

6
2
3

15. Consider the following segment table:

Segment	Base	Length
34	100	100
21	2500	200
0	1200	50
90	1700	300
7	500	500
2	600	50
99	650	200

What are the physical address for the following logical address?

i. 0,25
ii. 2,89
iii. 90,345
iv. 34,50
v. 99,201

3.9 REFERENCES AND SUGGESTED READINGS

- Computer Organization and Architecture, 10th edition, William Stallings, Pearson.
- Computer System Architecture Third Edition, M. Morris Mano, Rajib Mall, Pearson

Space for learners:

- Computer Organization, 5th Edition, Carl Hamacher, McGraw Hill
- Operating System Principles 8th edition by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, Willey

Space for learners:

UNIT 4: BASIC I/O SYSTEM-I

Space for learners:

Unit Structure:

4.1 INTRODUCTION

Input and Output (I/O) devices are integral parts of computer systems. I/O devices and I/O modules are the functional units of a computer along with the Central Processing Unit (CPU) and the memory units. There exists a wide variety of I/O devices having different characteristics. Thus I/O devices are not directly connected to the CPU; rather they are connected via I/O modules. I/O modules take the responsibility of establishing the communication between the CPU and I/O devices by bridging the gap between an I/O device and the CPU. Each I/O module connects with the system bus or to the central switch. An I/O module can control more than one device.

This unit will provide an understanding of basics of I/O interfacing. We begin this chapter with an overview of bus interconnection and bus arbitration, and then we illustrate the functioning of I/O operations via I/O module. At the end of the unit we present the basics of interrupts and direct memory access (DMA).

Space for learners:

4.2 UNIT OBJECTIVES

On completion of this unit students will be able to:

- Explain the basics of bus structures, bus arbitration and roles of different buses.
- Get familiarized with various input output devices.
- Comprehend various aspects of input output interfacing.
- Learn the functioning of input output modules
- Understand the significance of interrupts in communication within put output.
- Learn the concept of data transfer using direct memory access

4.3 BUS INTERCONNECTION

A bus is a pathway via which two or more devices can perform data transfer. Buses are shared transmission media; multiple units can use the same bus for the data transfer but at a time only one unit can send data. A bus can be used to connect either the major components of a computer or the internal components of a CPU or two different computers.

Typically a bus is comprised of multiple lines. Each line can transmit a single bit (0 or 1); thus it can transfer a group of bits in parallel in a single transfer .The number of bits that can be transferred in parallel is called as the bus width. For example, an 8-bit wide bus can transmit 8 bits at a time.

Computer systems have different types of buses for different levels of communications. The internal components of a CPU are connected via internal CPU bus. The major components of a computer system, i.e., the CPU, memory modules and I/O are connected via a special type of bus called as *system bus*.

STOP TO CONSIDER

Buses are used by different modules of a computer to transfer data to other modules. Via buses various forms of data are transferred.

4.3.1 Structure of Bus

As mentioned earlier, a system bus is a common bus shared by the CPU, memory and the I/O. A typical system bus comprises of about 50 to hundreds of separate lines. The connected modules can send different types of information such as data, address and control signals over these lines. Thus lines are usually divided into three groups: data, address and control lines. The schematic diagram of a typical system bus structure is shown in Fig 4.1.

The **data lines** or **data bus** is used to transfer the data among the components attached to it. The width of data bus of a contemporary machine, can be 32, 64 or even more. This width determines the amount of data that can be transferred at a time. The width of the data bus is a key parameter to determine the performance of the system. For example, if the length of an instruction is of 64 bit, then the processor would need to access memory only once if the data bus if 64-bit wide; on the other hand if the data bus is of 16-bit, then

Space for learners:

the processor would need to access the memory 4-times to fetch the 64-bit instruction. Thus wider the bus faster will be the data transfer.

The **address lines**, also known as **address bus** identifies the location of the source or the destination of the data available on the data bus. For example, if the processor has to read the data from memory location X, then it places the address X onto the address bus. The width of the address bus determines the system's memory capacity. For an instance, a system with 16-bit address bus can support a memory of 2^{16} blocks. Moreover, the address bus is also used to locate an I/O port. The higher order bits usually identify a particular I/O module and the lower order bits identify the particular port of the selected module.

The **control lines** or **control bus** are used to carry control signals and timing information to the various computer components. Control signals help in enabling a system to understand what has to be done and timing information indicates the validity of the information available in the data and the address bus. Typical control signals are Memory Read, Memory Write, I/O Read, I/O Write, Bus Request, Bus Grant, Interrupt etc.

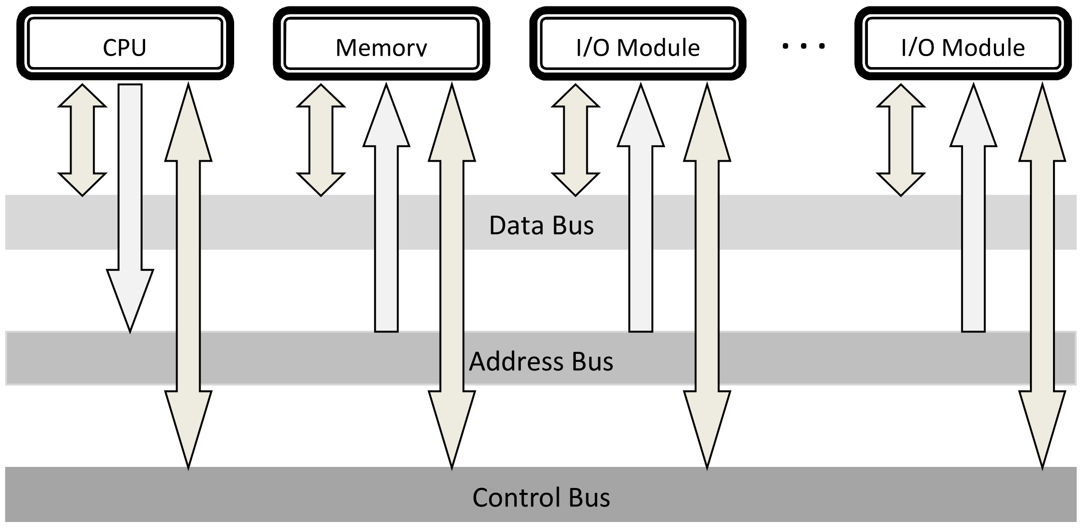


Fig. 4.1 Interconnection of Computer Modules via System Bus

Space for learners:

STOP TO CONSIDER

A common bus structure is used to connect the major components of a computer. Such a structure is called as system bus. System bus allows a computer module to transmit data, address and control signals to another module. Thus the bus lines are grouped into data, address and control lines.

Space for learners:

4.3.2 Aspects of Bus Design

There are a few aspects which are needed to be considered while designing a bus structure. The key aspects are *bus type*, *bus width*, *method of arbitration*, *timing* and *data transfer type*.

Bus Types:

Buses can be categorized into two broad types: dedicated and multiplexed. Dedicated buses are used either for a specific function (e.g., for data or address) or to connect specific physical modules. The advantage of dedicated bus is higher throughput. However, it increases the size as well as the cost of the system.

On the other hand, multiplexed buses are ones either for used multiple functionalities or to be shared amongst multiple physical modules. For example, a common bus can be used to share both data and address information. The main advantage of having multiplexed bus is that it uses of fewer lines which helps in making the system compact as well as cost effective. A major disadvantage of it is that it needed a more complex circuitry for each connecting module.

Bus Width:

We have already addressed the role of the bus width while discussing different bus types. It determines the amount of data that can be transferred at a time. Higher is the width of the data bus, higher is the transfer rate. Thus the width of the data bus has an impact on the performance of a system while width of the address bus determines the system's capacity to address memory blocks.

Bus Arbitration:

In case of a shared bus system more than one module may require to have the control of the buses. Typically, the CPU has the main control of the buses; however when a module wishes to perform the data transfer without CPU's intervention then the device which controls the data transfer may need to have the control of the buses. In such a scenario, the CPU has to transfer the control of the buses to the device managing the data transfer. The process of transferring the control of the buses from one device to another is called as bus arbitration. There are basically two types of bus arbitration methods: *centralized* and *distributed*. In case of centralized arbitration, a special hardware called as *bus arbiter* performs the allocation of the buses to the module requiring the buses. This device can be a part of the CPU or can be a standalone module. In distributed method, the modules mutually share the control of the buses without relying on any centralized arbiter.

