UNIT -IV
PROCESSOR AND MEMORY ORGANIZATION

Course Objective:

- To familiarize the concept of Processor Organization and memory Organization.

Syllabus:

Processor Organization: Fundamental Concepts, Execution of a Complete Instruction, Multiple-Bus Organization, Hardwired Control and Micro programmed Control

Memory Organization: Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size and Cost, Speed, Size and Cost, Cache Memories, Secondary Storage.

Course Outcomes:

At the end of the unit, student will be able to:

- How a processor executes instructions.
- The internal functional units of a processor and how they are interconnected.
- Hardware for generating internal control signals.
- Organization of a main memory
- Significance of virtual memory and cache memory.
- Magnetic and optical disks used for secondary storage.

Syllabus

PROCESSOR ORGANIZATION

4.1 Basic Processing Unit-Some Fundamental Concepts

- To execute a program, the processor fetches one instruction at a time and performs the operations specified. Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- The processor keeps track of the address of the memory location containing the next instruction to be fetched using the program counter, PC. After fetching an instruction, the contents of the PC are updated to point to the next instruction in the sequence. A branch instruction may load a different value into the PC. Another key register in the processor is the instruction register, IR.
- Suppose that each instruction comprises 4 bytes, and that it is stored in one memory word. To execute an instruction, the processor has to perform the following three steps:

1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are the instruction to be executed; hence they are loaded into the IR. In register transfer notation, the required action is

$$IR \leftarrow [[PC]]$$

2. Increment the PC to point to the next instruction. Assuming that the memory is byte addressable, the PC is incremented by 4; that is

$$PC \leftarrow [PC]+4$$

3. Carryout the operation specified by the instruction in the IR.

Fetching an instruction and loading it into the IR is usually referred to as the ***instruction fetch phase***. Performing the operation specified in the instruction constitutes the ***instruction***

execution phase.

4.1.1 Single Bus organization of Processor:

- Figure 4.1 shows the organization in which the arithmetic and logic unit (ALU) and all the registers are interconnected via a single common bus. This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.

- The data and address lines of the external memory bus are 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.

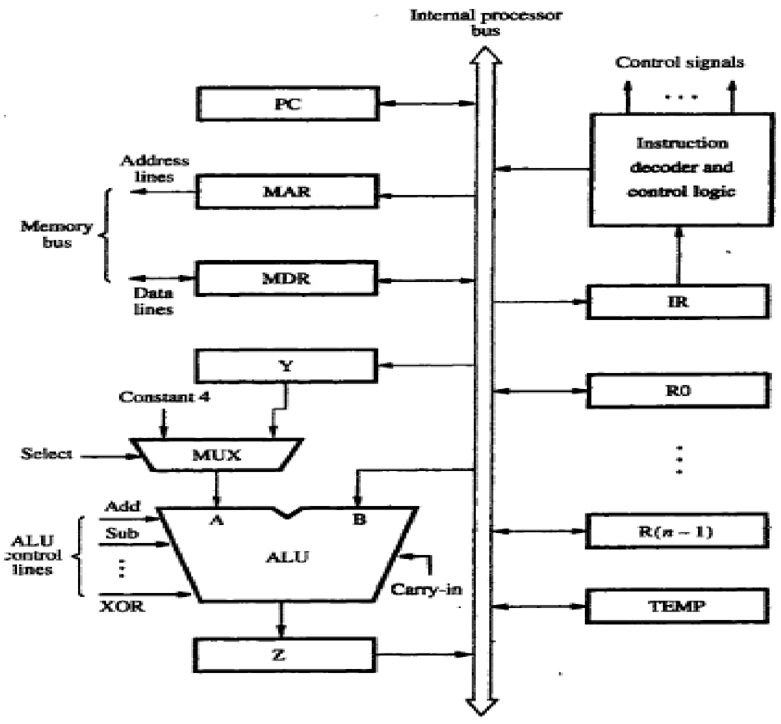


Figure 4.1: Single Bus Organization

- Three registers Y, Z, and TEMP registers are used by the processor for temporary storage during execution of some instructions. 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.

- With few exceptions, an instruction can be executed by performing one or more of the following operations in some specified sequence:
  - ➢ Transfer a word of data from one processor register to another or to the ALU
  - ➢ Perform an arithmetic or a logic operation and store the result in a processor register
  - ➢ Fetch the contents of a given memory location and load them into a processor register
  - ➢ Store a word of data from a processor register into a given memory location

4.1.2 Register Transfers

- Instruction execution involves a sequence of steps in which data are transferred from one register to another. For each register, two control signals are used to place the contents of that register on the bus or to load the data on the bus into the register.

- The input and output of register Ri are connected to the bus via switches controlled by the signals Ri_{in} and Ri_{out}, respectively. When Ri_{in} is set to 1, the data on the bus are loaded into Ri. Similarly, when Ri_{out}, is set to 1, the contents of register Ri are placed on the bus. While Ri_{out} is equal to 0, the bus can be used for transferring data from other registers.

