CBCS SCHEME

USN | | | | | | | | | |

18EC35

Third Semester B.E. Degree Examination, Feb./Mar. 2022
Computer Organization and Architecture

Time: 3 hrs. Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

1 a. Explain the basic operational concept between the processor and memory with neat block diagram. **(08 Marks)**

b. Explain the various parameters affecting the performance of a computer and also provide the basic performance equation. **(08 Marks)**

c. Write a short note on single bus structure with neat diagram. **(04 Marks)**

OR

2 a. List out and explain the three systems used for representing signed numbers and also brief about the modular number system concept. **(08 Marks)**

b. Explain IEEE standard used for single and double precession floating point number representation with examples. **(08 Marks)**

c. Write a short note on Big-endian and little-endian assignment. **(04 Marks)**

Module-2

3 a. What is addressing mode? Explain any four addressing modes with examples. **(08 Marks)**

b. What are assembler directives? Explain about the various directives used in the program with example. **(08 Marks)**

c. Write a short note on the assembly and execution of programs. **(04 Marks)**

OR

4 a. With neat diagram and program example, explain a simple I/O task between processor, keyboard and display. **(10 Marks)**

b. What is subroutine? Illustrate the subroutine function with parameter passing by value and reference with suitable program. **(10 Marks)**

Module-3

5 a. Explain the concept of memory mapped I/O with neat diagram of I/O interface with program example. **(10 Marks)**

b. Write short notes on: (i) Interrupt hardware (ii) Interrupt nesting **(10 Marks)**

OR

6 a. What is an interrupt? Explain about various implementation techniques of interrupt. **(10 Marks)**

b. Explain how simultaneous interrupt request is handled using the concept of Daisy Chain. **(10 Marks)**

Module-4

7 a. Explain the internal organization of memory chips with example. **(08 Marks)**

b. Explain the internal organization of 2M × 8 DRAM chip with neat diagram. **(08 Marks)**

c. Write a short note on ROM. **(04 Marks)**

OR

8 a. Discuss about the use of cache memory in the processor system. **(08 Marks)**
b. What is virtual memory? Explain its organization with neat diagram. **(08 Marks)**
c. Write a short note on magnetic hard disk. **(04 Marks)**

Module-5

9 a. Explain single-bus organization of the data path inside a processor with neat diagram. **(10 Marks)**
b. Explain the process of fetching a data word from memory using respective registers of a processor with neat diagram. **(10 Marks)**

OR

10 a. Explain the control signal generation required for proper sequence of instructions in the processor. **(10 Marks)**
b. What is microprogrammed control? Explain its basic organization with suitable diagram and example. **(10 Marks)**

* * * * *

1 a) To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as operands are also stored in the memory. Transfers between memory and processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data is transferred to or from the memory. The memory and processor connection is shown in Fig.

The Instruction register (IR) holds the instruction that is currently being executed. Its output is available to control circuits which generate the timing signals that control various processing elements involved in executing the instruction. The Program Counter (PC) holds the address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed. MAR and MDR facilitate communication with the memory.MAR (Memory Address Register) hold the address of the location to be accessed and MDR (Memory Data Register) contains data written into or read out of the addressed location.

1 b) The total time required to execute the program is known as *elapse time*. This is a measure of performance of entire computer system. The periods during which processor is active is used to measure the performance of processor. The sum of these periods is referred to as *processor time*. The processor time depends on the hardware involved in the execution of individual machine instructions. This hardware comprises the processor and the memory which are connected by a bus.

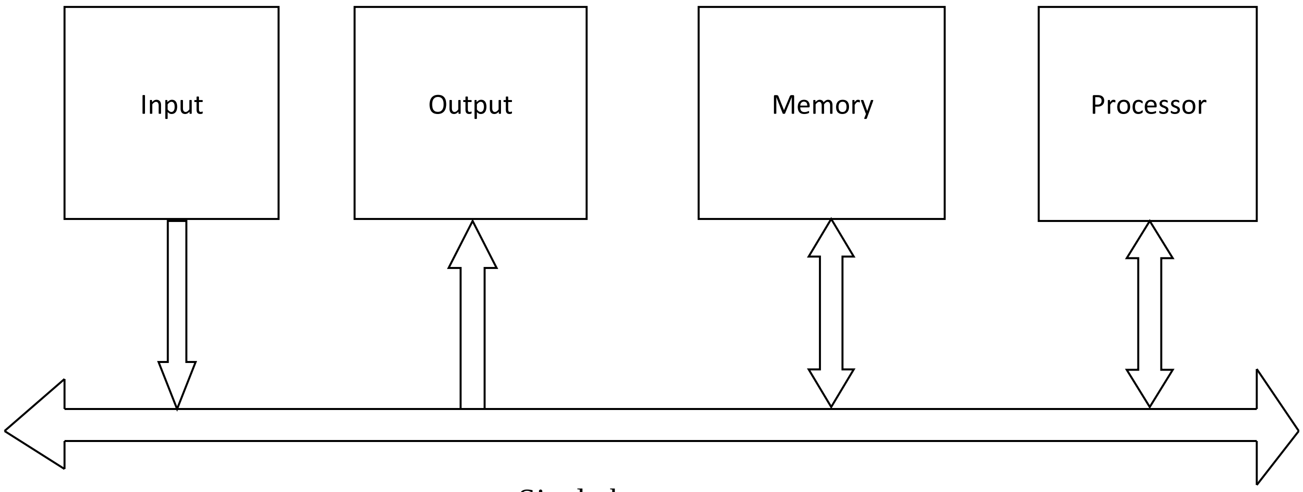
Processor circuits are controlled by a timing signal called *clock*. The clock defines regular time intervals called *clock cycles*. To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each can be completed in one clock cycle. The length P of one clock cycle is an important parameter that affects the processor performance. Its inverse is the clock rate, $R = 1/P$ which is measured in cycles per second.

Let T be the processor time required to execute a program that has been prepared by some high level language. The compiler generates machine level object program that corresponds to source program. Assume that complete execution of the program requires the execution of N machine language instructions. Suppose that the average number of basic steps needed to execute one machine instruction is S, where each basic step is completed in one clock cycle. If the clock rate is R cycles per second, the program execution time is given by *basic performance equation.*

$$T = \frac{N \times S}{R}$$

To achieve high performance, the value of T must be reduced which can be done by reducing N and S, and increasing R. The value of N is reduced if the source program is compiled in fewer machine instructions. The value of S is reduced if instructions have a smaller number of basic steps to perform or if the execution of instructions are overlapped. Using a higher-frequency clock increases the value of R which means the time required to complete a basic execution step is reduced.

1 c) The individual parts of a computer need to be connected in an organized way to increase the speed of operation. When a word of data is transferred between units, all its bits are transferred in parallel that is bits are transferred simultaneously over many wires or lines, one bit per line. A group of lines that serves as a connecting path for several devices is called a bus. The simplest way to inter connect functional units is to use a single bus as shown in Fig.



Single-bus structure

The main virtue of single-bus structure is its low cost and its flexibility for attaching peripheral devices. Systems that contain multiple buses achieve more concurrency in operation by allowing two or more transfers to be carried out at the same time. This leads to better performance but at increased cost. A common approach is to include buffer registers with the devices to hold the information during transfers.

2 a) Three systems are used for representing such numbers:

- Sign-and-magnitude
- 1's-complement
- 2's-complement

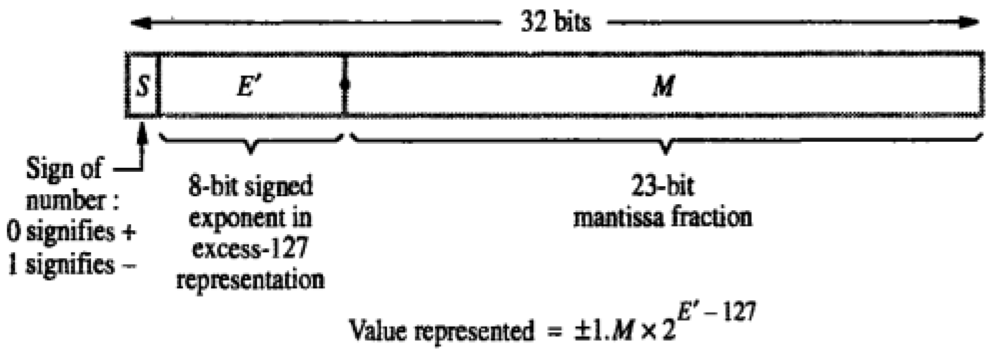
In the *sign-and-magnitude* system, negative values are represented by changing the most significant bit from 0 to 1 in the *B* vector as shown in Fig 1.7. In *1's complement* representation, negative values are obtained by complementing each bit of the corresponding positive number. Finally in *2's-complement* system, forming 2's complement of a number is done by subtracting that numbers from2^n . Hence the 2's complement of a number is obtained by adding 1 to the 1's complement of that number. The 2's complement system yields the most effective way to carry out addition and subtraction operations.