Timing:

Timing is a very important criterion of bus design. It defines a way to coordinate the events occurred on the bus. It can be *synchronous* or *asynchronous*.

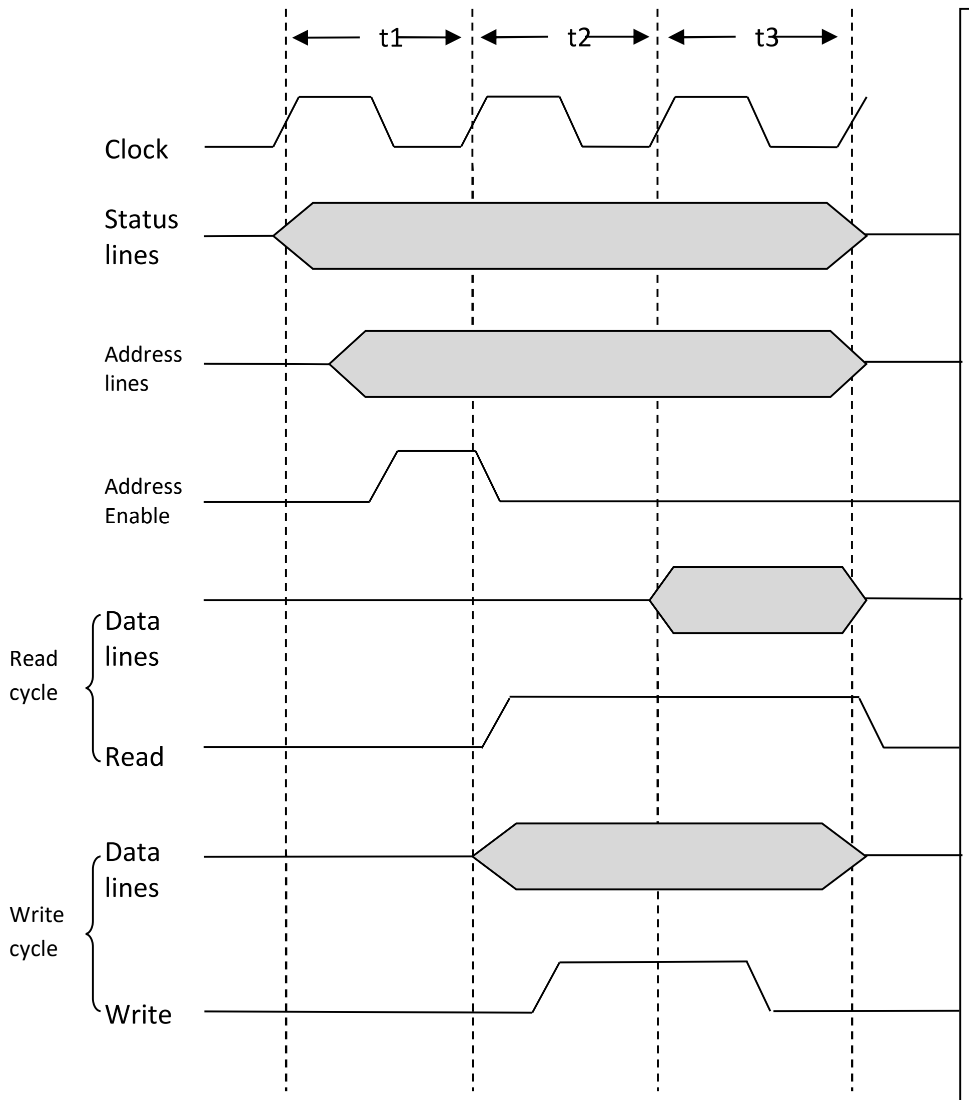
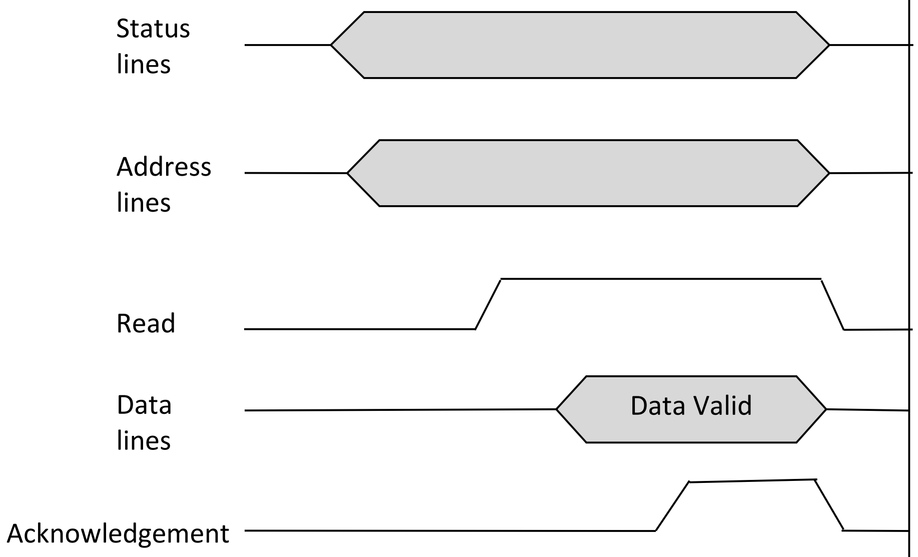


Fig. 4.2 Timing Diagram of Memory Read and Write Cycle

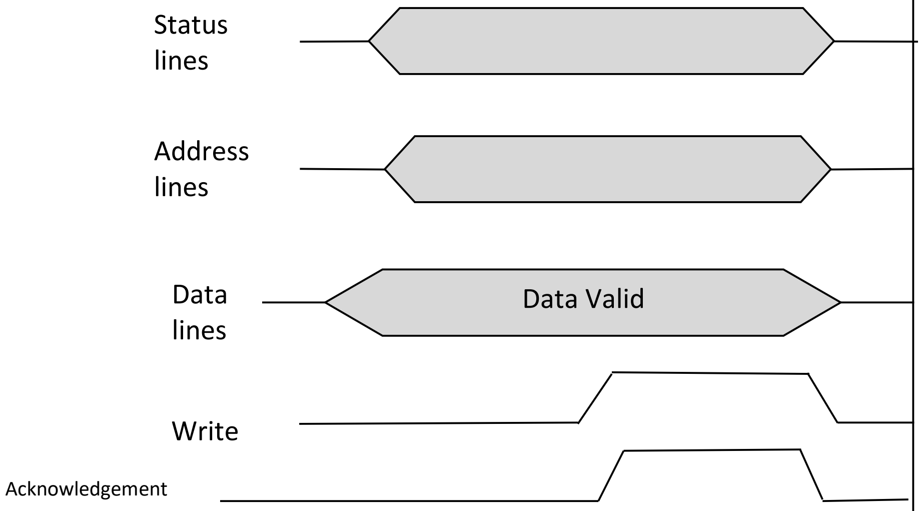
Space for learners:

The occurrences of the events in synchronous timing are controlled by the clock. A clock line is attached to the bus that transmits an alternating sequence of 0s and 1s repetitively. One single transition of 1-0 is termed as one *clock cycle*. A clock cycle defines a slot. All the modules attached to the bus can access the clock line and triggers all events at the beginning of a clock cycle. The Fig. 4.2 presents a sample timing diagram of both memory read and write cycles. In this example, a memory address is placed onto the address bus at the beginning of a clock cycle. Once the entire address is placed onto the bus, the processor asserts the address enable signal. During read cycle, the processor enables the read signal at the beginning of the second clock cycle; the system identifies the address and places the data from the designated memory address

Space for learners:



(a) Memory Read Cycle



(b) Memory Write Cycle

Fig. 4.3 Timing Diagram of Asynchronous Bus Operations

onto the data bus at the start of the third cycle. The processor reads the data from the bus and disables the read signal on completion of the read operation. During the write cycle, the processor places the data onto the data bus followed by activating the write command. The memory reads the data from the bus during the third cycle.