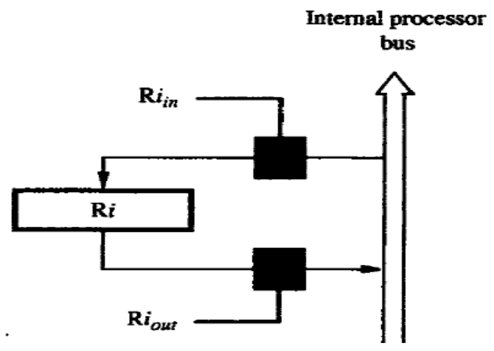


Figure 4.2: Register transfer

- Suppose that we wish to transfer the contents of register R1 to register R4. This can be accomplished as follows:
  - o Enable the output of register R1 by setting $R1_{out}$ to 1. This places the contents of R1 on the

processor bus.

- Enable the input of register R4 by setting $R4_{in}$,to1. This loads data from the processor bus into register R4.

All operations and data transfers within the processor take place within time periods defined by the ***processor clock*.**

4.1.3 Performing an Arithmetic or Logic Operation

- The ALU is a combinational circuit that has no internal storage. It performs arithmetic and logic operations on the two operands applied to its A and B inputs In figures 4.1 and 4.3, one of the operands is the output of the multiplexer MUX and the other operand is obtained directly from the bus. The result produced by the ALU is stored temporarily in register Z.
- Therefore, a sequence of operations to add the contents of register R1 to those of register R2 and store the result in register R3 is

1. $R1_{out}$,Y_{in}
2. $R2_{out}$,SelectY, Add, Z_{in}
3. $Z_{out}R3_{in}$

Step 1: The output of register R1 and the input of register Y are enabled, causing the contents of R1 to be transferred over the bus to Y.

Step 2: The multiplexer's Select signal is set to Select Y, causing the multiplexer to gate the contents of register Y to input A of the ALU. At the same time, the contents of register R2 are gated onto the bus and, hence, to input B. The function performed by the ALU depends on the signals applied to its control lines. In this case, the Add line is set to 1, causing the output of the ALU to be the sum of the two numbers at inputs A and B. This sum is loaded into register Z because its input control signal is activated.

Step 3: The contents of register Z are transferred to the destination register, R3. This last transfer cannot be carried out during step 2, because only one register output can be connected to the bus during any clock cycle.

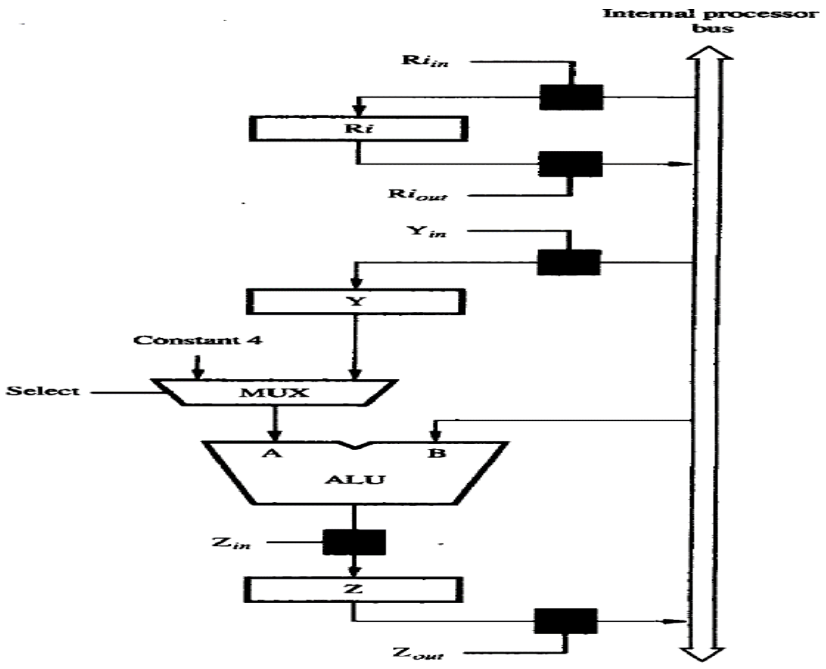


Figure 4.3: Input and Output gating for the registers

4.1.4 Fetching a Word from Memory

- To fetch a word of information from memory, the processor has to specify the address of the memory location where this information is stored and request a Read operation. The connections for register MDR are illustrated in Figure 4.4.
- It has four control signals: MDR_{in} and MDR_{out}, control the connection to the internal bus, and MDR_{inE} and MDR_{outE} control the connection to the external bus.