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	$+7$	$+7$	$+7$
0 1 1 0	$+6$	$+6$	$+6$
0 1 0 1	$+5$	$+5$	$+5$
0 1 0 0	$+4$	$+4$	$+4$
0 0 1 1	$+3$	$+3$	$+3$
0 0 1 0	$+2$	$+2$	$+2$
0 0 0 1	$+1$	$+1$	$+1$
0 0 0 0	$+0$	$+0$	$+0$
1 0 0 0	-0	-7	-8
1 0 0 1	-1	-6	-7
1 0 1 0	-2	-5	-6
1 0 1 1	-3	-4	-5
1 1 0 0	-4	-3	-4
1 1 0 1	-5	-2	-3
1 1 1 0	-6	-1	-2
1 1 1 1	-7	-0	-1

2 b) A computer must be able to represent numbers and operate on them in such a way that the position of the binary point is variable and automatically adjusted as the computation proceeds. In such a case binary point is said to float and the numbers are called *floating-point numbers*. For example, in the familiar decimal scientific notation, the numbers can be written as $6.0247 \times 10^{23}, 6.6254 \times 10^{-27}$. The numbers are said to be given to five *significant digits*. When the decimal point is placed to the right of the first (nonzero) significant digit, the number is said to be *normalized*. The floating-point number is represented by its sign, string

of significant digits, commonly called *mantissa*, and an exponent to an implied base for the scale factor.

The standard for representing floating-point numbers in 32 bits is specified by IEEE is given in Fig. A 24 bit mantissa can represent a 7-digit decimal number and an 8 bit exponent. The sign of the number is given in the first bit, followed by the representation for the exponent (to the base 2) of the scale factor. The signed exponent component E is stored in the form of unsigned integer $E' = E + 127$. This is called excess-127 format.


32 bits
S
E'
M
Sign of number : 0 signifies + 1 signifies −
8-bit signed exponent in excess-127 representation
23-bit mantissa fraction
Value represented = ±1.M × 2^(E' − 127)

2 c) The name *big-endian* is used when the lower byte addresses are used for the most significant bytes (the leftmost bytes) of the word. The *little-endian* is used for the opposite ordering, when the lower byte addresses for the less significant bytes (the rightmost bytes) of the word. In both cases, byte addresses 0,4,8, ..., are taken as the address for the successive words in the memory and are the addresses used when specifying the memory read and write operation for the words. The two ways that the byte addresses can be used across the words as shown in Fig

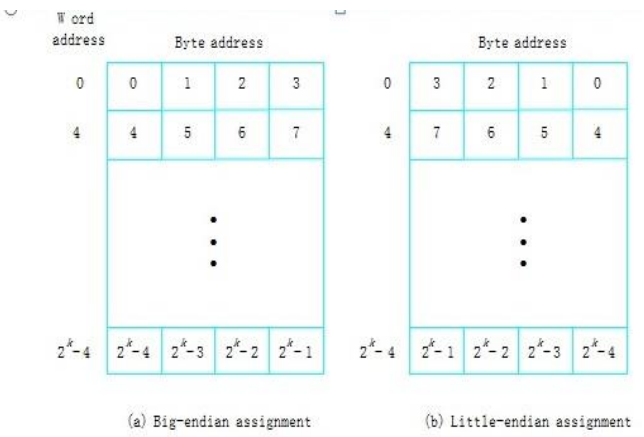
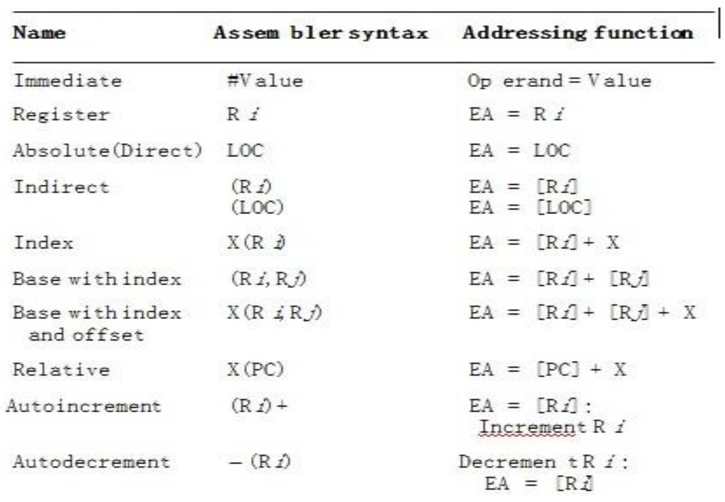

Word address
Byte address
0
4
2^k-4
0 1 2 3
4 5 6 7
2^k-4 2^k-3 2^k-2 2^k-1
(a) Big-endian assignment
Byte address
3 2 1 0
7 6 5 4
2^k-1 2^k-2 2^k-3 2^k-4
(b) Little-endian assignment

Figure 2.7. Byte and word addressing.

The word locations have *aligned* addresses where the word begins at a byte address that is a multiple of number of bytes in a word. If the word length is 16 (2 bytes), aligned words begins at byte addresses 0,2,4 ....

3a) The different ways in which the location of an operand is specified in an instruction is known as *addressing modes*.



Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute(Direct)	LOC	EA = LOC

Indirect	(R*i*)
	(LOC)
		EA = [R*i*]
		EA = [LOC]

Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*] ; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i* ; EA = [R*i*]

3b) statement does not denote the instruction that will be executed when the object program is run. It informs the assembler that the name SUM should be replaced by the value 200 wherever it appears in the program. Such statements are *assembler directives* (or commands) are used by the assembler when it translates the source program in to a object program.

ORIGIN is a directive that tells the assembler program where in the memory to place the data block.

DATAWORD directive is used to inform the assembler to place the data in the address.

RESERVE directive declares a memory block and does not cause any data to be loaded in these locations.

ORIGIN directive specifies that the instructions of an object program are to be loaded in the memory starting at an address.

END is directive which indicates the end of the source program text. The END directive includes the label START, which is the address of the location at which execution of the program is to begin.

RETURN is an assembler directive that identifies the point at which the execution of the program should be terminated.

3c) assembly language requires statements in a source program to be written in the form

Label Operation Operand(s) Comment

The assembler program replaces all the symbols denoting operations and addressing modes with the binary codes used in the machine instructions and replaces all names and labels with their actual values. The assembler assigns addresses to instructions and data blocks, starting at addresses given in the ORIGIN assembler directives. It also determines the value that replaces the names. The value of the name is specified by EQU directive. In some cases, the assembler does not directly replace a name representing an address with the actual value of the address. This is done in branch instruction. The assembler computes the branch offset, which is the distance to the target and puts in to the machine instruction.

The *two-pass assembler* has two phases. In the first pass, the assembler scans through the source program and keeps track of the numerical value that corresponds to these names in *symbol table.* The assembler then goes through the source program a second time and substitutes the values for all the names from the symbol table. Assembler stores the object the object program on a magnetic disk. Executing the loader performs a sequence of input operations needed to transfer machine language program from a disk into a specified place in the memory. Debugger program is used to find the programming errors. This program enables the programmer to stop execution of the object program at some points of interest and to examine the contents of various processor registers and memory locations.

4a) The problem of moving a character code from the keyboard to the processor. Striking a key store the corresponding character code in an 8-bit buffer register associated with the keyboard. Let us call this register DATAIN as shown in Fig 2.3. To inform the processor that a valid character is in DATAIN, a status control flag, SIN, is set to 1. A program monitor SIN, and when SIN is set to 1, the processor reads the contents of DATAIN. When the character is transferred to processor, SIN is automatically cleared to 0. If the second character is entered at the keyboard, SIN is again set to 1 and the process repeats.