In asynchronous timing no clock is used to coordinate the occurrence of the events, rather the occurrence of one event depends on a previous event. To coordinate the events, the processor asserts special status signals. During read cycle, the processor first places the address onto the address bus and asserts the status signals. The read command is issued once the address is stabilized to indicate the validity of the address. The memory module recognizes the address and copies the data from the corresponding memory address onto the data bus. The memory module confirms the accomplishment of the transfer of data to the bus by asserting the *acknowledgement* signal. The read signal is disabled once the data is read by the processor. The memory module then drops the acknowledgement signal and the processor desserts the read signal. Fig. 4.3(a) demonstrates the sequence of events of the read cycle with asynchronous bus.

During write cycle, the processor places the address, status and data onto the respective buses at the same time. The write signal is asserted by the processor to indicate data valid. The address is recognized by the memory module and fetches the data from the data bus to copy it to the address given. Once write is accomplished, the memory module sends the acknowledgement signal. The write signal is then dropped by the processor or the bus master after receiving the acknowledgement. The write cycle events with asynchronous bus are shown in Fig. 4.3(b).

Space for learners:

STOP TO CONSIDER

To design a bus structure, various criteria like bus type, bus width, type of arbitration and timing are needed to be considered. Based on different parameters chosen for different criteria, the bus has to be designed.

Space for learners:

4.4 I/O Devices

I/O devices are external devices which facilitate the exchange of data between the processor and the external environment. Such devices are also known as *peripheral devices* or simply *peripherals*. An I/O device is connected with the processor via an I/O module port. An I/O device can be used either for input or output or both. Some of the input devices are keyboard, mouse, mic, scanner etc while the output devices include monitor, speaker, printer etc.

I/O devices can be classified broadly into *human readable*, *machine readable* and *communication*. Human readable devices used to allow the users to interact with the computer. These enable the user either to give input or to see the output. Keyboard, monitor and printer are some examples of human readable I/O devices. The machine readable I/O devices are used to establish the communications between various devices or components of the computer. The magnetic disks, tapes, sensors and actuators are some examples of machine readable I/O devices. The communication devices are used to transmit data to a remote device. Examples of communication devices include modems, Infrared, Bluetooth and network interface card (NIC). The remote devices can be a human readable device like a terminal or can be a machine readable device or can even be another computer. Fig. 4.4 demonstrates the generic block diagram of I/O device. The control logic performs the controlling of overall operations of the I/O device. It decodes the task to be performed by the device based on the received control signal. It is also responsible for error detection and status reporting to the I/O module. The transducer's job is to convert the data received from the external environment to the format understandable by the device during input operation and converts the data from device understandable to the format which the external environment understands. The data buffers store data temporarily to be exchanged between the external environment and the I/O module.

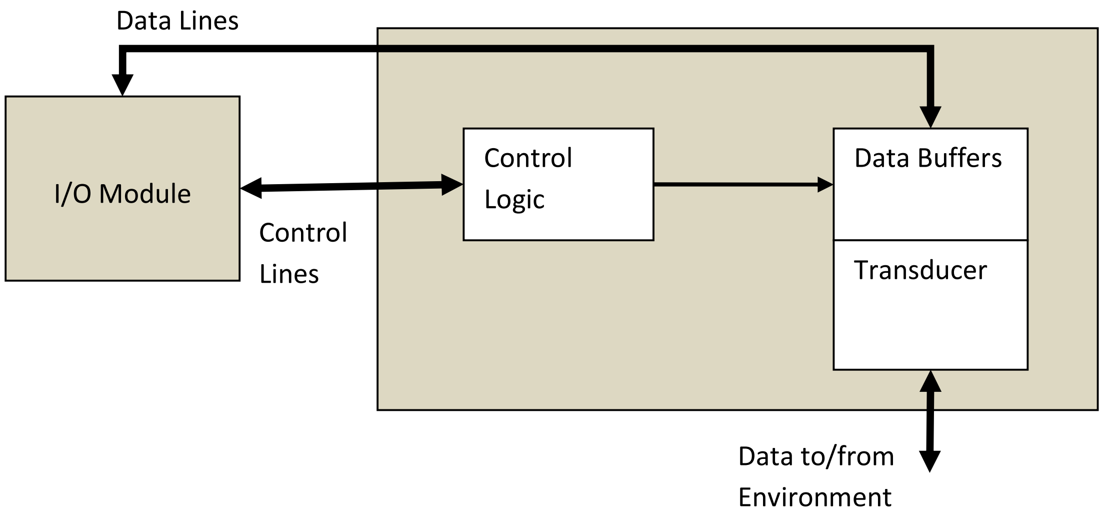


Fig. 4.4 Generic model of an I/O device

The most common I/O devices that almost every computer possesses are keyboard, mouse, monitor and Disk drives. A brief discussion on these four is presented below.

Keyboard

This is the universal input device for all computers. The keyboard layout is identical to that of a standard QWERTY typewriter. It also has several additional command and function keys. It has between 101 and 104 keys in total. Through this, a user can enter alphabets, numbers and symbols called as characters. Each character is associated with a unique 7 or 8 bit code. One of such code representation is *American Standard Code for Information Interchange* (ASCII). To enter data, you must press the precise combination of keys. The transducer in the keyboard interprets the electrical impulses generated by a keystroke and converts it into its corresponding 7 or 8 bit binary code.

Space for learners:

Mouse

Another input device which is used most commonly is the mouse. It has two or three buttons on the top and rolls on a little ball. Different buttons are used to perform different actions. The screen cursors of the mouse move in the direction of mouse movement when you roll it across a flat surface. With the mouse, the cursor moves quite quickly, providing you more freedom to operate in any direction. Moving using a mouse is easier and faster.

Monitor

It is the most common output device that is common in all the computers. It is a unit that displays the characters entered through the keyboard and to display any message. The message can be in the form of text, image or video. So monitors are also called as video display devices. In market various types of video display devices are available. In earlier time Cathode Ray Tubes (CRT) were used to design the monitors. Such monitors were either monochromatic or colored. Although, CRT monitors are still present, however Liquid Crystal Display (LCD) based monitors are more common in recent time. These monitors have a flat panel display and consume less power than the CRT monitors.

Disk Drive

It is a device used for data storage in the computer. It has mechanisms to exchange data and control signals with an I/O module. An I/O module can perform both read and write operations on the disk drive. The transducer on a fixed-head disk can transform magnetic patterns on the moving disk surface to bits in the device's buffer. The disk arm of a moving-head disk must be able to move radially in and out across the disk's surface.

STOP TO CONSIDER

I/O devices are external devices which enable exchange of data between external environment and the computer. There exists a variety of I/O devices for performing various tasks. Keyboard, mouse, monitor and magnetic disks are the most common I/O devices.

4.5 I/O Interfacing using I/O Modules

Space for learners:

A computer is connected with a diverse set of I/O devices. The devices differ largely in terms of data rates, data representations, data formats, word lengths and error conditions. The data rates of the devices differ from the main memory and the processor. Often peripheral devices are slower than the processor and the memory. But there are some devices faster than the memory and the processor. So there is a big gap between the processor and any I/O. In such a scenario direct communication between an I/O device and the processor is not easy. To solve this, I/O modules are used as a mediator between an I/O device and the processor. I/O modules interface to the memory and the processor through the system bus, which interface one or more I/O devices by the ports.

4.5.1 Functions of an I/O Module

As mentioned earlier I/O devices are connected with the processor via the I/O modules. For this, an I/O module needs to interact with both the processor and the I/O devices. The processor initiates the I/O operations and selects the I/O module that connects the target peripheral. The I/O devices send or receive the data to the I/O module to be sent to the processor. The major tasks performed by the I/O module are as follows:

- Control and timing
- Communication between the device and the processor
- Data buffering
- Error Checking

Control and Timing

The processor may need to interact with multiple peripheral devices, memory and buses as per the requirement of the program leading to multiple data transfer among various units. So there must be a proper coordination and sequencing of events in order to avoid any conflict. The events generated by a peripheral device are monitored and synchronized by the connected I/O module. The I/O module controls the activities of the peripheral based on the signals received from the processor.