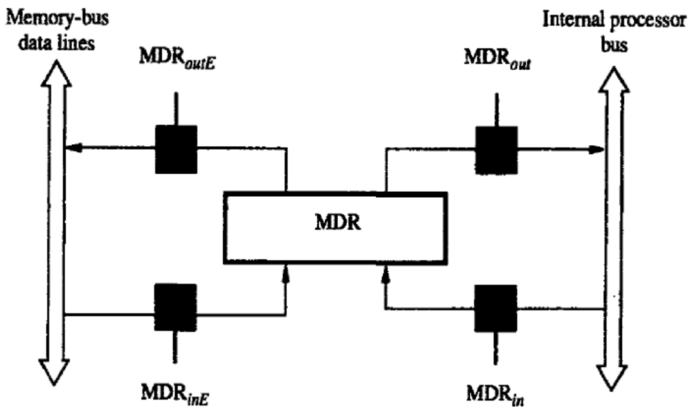


Figure 4.4: Connection and control signals for register MDR

- As an example of a read operation, consider the instruction Move (R1), R2. The actions needed to execute this instruction are:

1. MAR🡐[RI]
2. Start a Read operation on the memory bus
3. Wait for the MFC(Memory Function Completed) response from the memory
4. Load MDR from the memory bus
5. R2 🡐[MDR]

- These actions may be carried out as separate steps, but some can be combined into a single step. Each action can be completed in one clock cycle, except action 3 which requires one or more clock cycles, depending on the speed of the addressed device.
- The memory read operation requires three steps, which can be described by the signals being activated as follows:

1. Rl_{out},MAR_{in}, Read
2. MDR_{inE}, WMFC
3. MDR_{out},$R2_{in}$

where WMFC is the control signal that causes the processor's control circuitry to wait for the arrival of the MFC signal.

4.1.5 Storing a word in Memory

- Writing a word into a memory location follows a similar procedure. The desired address is loaded into MAR. Then, the data to be written are loaded into MDR, and a Write command is issued. Hence, executing the instruction Move R2,(R1) requires the following sequence:

1. $R1_{out}$, MAR_{in}
2. $R2_{out}$, MDR_{in}, Write
3. MDR_{outE}, WMFC

- As in the case of the read operation, the Write control signal causes the memory bus interface hardware to issue a Write command on the memory bus. The processor remains in step 3 until the memory operation is completed and an MFC response is received.

4.2 Execution of a complete instruction

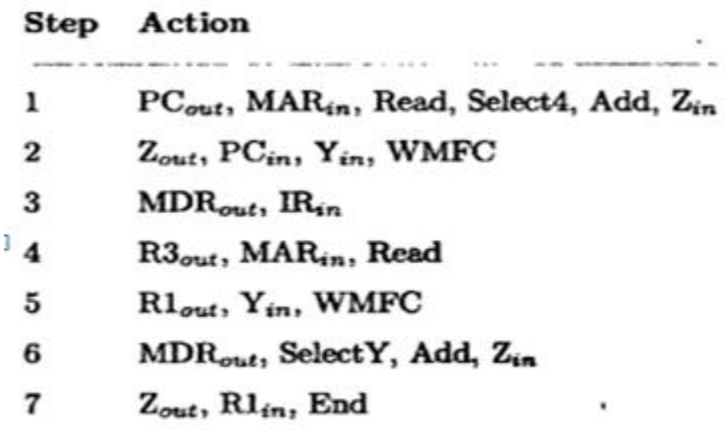
4.2.1 Executing an Instruction

- 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 byR3).
  3. Perform the addition.
  4. Load the result into RI.
  5. Instruction execution proceeds as follows.

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.

Step 2: The updated value is moved from register Z back into the PC, while waiting for the memory to respond.

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.)



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

Figure 4.5: Control Sequence for Execution of the instruction Add (R3), R1

Step 4: The instruction decoding circuit interprets the contents of the IR. 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 step4,and a memory read operation is initiated.

Step 5: the contents of R1 are transferred to register Y, to prepare for the addition operation.

Step 6: When the Read operation is completed, the memory operand is available in register MDR, and the addition operation is performed. 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.

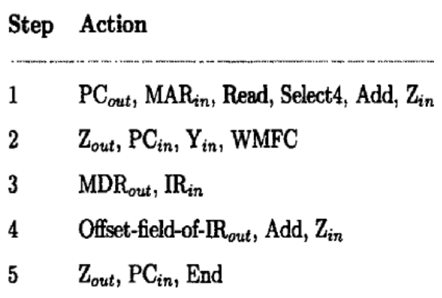
Step 7: The sum is stored in register Z, and then transferred to R1. The End signal causes a new instruction fetch cycle to begin by returning to step 1.