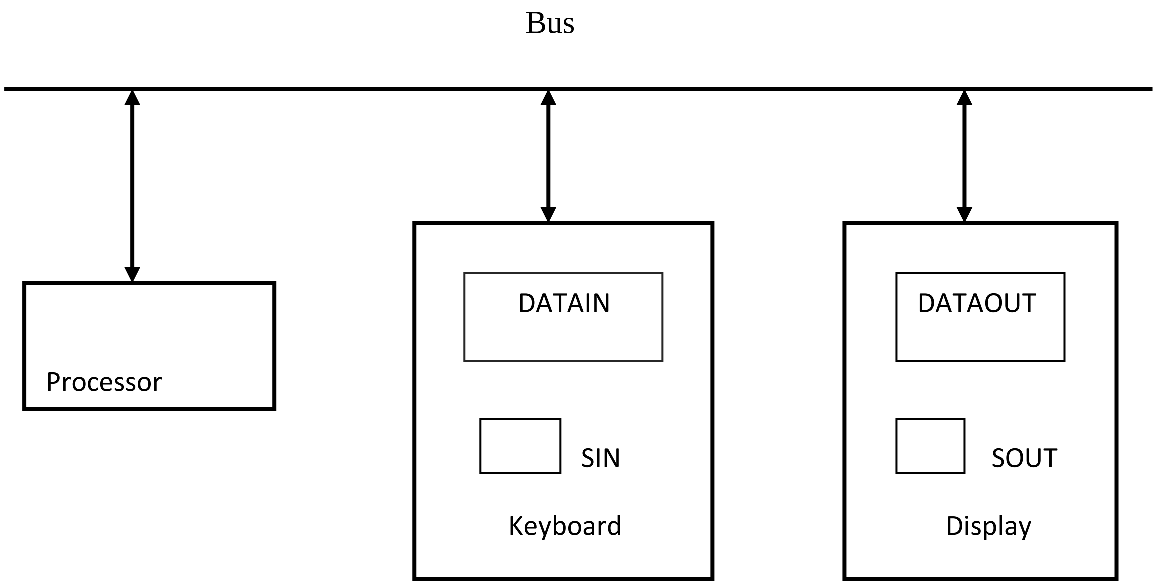


Fig 2.3: Bus connection for processor, keyboard and display

An analogous process takes place when the characters are transferred from the processor to display. A buffer register, DATAOUT and a status control register SOUT are used for this transfer. When SOUT equals 1, the display is ready to receive a character. Under program control, the processor monitors SOUT and when SOUT is set to 1, the processor transfers a character code to DATAOUT. The transfer of character to DATAOUT clears

SOUT to 0, when the display is ready to receive a second character; SOUT is set again to 1. The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry known as *device interface.* The processor can monitor the keyboard status flag SIN and transfer a character from DATAIN to register R! By the following sequence of operations:

READWAIT Branch to READWAIT if SIN=0

Input from DATAIN to R1

The first instruction tests the status flag and the second performs the branch. The processor monitors the status flag by executing a short *wait loop* and proceeds to transfer the input data when SIN is set to 1 as a result of key being struck. The input operation resets SIN to 0. The sequence of operations are used for transferring the output to display are

WRITEWAIT Branch to WRITEWAIT if SOUT=0

Output from R1 to DATAOUT

4b) It is often necessary to perform a particular subtask many times on different data values. Such subtask is called *subroutine.* When a program branches to a subroutine we call that it is *calling* a subroutine. The instruction that performs this branch operation is called a Call instruction. The subroutine is said to *return* to program that called it by executing a Return instruction. The location where the calling program resumes execution is the location pointed by the updated PC while the Call instruction being executed. Hence the contents of the PC must be saved by the Call instruction to enable correct return to the calling program. This way in which the computer makes it possible to call and return from subroutines is referred to as *subroutine linkage method.*

The Call instruction is a special branch instruction that performs the following operations:

1. Store the contents of PC in the link register.
2. Branch to the target address specified by the instruction.

The Return instruction is a special branch instruction that performs the operation:
Branch to the address contained in the link register.

Passing parameters through processor registers is straightforward and efficient. Fig shows a program for adding a list of numbers using a subroutine with parameters passed through registers.

Calling program

```
Move    N, R1          R1 serves as a counter
Move    #NUM1, R2      R2 points to the list
Call    LISTADD        Call subroutine
Move    R0, SUM        Save result
          •
          •
          •
```

Subroutine

```
LISTADD   Clear          R0              Initialize sum to 0
LOOP      Add            (R2)+, R0       Add entry from list
          Decrement      R1
          Branch > 0     LOOP
          Return                         Return to calling program
```

Fig 2.7: Program written as a subroutine; parameters passed through registers

If many parameters are involved, there may not be enough general purpose registers available for passing them to subroutine. Using a stack is highly flexible as stack can handle a large number of parameters. The following example illustrates this approach.

Assume that top of the stack is at level 1 below.

```
Move      #NUM1,   -(SP)     Push parameters onto stack
Move       N, -(SP)
Call       LISTADD           Call subroutine   (top of stack at level 2)

Move      4(SP), SUM         Save result.
Add       #8, SP             Restore top of stack    (top of stack at level 1)
             •
             •
             •

LISTADD   MoveMultiply     R0-R2, -(SP)     Save registers   (top of stack at level 3)

          Move             16(SP), R1       Initialize counter to n.
          Move             20(SP), R2       Initialize pointer to the list.
          Clear            R0               Initialize sum to 0.
LOOP      Add              (R2) +, R0       Add entry from list
          Decrement        R1
          Branch>0         LOOP
          Move             R0, 20(SP)       Put result on the stack
          MoveMultiply     (SP) +, R0-R2    Restore registers
          Return                            Return to calling program
```

calling program pushes the address NUM1 and the value *n* onto the stack and calls the subroutine LISTADD. The call instruction also pushes the return address onto the stack. The top of the stack is at level 2. We have single instruction *MoveMultiply,* to store the contents of registers R0 through R2 on the stack. The top of the stack is at level3. The subroutine accesses the parameters *n* and NUM1 from the stack using indexed addressing. At the end of computation, the register R0 contains the sum. Before subroutine returns to the calling program, the contents of R0 are placed onto the stack, replacing the parameter NUM1 which is no longer needed. Then the contents of the three registers used by the subroutine are restored from the stack. Now the top item on the stack is the return address at level 2. After subroutine returns, the calling program stores the result in location SUM and lowers the top of the stack to its original level by incrementing the stack pointer by 8.

5a) A simple arrangement to connect I/O devices to a computer is to use single bus arrangement as shown in Fig1. The bus enables the devices connected to it to exchange information. It consists of three set of lines to carry address, data and control signals. Each I/O device is assigned unique set of addresses.

With the memory mapped I/O any machine instruction that can access memory can be used to transfer data to or from an I/O device.

Move DATAIN, R0

Reads data from the DATAIN and stores into processor register R0. Similarly

Move R0, DATAOUT

Sends the contents of register R0 to location DATAOUT which is the output data buffer of a display unit or a printer.