Communication between the device and the processor

Space for learners:

During an I/O transfer, the I/O module performs four major tasks, namely *command decoding*, *status reporting*, *data exchange* and *address recognition*.

***Command decoding*:** The processor sends commands to the I/O module in the form of control signal. The I/O module decodes the command and instructs the I/O device to perform the necessary task.

***Status reporting*:** As there is a speed mismatch between an I/O device and the processor, it is necessary for the processor to know the current status of the I/O device before and during any data transfer. The processor requests the I/O module to check status of the I/O device. Typical status signals include ready and busy. The I/O module reports back the status of the I/O device to the processor.

***Data Exchange*:** When the I/O device is ready to send or receive the data, the processor requests the I/O module to initiate the transfer. In case of input operation, the I/O module gets the data from the I/O device and forwards the same to the processor. And for output operation, the I/O module gets the data from the processor and then forwards them to the I/O device.

***Address Recognition*:** To uniquely identify the I/O devices, each device is assigned a unique address. During I/O transfer, the processor refers the I/O devices using their unique address or identifier. The I/O module recognizes the specific I/O device it controlling based on the address received from the processor.

Data Buffering

The data buffering is an essential task that the I/O module has to perform as the data rates of processor or memory is much higher than most of the peripherals. The I/O devices cannot receive the data at the speed of memory or processor. The I/O module buffers data received from memory or processor till the I/O device gets ready to receive the data. Similarly, if the data rates of I/O devices are faster than the memory or the processor, the I/O module buffers data received from I/O device to match the speed of processor and memory.

Space for learners:

Error Checking

Errors are inevitable while transferring data over any medium. The error may be mechanical or electrical due to technical malfunctions of the devices or may due to transmission. The transmission errors alter the sequence of bit-pattern of the data. The I/O module includes error detecting codes to detect any transmission error. The module checks for error for each every data it receives.

4.5.2 Structure of I/O Module

The general structure of an I/O module is presented in Fig. 4.5. It contains a register set for storing data, status and control information. The data registers are used to store the buffered data. The status registers stores the current status information. The control information received from the processor is stored in the control registers. The register set is connected with the processor via the data bus. The processor uses the address lines and the control lines to send the address information and command to the I/O modules respectively. The control logic unit recognizes an I/O device based on the address information received via the address lines. It decodes the command received via the control lines. It also has logic to interface with the I/O devices.

STOP TO CONSIDER

Direct exchange of data between the CPU and I/O devices are difficult due to the difference in data transfer rates, data representation and unit of transfer. I/O modules are thus used a third party to establish the communication between CPU and I/O.

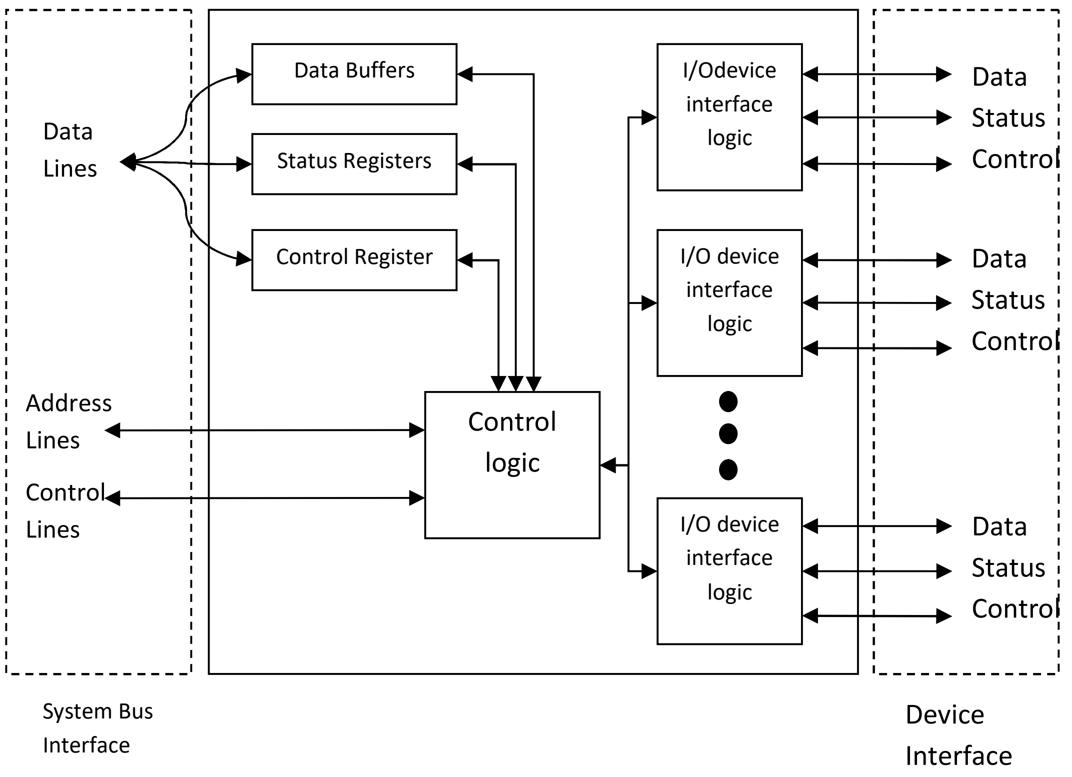


Fig. 4.5 Block Diagram of an I/O Module

4.6 I/O ADDRESSING

The I/O devices are given unique identifiers using any of two addressing modes: *memory mapped* I/O and *isolated* I/O. In *memory mapped* I/O, the I/O devices and memory locations share the same address space. For example, if a system has a 12-bit address bus supporting 4096 unique addresses, then these addresses will be shared among the memory locations and the I/O devices. That means if there is a memory address X, then the address X cannot be assigned to an I/O device. The processor treats I/O transfers exactly same as the memory transfer. Thus, only a single pair of read write lines is required for both memory read/write and I/O read/write. The processor uses the same instructions to access both memory and I/O. The advantage of memory mapped I/O is a large number of instructions are available for I/O operations. However it limits the address space for both memory and I/O.

In *isolated* I/O, memory locations and I/O devices do not share the same address space. If there is a memory address X, then there can be an I/O device with address X as memory locations and I/O have different address space. Thus, a full range of address space is available for both I/O and memory locations. The processor uses different instructions for memory transfer and I/O transfer. It uses separate lines for memory read and I/O read and same holds true for I/O write and memory write. When the memory read/write line is high then the address in the address bus is treated as a memory address and when the I/O read/write line is high then the address in address bus is treated as an I/O address.

Space for learners:

4.7 INTERRUPTS

In computer system, an interrupt is a signal generated by hardware to request the processor to give immediate service suspending the current executions. Hardware interrupts are generally used for handing I/O transfers. As most of the I/O devices are slower than the processor and the memory, the processor does not wait for the I/O to transfer the data. When the I/O is preparing to send or receive data, the processor remains busy with other execution. The I/O device sends interrupt signal to the processor via the I/O module when it gets ready to send or receive data.

For each interrupt, the processor has a routine called as *interrupt service routine* (ISR). This is a special routine that has the code to accomplish the task requested via the interrupt. The processor executes the ISR as a response to the interrupt suspending the current execution. After giving the service to the interrupt, the processor resumes it suspended work.

Apart from hardware interrupts, there are interrupts raised by softwares. These are basically exceptions occurred during the execution of a program. Divide by zero, not a number (NaN), over flow and underflow are some examples of software interrupts.

Space for learners:

4.7.1 Types of Interrupts

A computer system supports a variety of hardware interrupts. These can be broadly classified into two categories: *maskable* and *non-maskable*. Maskable interrupts are the ones that can be ignored. There is a facility to disable such interrupts. These interrupts can be ignored only if they are disabled. The non-maskable interrupts are the highest priority interrupts and cannot be ignored at any cost. Thus, there no option is available to disable such interrupts. TRAP is the example of a non-maskable interrupt.

4.7.2 Interrupt Latency

When the processor suspends the current execution in order to provide the service to interrupt request, it saves the necessary data including the program return address to resume the program execution. The program return address is usually saved onto the processor's stack memory. After saving these data, the program counter is updated by assigning the routine address. This causes a time delay to start the execution of ISR from the time interrupt request has been received. This delay is called as interrupt latency.