- This discussion accounts for all control signals, except Y in step2. There is no need to copy the updated contents of PC into register Y when executing the Add instruction.
- But, in Branch instructions the updated value of the PC is needed to compute the Branch target address.
- To speed up the execution of Branch instructions, this value is copied into register Y in step 2. Since step 2 is part of the fetch phase, the same action will be performed for all instructions. This does not cause any harm because register Y is not used for any other purpose at that time.

4.2.2 Branch Instruction

- A branch instruction replaces the contents of the PC with the branch target address. This address is usually obtained by adding an offset X, which is given in the branch instruction, to the updated value of the PC. Figure gives a control sequence that implements an unconditional branch instruction. Processing starts, as usual, with the fetch phase. This phase ends when the instruction is loaded into the IR in step 3.
- The offset value is extracted from the IR by the instruction decoding circuit, which will also perform sign extension if required. Since the value of the updated PC is already available in register Y, the offset X is gated onto the bus in step 4, and an addition operation is performed. The result, which is the branch target address, is loaded into the PC in step 5.

- The offset X used in a branch instruction is usually the difference between the branch target address and the address immediately following the branch instruction.



Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 4.6: Control Sequence of Unconditional Branch

- For example: if the branch instruction is at location 2000 and if the branch target address is 2050, the value of X must be 46. The PC is incremented during the fetch phase before knowing the type of the instruction being executed. Thus, when the branch address is computed in step 4, the PC value uses the updated value, which points to the instruction following the branch instruction in the memory.

4.3 Multiple Bus Organization

- We used the simple single-bus structure to illustrate the basic ideas. The resulting control sequences are quite long because only one data item can be transferred over the bus in a clock cycle.
- To reduce the number of steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel. Figure 4.7 depicts a three-bus structure used to connect the registers and the ALU of a processor.
- The register file is said to have three ports. There are two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. The third port allows the data on bus C to be loaded into a third register during the same clock cycle.
- Buses A and B are used to transfer the source operands to the A and B inputs of the ALU, where an arithmetic or logic operation may be performed. The result is transferred to the destination over bus C. If needed, the ALU may simply pass one of its two input operands unmodified to bus C. We will call the ALU control signals for such an operation R=A or

R=B. The three-bus arrangement obviates the need for registers Y and Z.

- A second feature in Figure 4.7 is the introduction of the Incrementer unit, which is used to increment the PC by 4. Using the Incrementer eliminates the need to add 4 to the PC using the main ALU. The source for the constant 4 at the ALU input multiplexer is still useful. It can be used to increment other addresses, such as the memory addresses in Load Multiple and Store Multiple instructions.

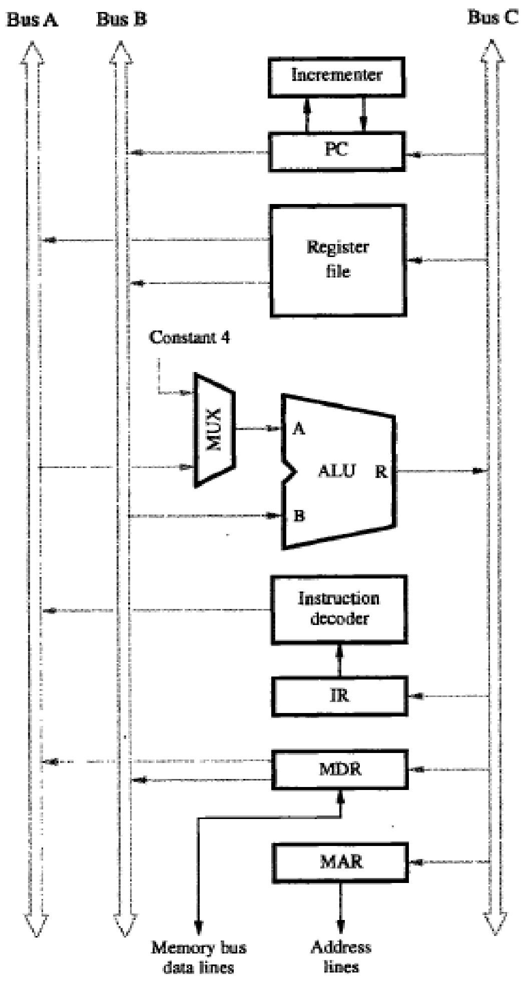
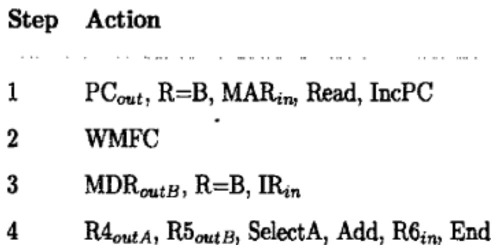


Figure 4.7: Three Bus organization of the datapath

- Consider the three-operand instruction

Add R4, R5, R6

The control sequence for executing this instruction is given as below



Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Figure 4.8: Control Sequence for the instruction Add R4, R5, R6

Step 1: the contents of the PC are passed through the ALU, using the R=B control signal, and loaded into the MAR to start a memory read operation. At the same time the PC is incremented by 4. Note that the value loaded into MAR is the original contents of the PC. The incremented value is loaded into the PC at the end of the clock cycle and will not affect the contents of MAR.