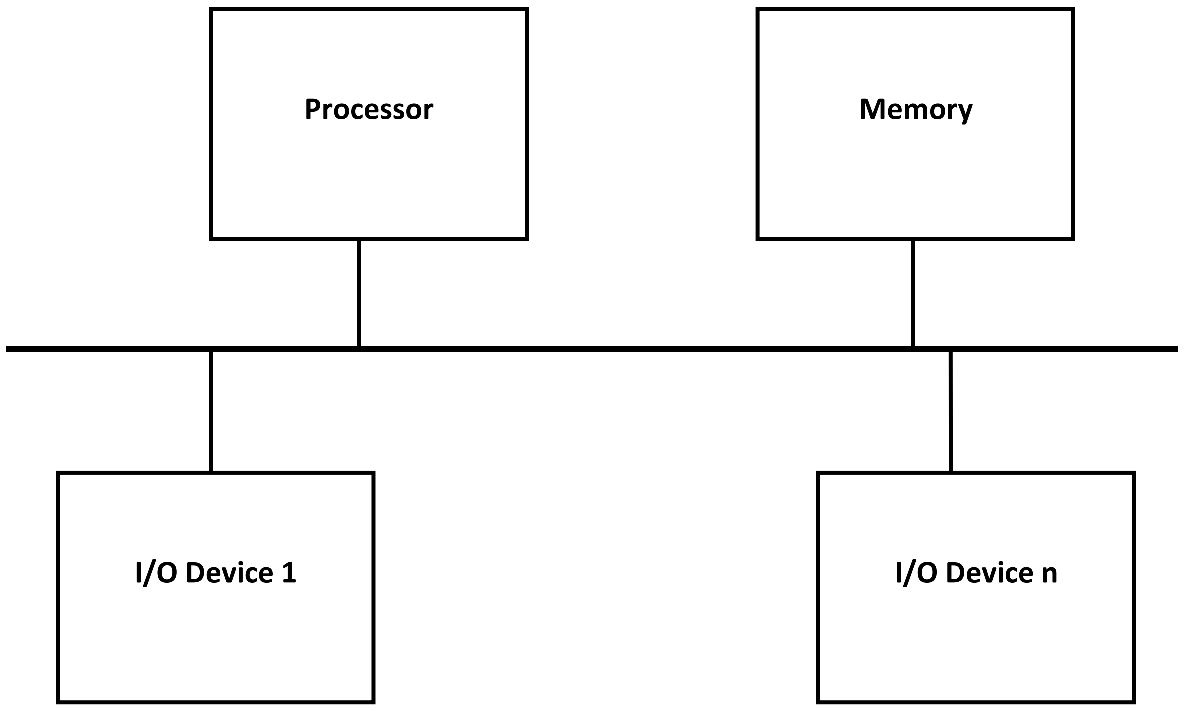
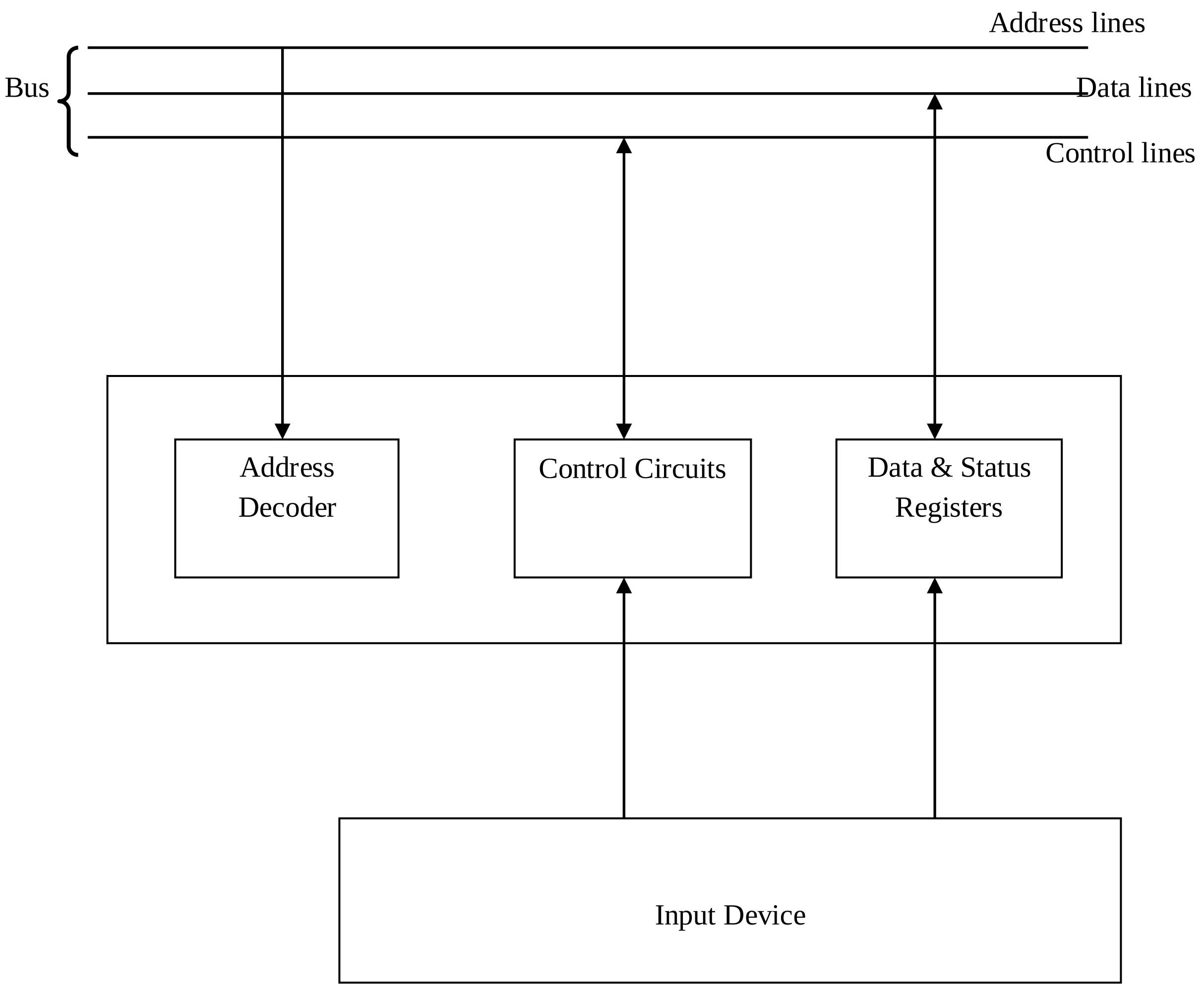


Fig illustrates the hardware required to connect the I/O device to the bus. The address decoder enables the device to recognize its address when its address appears on the address lines. The data register holds the data being transferred to or from the processor. The status register contains the information relevant to the operation of I/O device. Both status and data registers are connected to the data bus and assigned unique addresses. The address decoder, data & status registers and the control circuitry required to coordinate I/O transfers constitutes the device interface circuit.



I/O interface of an input device

5b) Interrupt Hardware: I/O device requests an interrupts by activating a bus line called interrupt request. A single interrupt may be used to serve ‘n’ devices.

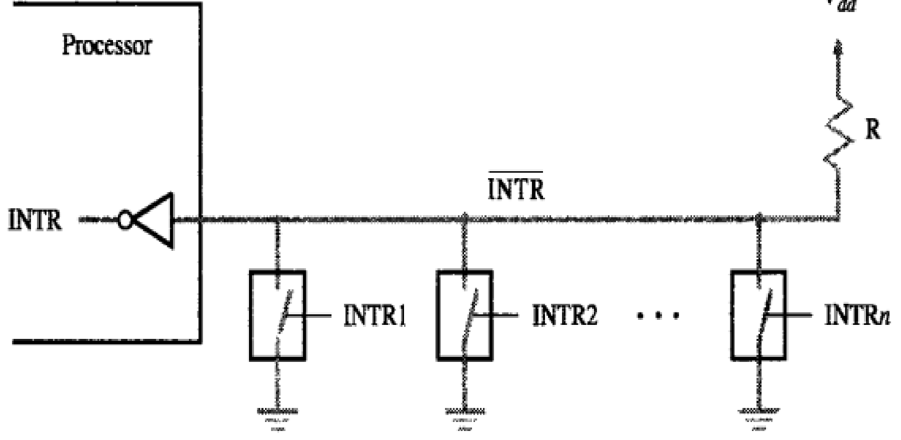


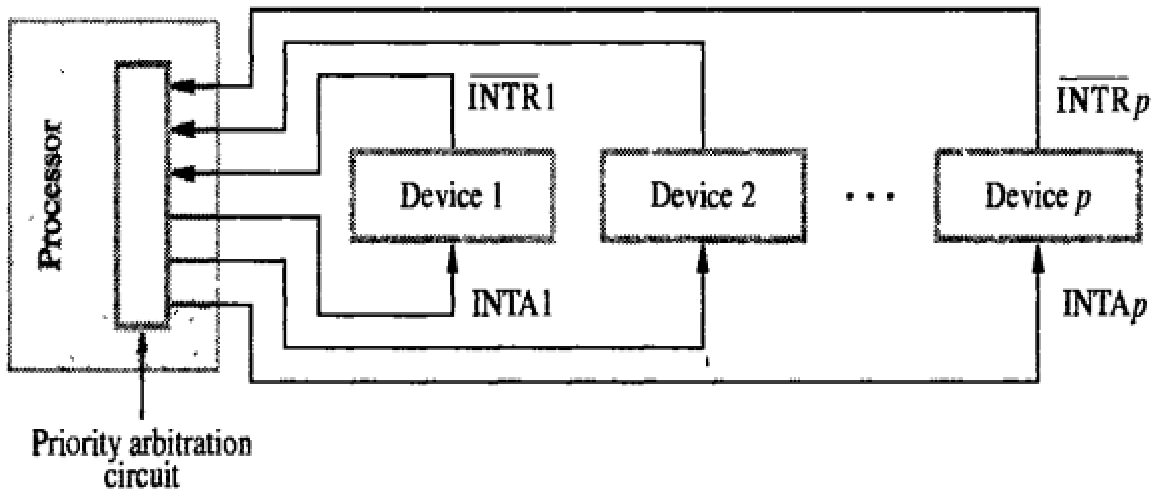
Fig 5: Circuit for common interrupt request line