STOP TO CONSIDER

When a process or event requires immediate attention, hardware or software emits an interrupt signal.

4.8 DIRECT MEMORY ACCESS

Space for learners:

DMA is a feature of computer systems that allows certain hardware subsystems to access primary system memory (random-access memory) without the intervention of the CPU.

When employing programmed I/O or interrupt driven I/O, without DMA the CPU is often totally engaged for the duration of the read or write operation, leaving it unavailable to execute other tasks. The CPU initiates the transfer via DMA, then does other tasks while the transfer is ongoing, and ultimately receives an interrupt from the DMA controller when the operation is completed.

When the CPU can't keep up with the rate of data transfer, or when the CPU needs to do work while waiting for a relatively slow I/O data transfer, this capability comes in handy. DMA is used by many hardware systems, including disk controllers, graphics cards, network interface cards, and sound devices. In multi-core CPUs, DMA is also employed for intra-chip data transfer. DMA channels allow computers to transport data to and from devices with significantly less CPU overhead than computers without them. A processing element inside a multi-core processor can also transmit data to and from its local memory without consuming processor time, permitting processing and data transfer to happen in parallel.

STOP TO CONSIDER

DMA is technique used to perform data transfer without actively involving the CPU. During the DMA transfer the CPU remains free and can perform some other operations which do not require the system bus.

CHECK YOUR PROGRESS

i. The key advantage of adopting a single bus structure is that it ______

- **a.** faster transfer
- **b.** ease of access
- **c.** cost effective
- **d.** none of the above

ii. System bus is used to transmit

- **a.** data
- **b.** address
- **c.** control signal
- **d.** all of the above

iii. Width of ______ bus determines the performance of the overall system.

- **a.** data
- **b.** address
- **c.** control signal
- **d.** all of the above

iv. Width of the address bus determines___________

- **a.** the performance of the system
- **b.** system's memory capacity
- **c.** both a and b
- **d.** none of the above

v. Usual bus structure used to connect I/O devices follows

- **a.** single bus structure
- **b.** multiple bus structure
- **c.** star bus structure
- **d.** none of the above

vi. I/O modules are used to overcome difference in ________ between I/O and CPU.

- **a.** speed of data transfer
- **b.** data representation
- **c.** units of data transfer
- **d.** all of the above

vii. Memory mapped I/O has the following advantage over Isolated I/O

- **a.** fewer address lines
- **b.** more instructions for I/O operations
- **c.** bigger buffer space
- **d.** all of the above

Space for learners:

viii. Isolated I/O has the following advantage over Memory mapped I/O

- **a.** fewer address lines
- **b.** more instructions for I/O operations
- **c.** bigger buffer space
- **d.** all of the above

ix. What is the mechanism for synchronizing the CPU with the I/O device in which the device sends a signal when it is ready?

- **a.** DMA
- **b.** interrupt
- **c.** signal handling
- **d.** exception

x. DMA transfer has the following advantage

- **a.** faster data transfer
- **b.** increased CPU throughput
- **c.** both a and b
- **d.** none of the above

Space for learners:

4.9 SUMMING UP

- A bus is a pathway via which two or more devices can perform data transfer. Buses are shared transmission media; multiple units can use the same bus for the data transfer but at a time only one unit can send data.
- The major components of a computer system, i.e., the CPU, memory modules and I/O are connected via a special type of bus called as *system bus*. The system bus has three groups of lines for data, address and control.
- The key aspects of bus design are *bus type*, *bus width*, *method of arbitration*, *timing* and *data transfer type*.
- I/O devices are external devices which facilitate exchange of data between the processor and the external environment. Such devices are also known as a *peripheral device* or simply a *peripheral*. An I/O device is connected with the processor via an I/O module port.
- I/O devices are not directly connected to the CPU; rather they are connected via I/O modules. I/O modules take the

responsibility of establishing the communication between the CPU and I/O devices by bridging the gap between an I/O device and the CPU. Each I/O module connects with the system bus or to the central switch. An I/O module can control more than one device.

- The I/O devices are given unique identifiers using any of two addressing modes: *memory mapped* I/O and *isolated* I/O. In *memory mapped* I/O, the I/O devices and memory locations share the same address space.
- In computer system, an interrupt is a signal generated by hardware to request the processor to give immediate service suspending the current executions.
- DMA is a feature of computer systems that allows certain hardware subsystems to access primary system memory (random-access memory) without the intervention of the CPU.

Space for learners:

4.10 ANSWERS TO CHECK YOUR PROGRESS

i. c	**ii. d**	**iii. a**	**iv. b**	**v. a**
vi. d	**vii. b**	**viii. a**	**ix. b**	**x. c**

4.11 POSSIBLE QUESTIONS

1. What is the role of a computer bus?
2. Differentiate between multiplexed and dedicated bus.
3. What are the various aspects of bus design?
4. Why is it not possible to connect an I/O device directly to a computer?
5. Explain the tasks performed by an I/O module.
6. What are the signals shared by an I/O module?

7. What do you mean by an interrupt in terms of a computer system?
8. What do you mean by DMA? What are the advantages of using DMA?
9. Discuss various types of I/O devices.
10. Differentiate between maskable and non-maskable interrupt.

Space for learners:

4.12 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, ZvonkoVranesic, SafwatZaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.

---×---

UNIT 5: BASIC I/O SYSTEM-II

Unit Structure:

5.1 INTRODUCTION

I/O operations are performed through a large variety of I/O devices. These devices provide a way of interchanging data between the external environment and the computer. Different I/O devices have different data transfer rates, different data formats and different word lengths. These variations make the direct interaction between I/O devices and processor (or memory) very complex. Thus, the processor or the memory does not interact with the I/O devices rather I/O modules are used to establish the interactions between I/O devices and the processor (or memory) as a mediator. For an instance, if the processor wishes to send some data to an I/O device, it sends it to the I/O module which forwards the same to the specific I/O device. The I/O operations are performed using three techniques: programmed I/O, interrupt driven I/O and direct memory access.

This unit begins with a discussion on the three mentioned I/O operation techniques. The unit also presents a discussion on various way of handling multiple interrupts.

5.2 UNIT OBJECTIVES

On completion of this unit students will be able to:

- Explain the various aspects of I/O transfer based on Programmed I/O, Interrupt Driven I/O and DMA Transfer
- Compare Programmed I/O, Interrupt Driven I/O and DMA Transfer
- Explain different ways of handling multiple interrupt requests

5.3 PROGRAMMED I/O

In programmed I/O, the processor exchanges data with the I/O module. The processor allows the I/O module to control the I/O operations directly. The I/O module can read the device status, send, read or write command and transfer data. The processor sends a command to the I/O module and waits for the I/O module to complete the operation.

When the processor sees an instruction associated with I/O, it issues necessary commands to the concerned I/O module. The I/O module then loads the status register with appropriate values. The processor checks the status of the I/O module periodically until the module is ready for the transfer. The data transfer takes place only when the I/O module is ready. Most of the I/O devices have much slower data rates than the processor and the memory. So, the processor may need to wait for a longer amount of time for the I/O to get ready. This is the major disadvantage of programmed I/O as it reduces the throughput of the processor.

The processor issues some commands to the I/O module along with an address referring an I/O module and an I/O device. There are four types of commands: *control*, *test*, *read* and *write*.

Space for learners

The **control** command is used to specify the operation to be performed by the external device. For example, it may send commands like READ SECTOR, WRITE SECTOR, SCAN record ID to a magnetic disk. The commands are made according to the operations an I/O device performs.

Test commands are used to test various status signals of both the I/O module and the I/O devices. Before any I/O transfer, the processor needs to test the current status of the I/O module or the device to check whether the module or the device is powered on, ready or busy. It may also need to know if the last data transfer is successful or any error has occurred.

The **read** signals are sent to the I/O modules when the processor needs data from any I/O. The I/O module gets the data from the particular I/O device and buffers it in its internal storage (data buffers/ data registers) temporarily before sending back to the processor. The I/O module sends back the data to the processor by placing them onto the data bus on receiving request from the processor.

With **write** signal, the processor requests the I/O module to send the data available on the data bus to a specific I/O device. The I/O module obtains the data from the bus and buffers it until the corresponding I/O device is ready to accept the data.