Step2: the processor waits for MFC and loads the data received into MDR.

.*Step 3:* Transfers the data received in MDR to IR.

Step4: The execution phase of the instruction requires only one control step to complete.

By providing more paths for data transfer a significant reduction in the number of clock cycles needed to execute an instruction is achieved.

4.4 Hardwired Control and Micro programmed Control

4.4.1 Hardwired Control

- To execute instructions, the processor must generate the control signals in proper sequence.
- There are two basic approaches: hardwired control and microprogrammed control.
- Consider the sequence of control signals given in Figure 4.5. Each step in this sequence is completed in one clock period. A counter may be used to keep track of the control steps, as shown in Figure 4.9. Each state or count of this counter corresponds to one control step.
- The required control signals are determined by the following information:
  - Contents of the control step counter
  - Contents of the instruction register

  - Contents of condition code flags
  - External input signals, such as MFC interrupt requests

- The decoder/encoder block in Figure 4.9 is a combinational circuit that generates the required control outputs depending on the state of all its inputs.
- By separating the decoding and encoding functions, we obtain more detailed diagram shown in Figure 4.10. The step decoder provides a separate signal line for each step, or time slot, in the control sequence.
- Similarly the output of the instruction decoder consists of a separate line for each machine instruction.

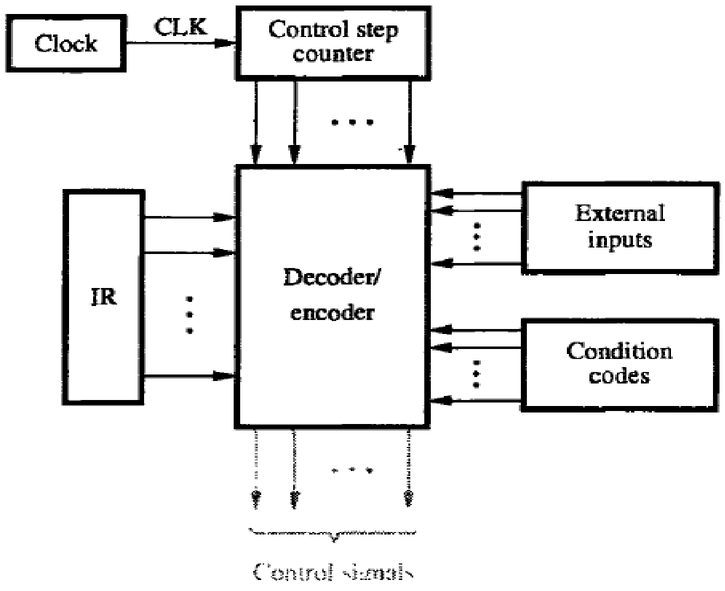


Figure 4.9 Control Unit Organization

- For any instruction loaded in the IR, one of the output lines INS_1 through INS_m is set to 1, and all other lines are set to 0.
- The input signals to the encoder block are combined to generate the individual control signals Y_{in}, PC_{out}, Add, End and so on.
- An example of how encoder generates the Zin control signal for the processor organization in Fig 4.1 is given in Fig 4.11. This circuit implements the logic function