To request an interrupt device closes its associated switch. Thus if all the interrupt request signals INTR1 to INTRn are inactive, the voltage on the interrupt request line is Vdd. This is the inactive state of the line. When the device requests the interrupt by closing its switch, the voltage line drops to zero causing the interrupt request line INTR received by the processor to go to 1. The value of INTR is the logical OR of the requests from individual devices, that is

$$INTR=INTR_1 + ...+ INTR_n$$

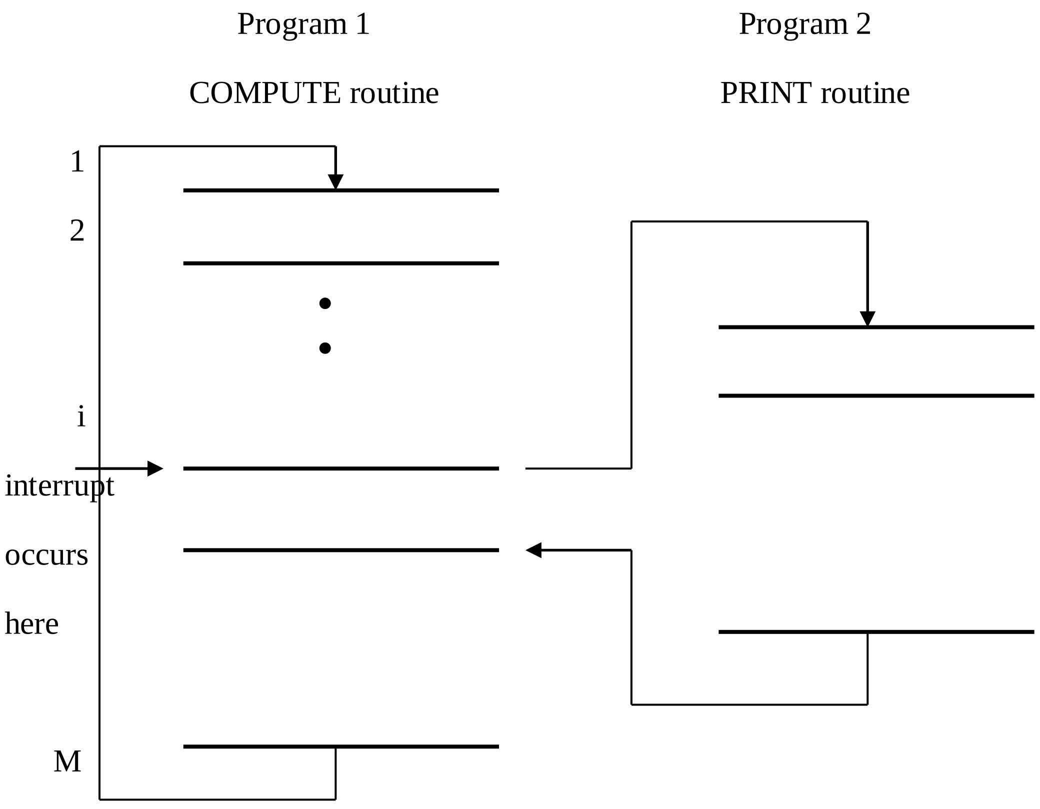
R is the pull up register because it pulls line voltage up to high voltage when the switches are open.

Interrupt Nesting: I/O devices should be organized in a priority structure. An interrupt request from a high priority should be accepted while the processor is serving another request from the lower priority device. We can assign priority level to the processor that can be changed under program control. The priority level of the processor is the priority of the program that is currently being executed. The processor accepts interrupts from devices that have priorities higher than its own. The processor is in supervisory mode when it is executing the OS routines. It switches to User mode before beginning to execute application programs. The privileged instructions can be executed only while the processor is running in the supervisory mode. A multiple priority scheme can be implemented by using separate interrupt request and interrupt acknowledge lines from each device.



Implementation of interrupt priority using individual interrupt request & acknowledge lines

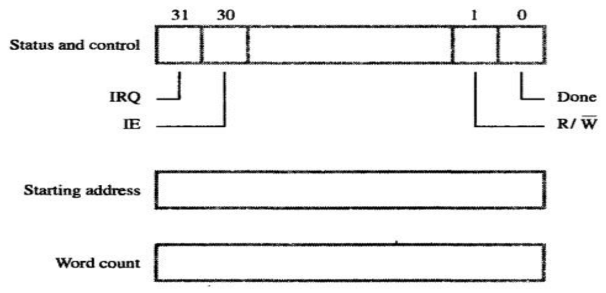
6a) The other tasks can be performed by the processor while waiting for the I/O device to become ready. When the I/O device becomes ready, it sends a hardware signal called interrupt to the processor. Using the interrupts waiting periods can be eliminated. Consider a task that requires some computations to be performed and the results to be printed on a line printer. This is followed by more computations and output and so on. Let the program consists of two routines COMPUTE and PRINT. Assume COMPUTES produces a set of 'n' lines of output to be printed by PRINT routine.



Transfer of control through the use of interrupts

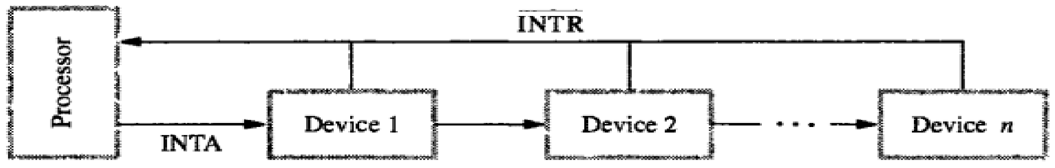
It is possible to overlap printing and computation ie to execute COMPUTE routine while printing is in progress, a faster overlap speed of execution will result. Whenever printer becomes ready, it alerts the processor by sending a interrupt request signal. In response the processor interrupts the COMPUTE routine and transfers the control to the PRINT routine. This process continues until all 'n' lines are printed and PRINT routine ends. If COMPUTE takes longer to generate 'n' lines than the time required to print them, then the processor will be performing useful computations all the time. Saving registers also increases the delay between the time the interrupt request is received and the start of execution of interrupt service routine. This delay is called interrupt latency.

Vectored Interrupts A device requesting an interrupt can identify itself by sending special code to the processor over the bus. The code supplied by the device represents the starting address of the interrupt service routine. The code length is 4 to 8 bits. The processor reads this address called the interrupt vector and stores it in to the PC. The interrupt vector may also include a new value for a processor status register. The interrupted device must wait to put on the bus only when the processor is ready to receive it. When the processor is ready to receive the vector interrupt code, it activates the interrupt acknowledge line INTA. The I/O device responds by sending its interrupt vector code and turning off INTR signal.