Fig. 5.1 demonstrates the process of transferring blocks of data from memory to I/O using programmed I/O. The processor first fetches a memory word and tests the status of I/O. If the I/O module is ready it transfers the data immediately otherwise it waits. During the waiting period, the processor keeps on sensing the I/O status periodically.

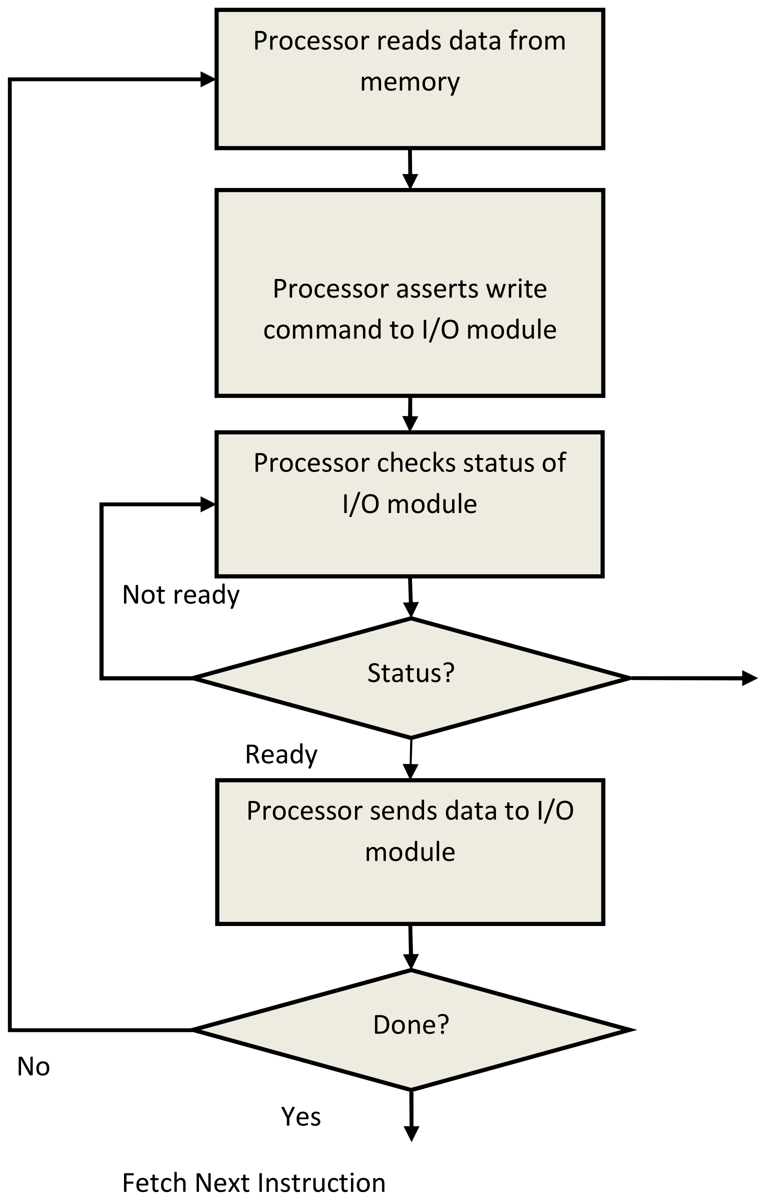

Processor reads data from memory
Processor asserts write command to I/O module
Processor checks status of I/O module
Not ready
Status?
Ready
Processor sends data to I/O module
Done?
No
Yes
Fetch Next Instruction

Fig. 5.1 Flowchart showing transfer of data from the processor to I/O using programmed I/O

STOP TO CONSIDER

Programmed I/O is an I/O transfer technique wherein the processor continuously senses the status of I/O until the later gets ready for data transfer. This reduces the performance of the processor.

Space for learners

5.4 INTERRUPT DRIVEN I/O

Space for learners

The main problem with programmed I/O is that the processor has to wait for long time for the I/O module to be ready. During the waiting time, the processor must check the status of the I/O continuously. This adversely affects the performance of the overall system. Consequently an alternative solution is required to enhance the performance of the entire system. One best solution is the use of interrupt signals. Instead the processor checking repetitively the status of I/O, the I/O module can send interrupt to the processor when it is ready. Such type of I/O transfer is called as interrupt driven I/O.

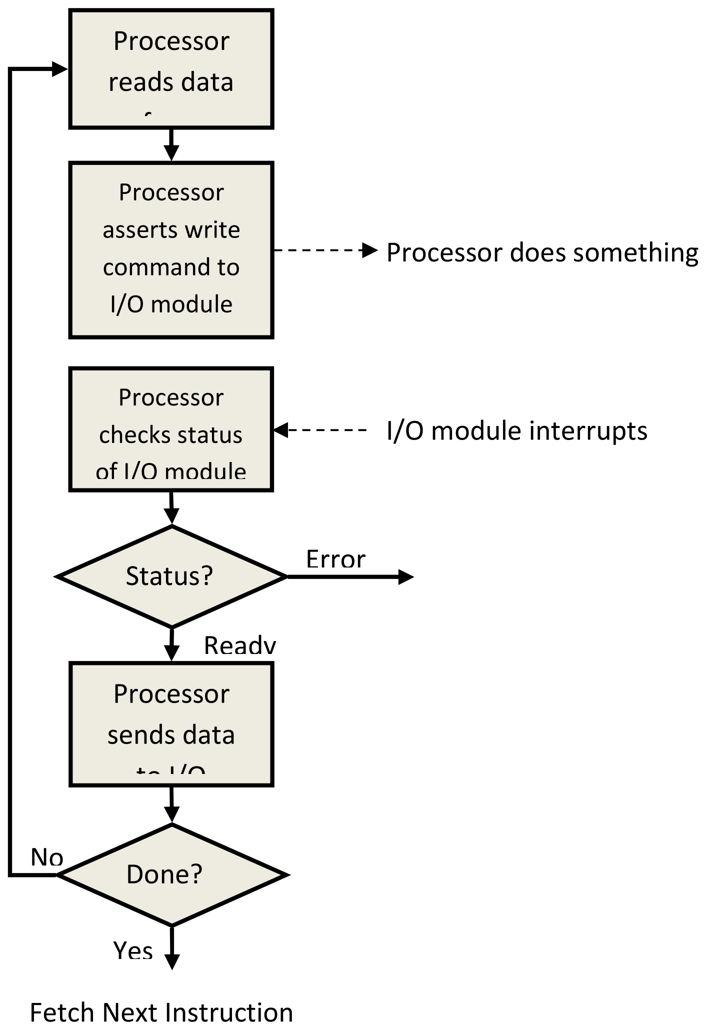


Fig. 5.2 Flowchart showing transfer of data from the processor to I/O using interrupt driven I/O

Fig. 5.2 presents the flowchart of transfer of memory words to I/O using interrupt driven I/O. The processor first reads the data from memory and asserts the write signal to the I/O module to which the concerned I/O device is connected. It specifies the I/O device by placing its address on the address bus. The processor does not wait for the I/O to get ready and continues its execution. In interrupt driven I/O, the processor issues a command to an I/O module and then gets busy in doing other processing. The I/O module will send an interrupt request to the processor when it is available to perform the data transfer. Every interrupt has a specific program or routine called as interrupt service routine (ISR) to process the interrupt request. On receiving the interrupt request, the processor finishes its current instruction and then goes on to give the service to the interrupt request by executing the corresponding ISR. The processor stops the current execution temporarily while executing an ISR. It goes back to its previous program immediately after finishing the ISR.

The I/O module identifies the I/O device based on the address available on the address bus. It checks the status of the corresponding I/O device if it is ready. The I/O device sets the status as ready to inform the I/O module when it is ready to send any data. On receiving this information, the I/O module interrupts the processor. The processor then transfers the data and checks if any data is remaining to transfer. If not, the processor continues with the data transfer as shown in the diagram.