$$Zin = T_1 + T_6 . ADD + T_4 . BR + \ldots\ldots$$

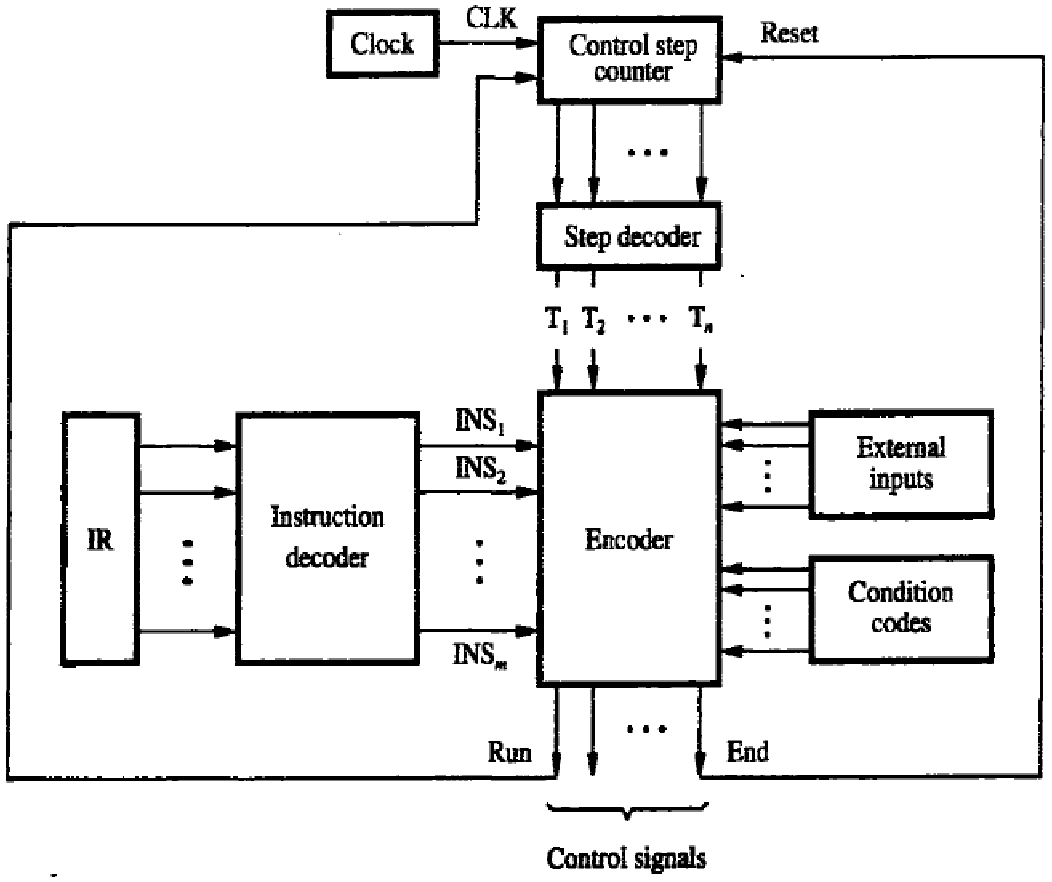


Figure 4.10 Separation of the decoding and encoding functions

- This signal is asserted during time slot T_1 for all instructions, during T_6 for an ADD instruction, during T_4 for an unconditional branch instruction, and so on.

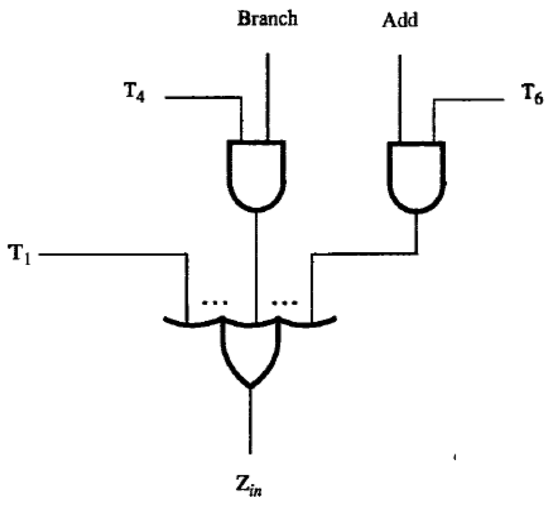


Fig 4.11 Generation of the Z_{in} control signal for the processor

- The logic function for Zin is derived from the control sequences in Figures 4.5 and 4.6.
- As another example Fig 4.12 gives a circuit that generates the END control signal from the logic function

  END= T_7 . ADD + T_5 . BR + (T_5.N + T_4. $\overline{N}$) . BRN + ……
- The END signal starts a new instruction cycle by resetting the control step counter to its starting value.
- Figure 4.10 contains another control signal called RUN. When set to 1, RUN causes the counter to be incremented by one at the end of every clock cycle. When RUN is 0, the counter stops counting. This is needed whenever the WMFC signal is issued, to cause the processoe to wait for the reply from the memory.