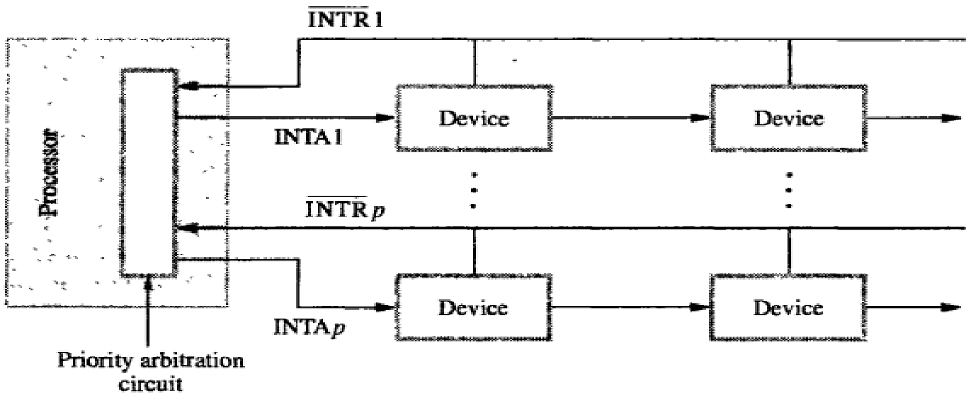
DMA transfers are performed by control circuits that are part of I/O interface called DMA controller. The DMA controller performs functions that would normally be carried out by processor when accessing main memory.   The R/W¯bit determine the direction of transfer. When this bit is set to 1 by a program instruction, the controller performs read operation that is it transfers data from memory to I/O device. When transfer is complete, it sets done flag to 1. When IE is1, it causes the controller to raise an interrupt after it has completed transferring block of data. Finally IRQ bit is set to 1 when it has requested interrupt. Requests from DMA devices are given high priority than processor requests. Among different DMA devices high priority is given to high speed peripherals such as disks, high speed network interface or graphic display device.

6b) If several devices share one interrupt request line, some other mechanism is needed. When several devices raises interrupt request and $\overline{INTR}$ line is activated, the processor responds by setting the INTA line to 1. The signal is received by device 1. Device 1 passes the signal onto device 2 only if it does not require any service. If device 1 has pending request for interrupt, it blocks the INTA signal and proceeds to put its identification code on to data lines. In daisy chain the device that is electrically closest to the processor has the highest priority.



Daisy chain

Devices can be organized in groups and each group is connected at a different priority level. Within group devices are connected in daisy chain.



7a) The following figure shows such an organization of a memory chip consisting of 16 words of 8 bits each, which is usually referred to as a 16 x 8 organization.

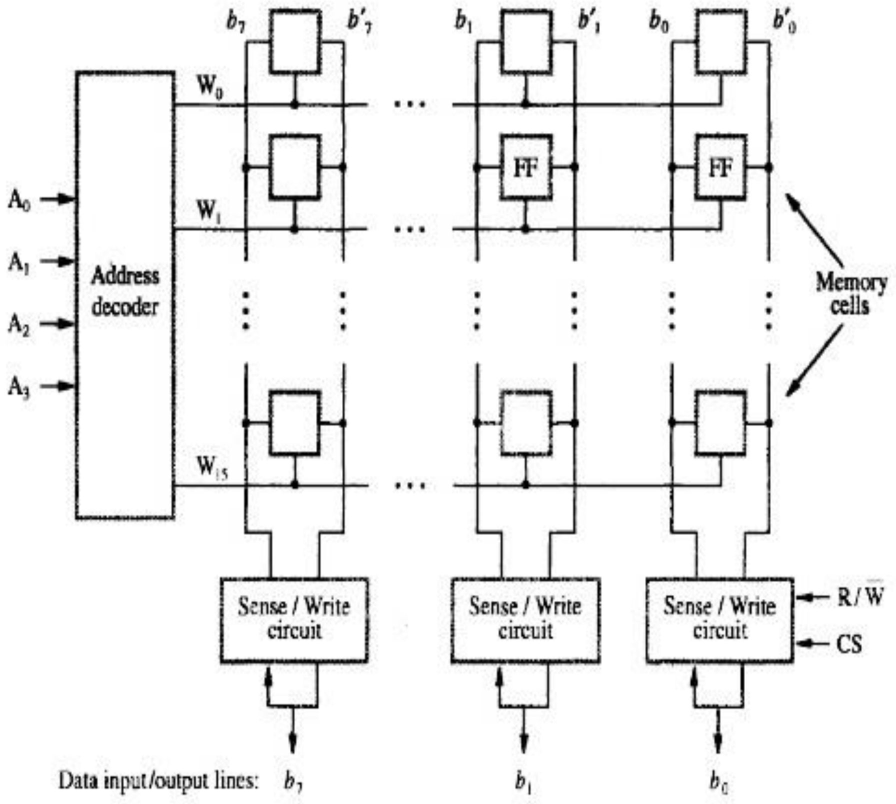


Figure 5.2 Organization of bit cells in a memory chip.

The data input and the data output of each Sense/Write circuit are connected to a single bi-directional data line in order to reduce the number of pins required. One control line, the R/W (Read/Write) input is used a specify the required operation and another control line, the CS (Chip Select) input is used to select a given chip in a multichip memory system. This circuit requires 14 external connections, and allowing 2 pins for power supply and ground connections, can be manufactured in the form of a 16-pin chip. It can store 16 x 8 = 128 bits.

7b)

A 16-megabit DRAM chip, configured as 2M × 8, is shown in Figure 5.7. The cells are organized in the form of a 4K × 4K array. The 4096 cells in each row are divided into 512 groups of 8, so that a row can store 512 bytes of data. Therefore, 12 address bits are needed to select a row. Another 9 bits are needed to specify a group of 8 bits in the selected row. Thus, a 21-bit address is needed to access a byte in this memory. The high-order 12 bits and the low-order 9 bits of the address constitute

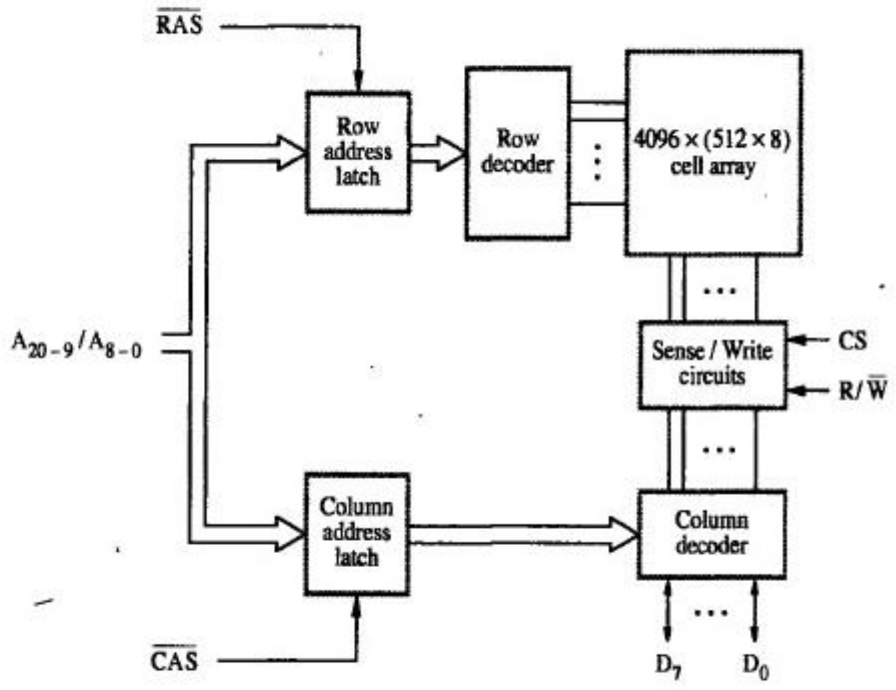


Figure 5.7 Internal organization of a 2M × 8 dynamic memory chip.

7c)

Figure 5.12 shows a possible configuration for a ROM cell. A logic value 0 is stored in the cell if the transistor is connected to ground at point *P*; otherwise, a 1 is stored. The bit line is connected through a resistor to the power supply. To read the state of the cell, the word line is activated. Thus, the transistor switch is closed and the voltage on the bit line drops to near zero if there is a connection between the transistor and ground. If there is no connection to ground, the bit line remains at the high voltage, indicating a 1. A sense circuit at the end of the bit line generates the proper output value. Data are written into a ROM when it is manufactured.

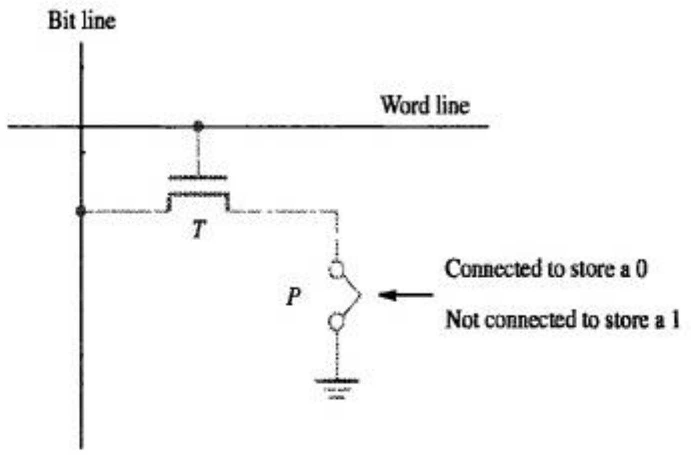


Figure 5.12 A ROM cell.