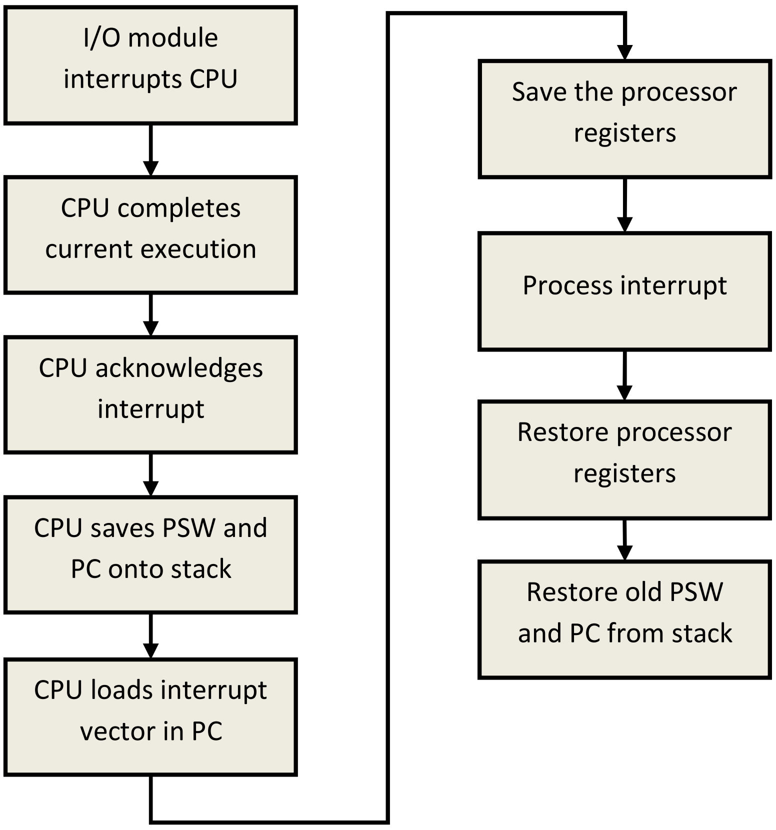

I/O module interrupts CPU
CPU completes current execution
CPU acknowledges interrupt
CPU saves PSW and PC onto stack
CPU loads interrupt vector in PC
Save the processor registers
Process interrupt
Restore processor registers
Restore old PSW and PC from stack

Fig. 5.3 Block Diagram of Interrupt Processing

Fig. 5.3 presents a block diagram of the sequence of events occurred during the processing of a typical interrupt. The following sequence of events occurs during an I/O transfer.

1. The I/O module sends an interrupt signal to the processor.
2. The processor completes current instruction before answering the interrupt.
3. The processor tests for the interrupt at the end of every instruction cycle. When it sees any interrupt, it sends acknowledgement to the I/O module. The I/O module then disables the interrupt signal.

4. The processor prepares to start the execution of the ISR. It saves the program return address (current value of the program counter) and the ALU flags or program status word (PSW) onto the stack.
5. The processor loads the routine address or the interrupt vector onto the program counter (PC).
6. The processor then saves the current status of the executing program, particularly the contents of the ALU registers onto stack. This is very essential as the ISR may need to use these registers.
7. The processor starts processing the interrupt by executing the ISR. At this stage the processor begins its next instruction cycle.
8. After completion of the execution of the ISR, the processor restores ALU registers.
9. Finally it restores PSW and the old value of PC stored from the stack.

Space for learners

STOP TO CONSIDER

Unlike in programmed I/O, in interrupt driven I/O the processor does not continuously check the status of the I/O device. After initiating the I/O transfer the processor gets involved in some important tasks without waiting for the I/O. The I/O module sends an interrupt signal whenever the I/O device is ready for the data transfers.

Design Issues

When it comes to interrupt driven I/O, there are two design challenges to consider. First, how will the processor identify the interrupting device if multiple devices are connected? Second, which interrupt to process if multiple interrupts occur at some time?

To address the first issues, i.e. device identification, four techniques are used in common:

- Multiple interrupt lines
- Software Poll
- Hardware Poll (Daisy Chain)
- Bus Arbitration

The simple stand straightforward solution to handle multiple interrupt is the use of **multiple interrupt lines** for multiple I/O devices. However, it is not a practical solution to have too many lines for interrupts. Typically, interrupt lines not assigned to the I/O devices; instead they are assigned to the I/O modules. This method helps the processor to identify easily the interrupted module. But an I/O module can connect more than one device, so to identify the specific device (the one which triggered the interrupt) from many, one of the remaining three methods can be used.

Instead of using multiple interrupt lines **Software polling** can be used alternatively to handle multiple interrupts. In this, a common ISR is executed when the processor sees an interrupt. The job of this ISR is to detect the interrupted module by polling each module. The polling can be done by using a dedicated command line (TESTI/O). The processor sets the TESTI/O and places the I/O address in the address bus. An I/O module responds to this signal positively if the interrupt is raised by it. Alternatively, each I/O module can possess a status register which will be set when it raises the interrupt signal. The processor will check the status register of each I/O module and will determine the I/O module that caused the interrupt based status information. After identification of the interrupted module, the processor executes the ISR of the interrupted device. The advantage of software polling is that a single interrupt line is sufficient for implementing interrupt driven I/O. However it is very time consuming.

Space for learners

Space for learners

Hardware polling is a very efficient alternative to software polling for handling multiple interrupts. A technique called **Daisy Chain** can be used to implement this. In this approach, a common interrupt request line is shared among all I/O modules. The I/O modules are connected in a serial order. The interrupt acknowledgement line is shared with the I/O modules through a daisy chain as shown in Fig. 5.4. The processor sets the interrupt acknowledgement signal when it sees any interrupt request. This signal is received by the I/O module which is directly connected with the interrupt acknowledgement line. If the interrupt request is raised by that particular I/O module then it will respond by placing a vector in the data lines; otherwise the module will forward the acknowledgement signal to the next module in the sequence. The next module will react to the signal exactly in the similar manner. Thus the interrupt acknowledgment signal will be propagated through the I/O modules until any response is received from the interrupted module. The vector is usually an address that refers an I/O module. The processor calls the device specific ISR based on the value of the vector.

Another alternative is **bus arbitration**. In this approach, only one module can send interrupt request. To do so, the I/O module has to obtain the control of the bus first. The processor responds to the interrupt by sending an interrupt acknowledgement signal. The I/O module responds to this signal by placing its interrupt vector onto the data bus.

To solve the second issue, different levels of priorities can be assigned to different modules. When more than one module interrupts, the modules are given services according to their priority levels. The module with the highest priority is given the service first. The above mentioned techniques can also be used to handle this priority interrupt. When there are multiple interrupt lines, the

Space for learners

processor simply chooses the one with the highest priority. In software polling, the module polling order is designed according to their priority. In case of hardware polling, the modules in the daisy chain are arranged according to their priority with the highest priority first. In case of bus arbitration, the bus arbiter determines which module should get the control of the bus depending on their priority.