8a)

Analysis of a large number of typical programs has shown that most of their execution time is spent on a few main row lines in which a number of instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops or few procedure that repeatedly call each other. The main observation is that many instructions in a few localized are as of the program are repeatedly executed and that the remainder of the program is accessed relatively infrequently. This phenomenan is referred to as locality of reference.

If the active segments of a program can be placed in a fast memory, then the total execution time can be significantly reduced, such a memory is referred as a cache memory which is in served between the CPU and the main memory as shown in fig.1

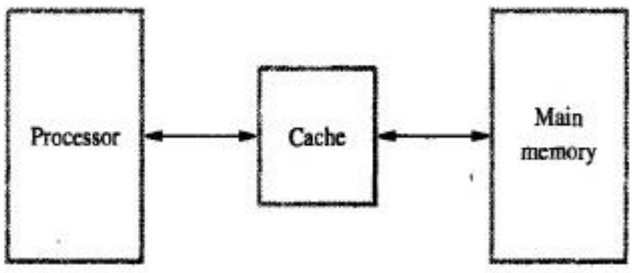


Figure 5.14 Use of a cache memory.

Two Level memory Hierarchy: We will adopt the terms Primary level for the smaller, faster memory and the secondary level for larger, slower memory, we will also allow cache to be a primary level with slower semiconductor memory as the corresponding secondary level. At a different point in the hierarchy, the same S.C memory could be the primary level with disk as the secondary level.

8b) Virtual memory is the technique of using secondary storage such as disks to enter the apparent size of accessible memory beyond its actual physical size. Virtual memory is implemented by employing a memory-management unit (MMU) to translate every logical address reference into a physical address reference as shown in fig 1. The MMU is imposed between the CPU and the physical memory where it performs these translations under the control of the operating system. Each memory reference is sued by the CPU is translated from the logical address space to the physical address space. Mapping tables guide the translation, again under the control of the operating system.

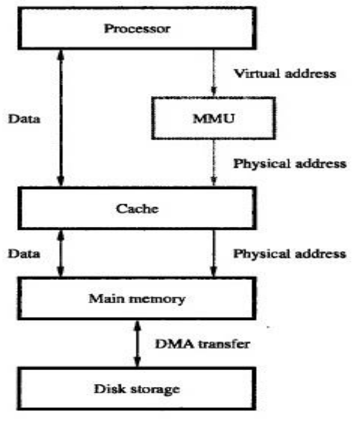


Figure 5.26 Virtual memory organization.

Virtual memory usually demand paging, which means that a Page is moved from disk into main memory only when the processor accesses a word on that page. Virtual memory pages always have a place on the disk once they are created, but are copied to main memory only on a miss or page fault.

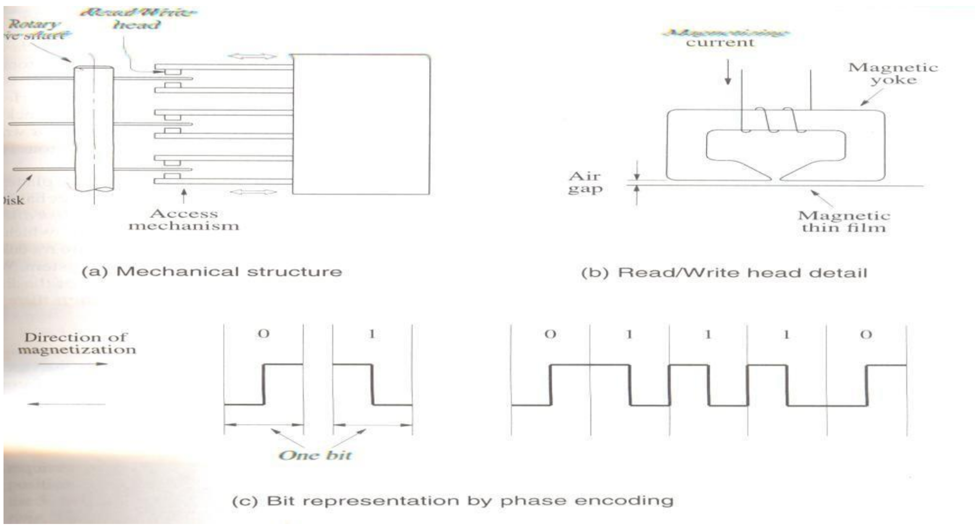
Advantages of Virtual memory:-

1. Simplified addressing
2. Cost effective use of memory
3. Access control

8c) Magnetic Disk system consists o one or more disk mounted on a common spindle. A thin magnetic film is deposited on each disk, usually on both sides.

The disk are placed in a rotary drive so that the magnetized surfaces move in close proximity to read /write heads. Each head consists of **magnetic yoke & magnetizing coil**. Digital information can be stored on the magnetic film by applying the current pulse of suitable polarity to the magnetizing coil. Only changes in the magnetic field under the head can be sensed during the Read operation. Therefore if the binary states 0 & 1 are represented by two opposite states of magnetization, a voltage is induced in the head only at 0-1 and at 1-0 transition in the bit stream. A consecutive (long string) of 0‟s & 1‟s are determined by using the clock which is mainly used for synchronization. Phase Encoding or Manchester Encoding is the technique to combine the clocking information with data. The Manchester Encoding describes that how the self-clocking scheme is implemented.

The Read/Write heads must be maintained at a very small distance from the moving disk surfaces in order to achieve high bit densities. When the disk are moving at their steady state, the air pressure develops between the disk surfaces & the head & it forces the head away from the surface. The flexible spring connection between head and its arm mounting permits the head to fly at the desired distance away from the surface.



9a)

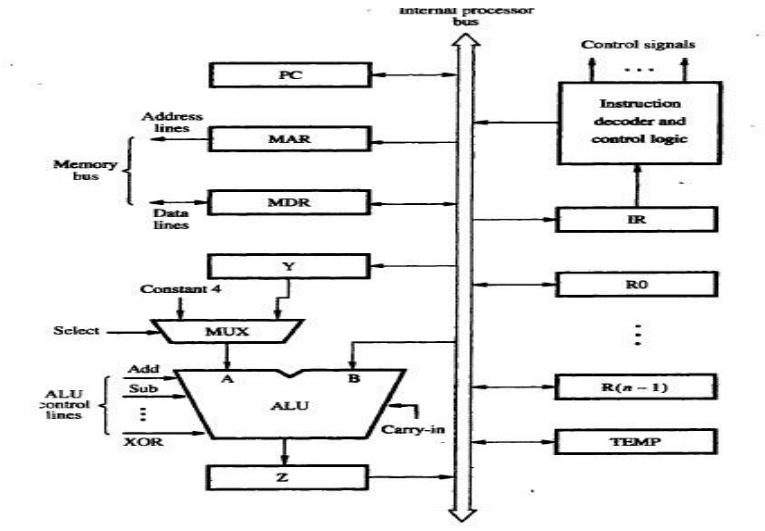


Figure 7.1 Single-bus organization of the datapath inside a processor.

Figure shows an organization in which the arithmetic and logic unit (ALU) and all the registers are interconnected through a single common bus, which is internal to the processor. The data and address lines of the external memory bus are shown in Figure 7.1 connected to the internal processor bus via the memory data register, MDR, and the memory address register, MAR, respectively. Register MDR has two inputs and two outputs. Data may be loaded into MDR either from the memory bus or from the internal processor bus. The data stored in MDR may be placed on either bus. The input of MAR is connected to the internal bus, and its output is connected to the external bus. The control lines of the memory bus are connected to the instruction decoder and control logic block. This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus. The number and use of the processor registers R0 through R(n - 1) vary considerably from one processor to another. Registers may be provided for general-purpose use by the programmer. Some may be dedicated as special-purpose registers, such as index registers or stack pointers. Three registers, Y, Z, and TEMP in Figure 2, have not been mentioned before. These registers are transparent to the programmer, that is, the programmer need not be concerned with them because they are never referenced explicitly by any instruction. They are used by the processor for temporary storage during execution of some instructions. These registers are never used for storing data generated by one instruction for later use by another instruction.The multiplexer MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU. The constant 4 is used to increment the contents of the program counter. We will refer to the two possible values of the MUX control input Select as Select4 and Select Y for selecting the constant 4 or register Y, respectively.As instruction execution progresses, data are transferred from one register to another, often passing through the ALU to perform some arithmetic or logic operation. The instruction decoder and control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register. The decoder generates the control signals needed to select the registers involved and direct the transfer of data. The registers, the ALU, and the interconnecting bus are collectively referred to as the data path.

9b) **Address** into MAR; issue **Read** operation; data into **MDR**.
Response time of each memory access **varies** (cache miss, memory-mapped I/O,…).
Processor **waits until** it receives an indication that the **requested** operation has been **completed (Memory-Function-Completed, MFC).**

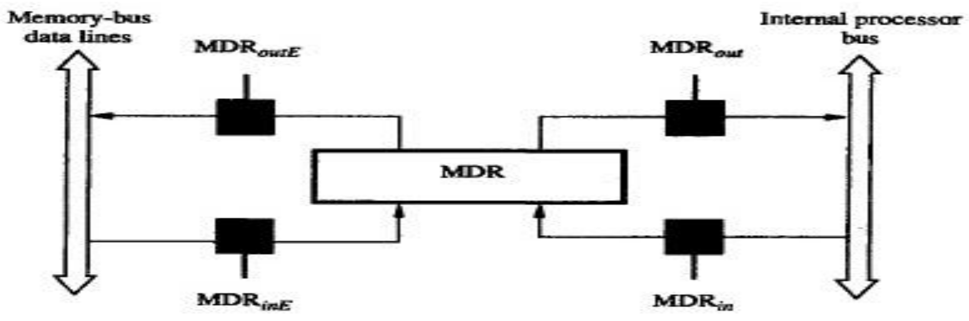


Figure 7.4 Connection and control signals for register MDR.

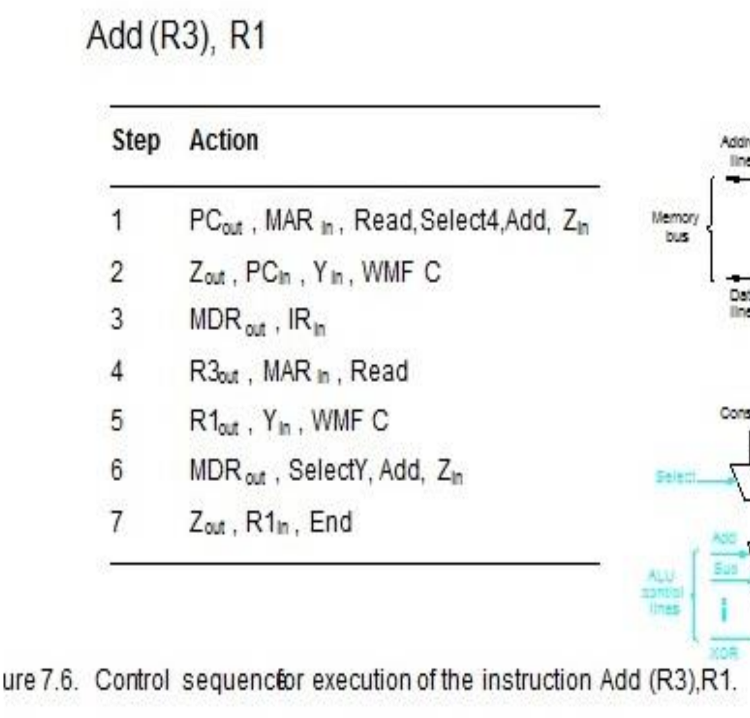
Move (R1), R2

- MAR ← [R1]
- Start a Read operation on the memory bus
- Wait for the MFC response from the memory
- Load MDR from the memory bus
- R2 ← [MDR]

10 a) Consider the instruction

Add (R3), R1

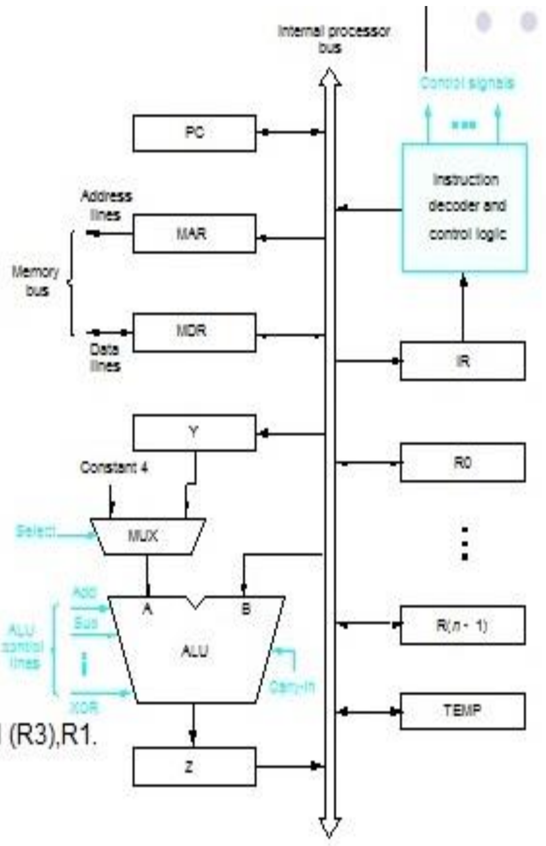
;which adds the contents of a memory location pointed to by R3 to register R1.
Executing this instruction requires the following actions:
1.Fetch the instruction.
2.Fetch the first operand (the contents of the memory location pointed to by R3).
3.Perform the addition.
4 .Load the result into Rl.



Add (R3), R1

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

ure 7.6. Control sequencefor execution of the instruction Add (R3),R1.



The listing shown in figure above indicates the sequence of control steps required to perform these operations for the single-bus architecture of Figure 7.2.
Instruction execution proceeds as follows.
In step 1, the instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a Read request to the memory. The Select signal is set to Select4, which causes the multiplexer MUX to select the constant 4.
This value is added to the operand at input B, which is the contents of the PC, and the result is stored in register Z. The updated value is moved from register Z back into the PC during step 2, while waiting for the memory to respond.
In step 3, the word fetched from the memory is loaded into the IR.
Steps 1 through 3 constitute the instruction fetch phase, which is the same for all instructions. The instruction decoding circuit interprets the contents of the IR at the beginning of step 4. This enables the control circuitry to activate the control signals for steps 4 through 7, which constitute the execution phase. The contents of register R3 are transferred to the MAR in step 4, and a memory read operation is initiated.
Then the contents of Rl are transferred to register Y in step 5, to prepare for the addition operation. When the Read operation is completed, the memory operand is available in register MDR, and the addition operation is performed in step 6. The contents of MDR are gated to the bus, and thus also to the B input of the ALU, and register Y is selected as the second input to the ALU by choosing Select Y. The sum is stored in register Z, then transferred to Rl in step 7. The End signal causes a new instruction fetch cycle to begin by returning to step 1.

10 b)

The control signals required inside the processor can be generated using a control step counter and a decoder/ encoder circuit. Now we discuss an alternative scheme, called micro programmed control, in which control signals are generated by a program similar to machine language programs. A control word (CW) is a word whose individual bits represent the various control signals in Figure 7.15. Each of the control steps in the control sequence of an instruction defines a unique combination of Is and Os in the CW. The CWs corresponding to the 7 steps of Figure 6 are shown in Figure 15. We have assumed that Select Y is represented by Select = 0 and Select4 by Select = 1. A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the micro routine for that instruction, and the individual control words in this micro routine are referred to as microinstructions. The micro routines for all instructions in the instruction set of a computer are stored in a special memory called the control store. The control unit can generate the control signals for any instruction by sequentially reading the CWs of the corresponding micro routine from the control store. This suggests organizing the control unit as shown in Figure 7.16. To read the control words sequentially from the control store, a micro program counter (µPC) is used. Every time a new instruction is loaded into the IR, the output of the block labeled "starting address generator" is loaded into the µPC. The µPC is then automatically incremented by the clock, causing successive microinstructions to be read from the control store. Hence, the control signals are delivered to various parts of the processor in the correct sequence.