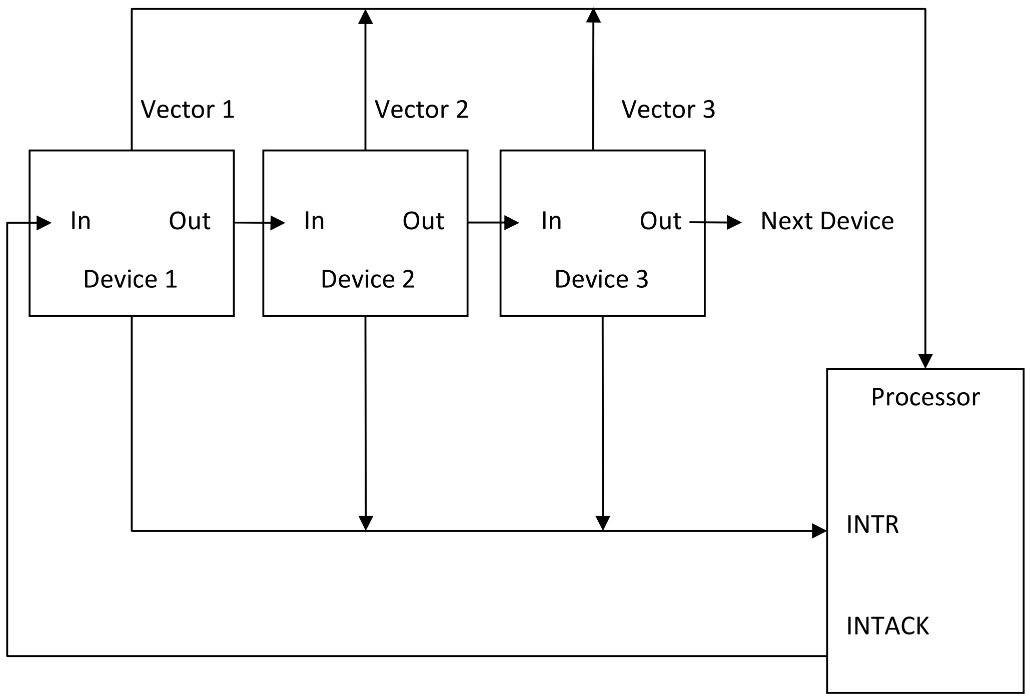


Fig. 5.4 Hardware Polling using Daisy Chaining

5.5 DIRECT MEMORY ACCESS

Both programmed I/O and interrupt driven I/O require the active intervention of the processor to perform the data transfer between memory and I/O. When there is a need to transfer a large amount of data, the processor is often tied up with the I/O transfer. Also, the data transfer speed is affected by lot of testing and condition checking. To avoid these issues, a more efficient technique called

direct memory access (DMA)can be used while transferring a large amount of data.

It is a data transfer technique in which transfer of data from memory to I/O takes place without the active involvement of the processor. To accomplish this, an additional module called a DMA controller is required. A DMA controller shares the system bus along with processor, memory and I/O. Its role is to control the entire data transfer. For this, it has to acquire the control of the system bus. The structure of a typical DMA controller is shown in Fig. 5.5.

When the processor needs to perform DMA transfer, it issues a DMA request to the DMA controller and sends the following information to the DMA controller:

- Depending on the operation type, the processor asserts read or write signal to the DMA controller by raising the corresponding control line between the processor and the DMA controller.
- The address of the target I/O device.
- The address in the memory from/to where data transfer to begin through the data lines. The DMA controller saves this address in its address register.
- The number of words to be transferred. This value is then stored in the data count register.

After initiating the transfer, the processor relinquishes the buses and continues with other works while the DMA controller gains the control of the buses and takes over the remaining transfer. The DMA controller transfers the entire blocks of data one by one. Once the DMA transfer is complete, the controller sends an interrupt to the processor.

There are basically two types of DMA transfers: *burst mode* and *cycle stealing*. Burst mode transfers a whole block of data in a single contiguous sequence. When the processor grants the DMA controller the access to the system bus, it transfers entire bytes of data in the data block before returning control of the system buses to the processor; however this leaves the processor inactive for a long time.

In systems where the processor should not be disabled for the length of time required for burst transfer modes, the cycle stealing mode is used. The DMA controller gains control the system bus in cycle stealing mode in the same way as it does in burst mode, by using the BR (Bus Request) and BG (Bus Grant) signals, which control the interface between the processor and the DMA controller. In cycle stealing mode, however the control of the system bus is delegated to the processor via BG after one byte of data transfer. After a cycle, the DMA controller again obtains the buses using BR and BG signal for the next transfer. This switching of the buses between the processor and the DMA controller continues until entire blocks of data are transferred.

CHECK YOUR PROGRESS

i. _________ is a way of accessing I/O devices by continuously checking the status flags.
  a. Programmed I/O
  b. Interrupt driven I/O
  c. DMA
  d. None of the above

ii. The address of an ISR is termed as
  a. interrupt location
  b. interrupt vector
  c. interrupt address
  d. none of the above

iii. ______ is used to store the return address of ISR.

Space for learners

a. Registers
b. Cache
c. System heap
d. Stack

iv. In case of interrupt driven I/O, I/O module sends _________ signal to the processor when an I/O device is ready for data transfer.
a. interrupt request
b. interrupt acknowledgement
c. read/write
d. none of the above

v. After receiving an interrupt, the signal delivered from the processor to the device is
a. interrupt request
b. interrupt acknowledgement
c. read/write
d. none of the above

vi. _________ is a technique to handle multiple interrupt.
a. Software polling
b. Daisy Chaining
c. Multiple interrupt line
d. all of the above

vii. DMA transfer is initiated by the
a. DMA controller
b. processor
c. I/O device
d. none of the above

viii. _______ is responsible for controlling the transfer of data during DMA.
a. DMA controller
b. processor
c. I/O device
d. none of the above

ix. During DMA transfer, ______ becomes the master of the system bus.
a. DMA controller
b. processor
c. I/O device
d. none of the above

x. The method by which the DMA controller steals the processor's access cycles is known as

Space for learners

a. bust mode
b. cycle stealing
c. memory stealing
d. bus stealing

Space for learners

5.6 SUMMING UP

- I/O devices provide a way of interchanging data between the external environment and the computer. Different I/O devices have different data transfer rates, different data formats and different word lengths.
- Due to the differences present, the processor or the memory does not interact with the I/O devices rather I/O modules are used to establish the interactions between I/O devices and the processor (or memory) acts as a mediator.
- The I/O operations are performed using three techniques: programmed I/O, interrupt driven I/O and direct memory access.

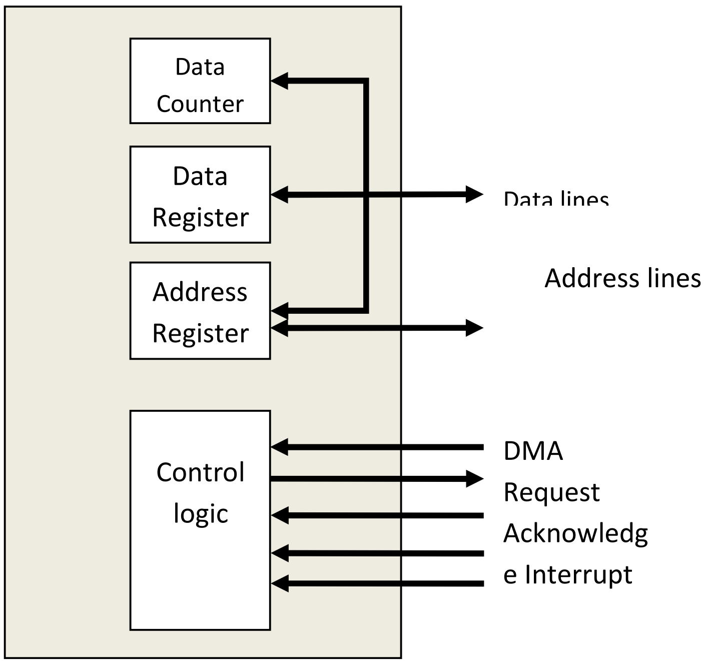


Fig. 5.5 Structure of a DMA controller

- In programmed I/O, the processor exchanges data with the I/O module. The processor allows the I/O module to control the I/O operations directly. The processor senses the status of I/O continuously until the device is ready. It transfers the data only when the I/O is ready.
- In interrupt driven I/O, instead the processor checking repetitively the status of I/O, the I/O module sends interrupts to the processor when it is ready. The processor continues with meaningful tasks after initiating the I/O transfer without waiting for the I/O to get ready.
- DMA is a data transfer technique in which transfer of data from memory to I/O takes place without the active involvement of the processor. To accomplish this, an additional module called a DMA controller is required. A DMA controller shares the system bus along with processor, memory and I/O.

Space for learners

5.7 ANSWERS TO CHECK YOUR PROGRESS

i. a	**ii. b**	**iii. c**	**iv. a**	**v. b**
vi. d	**vii. b**	**viii. a**	**ix. a**	**x. b**

5.8 POSSIBLE QUESTIONS

1. What is meant by interrupt?
2. What is the difference between Programmed I/O and Interrupt driven I/O?
3. How does a computer handle multiple interrupt?
4. What is meant by interrupt priority? What are techniques available to handle priority interrupt?
5. What is polling?
6. What is the difference between Software and Hardware Polling?
7. What is the advantage of DMA transfer?
8. What are the different techniques used for DMA transfer?
9. Differentiate between Cycle Stealing and Burst Mode.

10. What are the major components of a DMA controller?

Space for learners

5.9 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, ZvonkoVranesic, SafwatZaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.