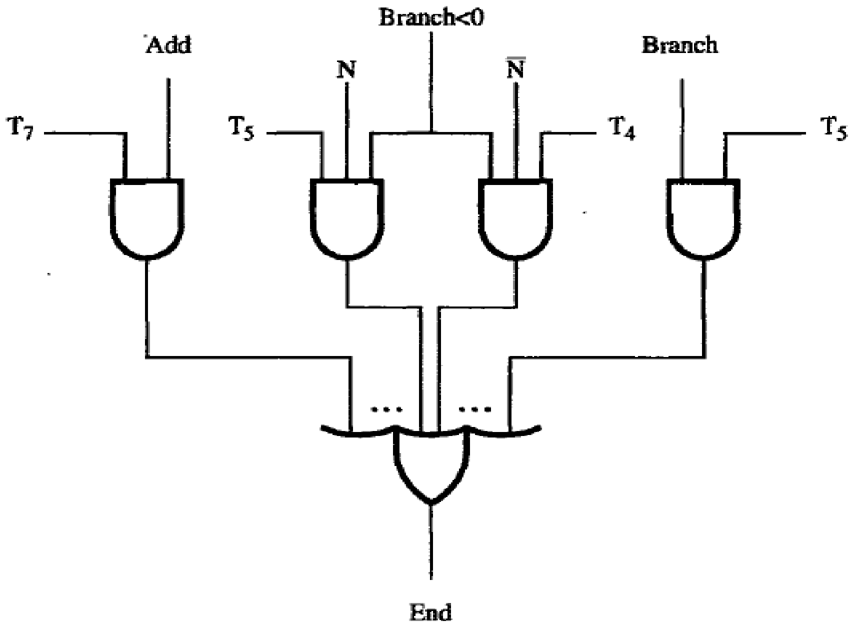


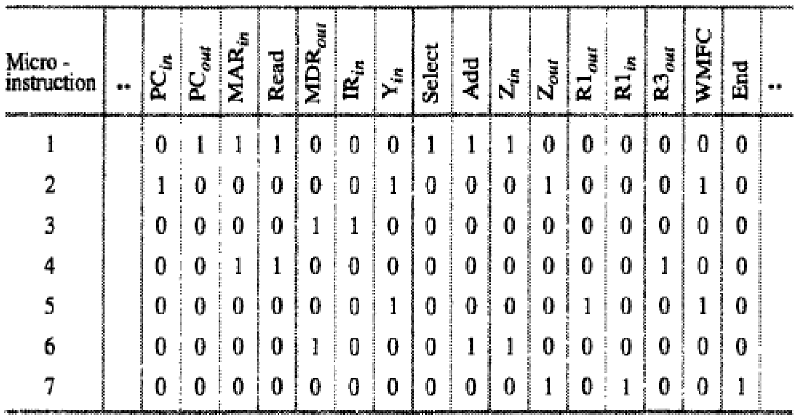
Fig 4.12 Generation of END Control Signal.

4.4.2 Micro programmed Control

- In micro programmed control, control signals are generated by a program similar to machine language programs.
- Some common terms are:
- *Control Word (CW):* It is a word whose individual bits represent various control signals in Fig

4.10. Each of the control steps in the control sequence of an instruction defines a unique combination of 1s and 0s in the CW.

- The CWs corresponding to the 7 steps of Figure 4.5 are shown in Figure 4.13. We have assumed that SelectY 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 *Microroutine* for that instruction and the individual control words in this microroutine are referred to as *microinstructions.*



Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Fig 4.13: An example of microinstructions for Fig 4.5

- 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.
- 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 in to the µPC. The µPC is then automatically incremented by clock, causing successive micro instructions to be read from the control store. Hence, the control signals are delivered to various parts of the processor in the correct sequence.
- One important function of the control unit can not be implemented by the simple organization in Fig 4.14. This is the situation that arises when the control unit is required to check the

status of the condition codes or the external inputs to choose between alternative courses of action.

- In the case of hardwired control, this situation is handled by including an appropriate logic function in the encoder circuitry. In Microprogrammed control, an alternative approach is to use conditional branch instructions. In addition to branch address, these micro instructions specify which of the external inputs, condition codes, or possibly, bits of the instruction register should be checked as a condition for branching to take place.

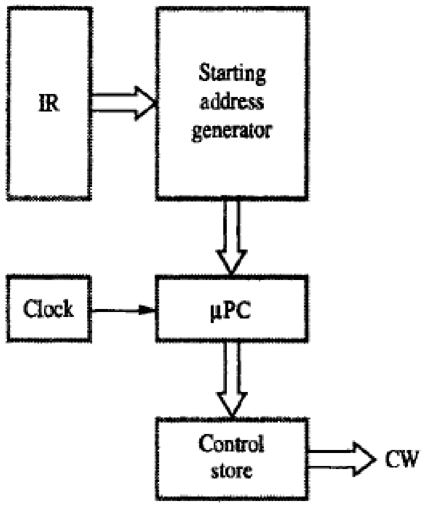
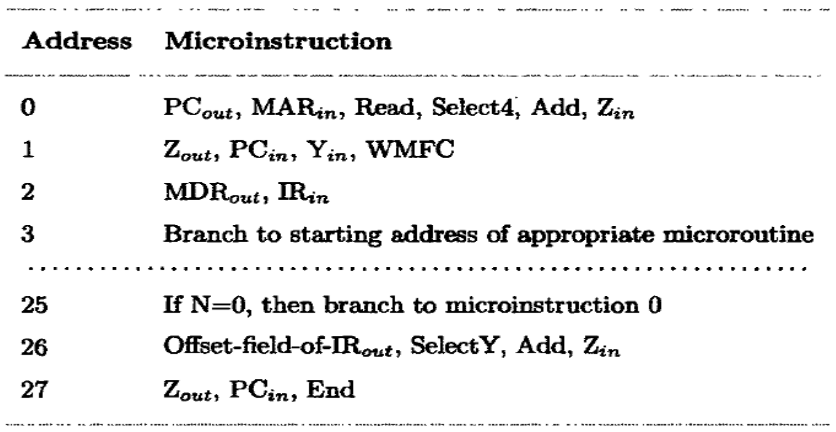


Fig 4.14 Basic organization of a micro programmed control

- The instruction Branch < 0 will implemented by a micro routine shown in Figure 4.15.



Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
....	
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

Fig 4.15 Micro Routine for the instruction Branch < 0

- After loading this instruction to IR, a branch instruction, transfers control to the corresponding micro routine, which is assumed to start at location 25 in the control store. This address is the output of the starting address generator block in Figure 4.14.
- The micro instruction at location 25 tests the N bit of the condition codes. If this bit is equal to 0 to fetch new machine instruction. Otherwise, the micro instruction at location 26 is executed to put the branch target address into register Z. The micro instruction in location 27 loads this address into PC.

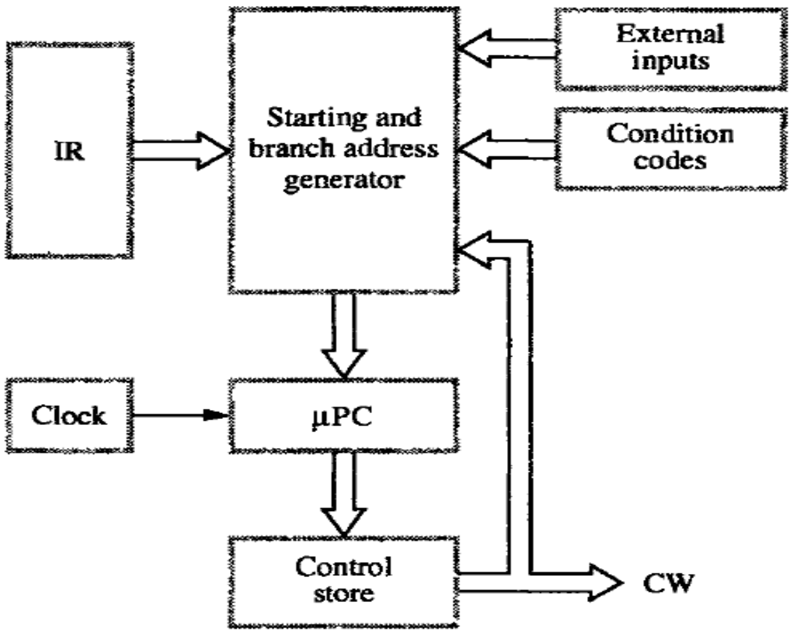


Fig 4.16 Organization of the control unit to allow conditional branching in the microprogram.

- To support micro program branching, the organization of the control unit should be modified shown in Figure 4.16.
- The starting address generator block of Figure 4.14 becomes the starting and branch address generator. This block loads a new into the µPC when a microinstruction instructs it to do so.
- To allow implementation a conditional branch, inputs to this block consist of the external inputs and a codes as well as the contents of the instruction register.
- In this control unit, the µPC is incremented every time a new microinstruction is fetched from the microg memory, except in the following situations:

1. When a new instruction is loaded into the IR, the µPC is loaded with the address of the

micro routine for that instruction.

2. When a Branch microinstruction is encountered and the branch condition is satisfied, the µPC is loaded with the branch address.

3. When an End microinstruction is encountered, the µPC is loaded with the a of the first CW in the micro routine for the instruction fetch cycle (this address is 0 in Figure 4.15).

Memory Organization

4.5 Basic Concepts

- The maximum size of the memory that can be used in any computer is determined by the addressing scheme. For example, a computer that generates 16-bit addresses is capable of addressing up to 2^{16}= 64K (kilo) memory locations. Machines whose instructions generate 32-bit addresses can utilize a memory that contains up to 2^{32}= 4G (giga) locations, whereas machines with 64-bit addresses can access up to 2^{64}= 16E (exa) ≈ 16 × 10^{18} locations. The number of locations represents the size of the address space of the computer.
- The memory is usually designed to store and retrieve data in word-length quantities. The connection between the processor and its memory consists of address, data, and control lines, as shown in Figure 4.17.
- The processor uses the address lines to specify the memory location involved in a data transfer operation, and uses the data lines to transfer the data. At the same time, the control lines carry the command indicating a Read or a Write operation and whether a byte or a word is to be transferred. The control lines also provide the necessary timing information and are used by the memory to indicate when it has completed the requested operation.
- Data transfer between the memory and the processor takes place through the use of two processor registers, usually called MAR (memory address register) and MDR (memory data register). If MAR is k bits long and MDR is n bits long, then the memory unit may contain up to 2^k addressable locations.
- During a memory cycle, n bits of data are transferred between the memory and the processor. This transfer takes place over the processor bus, which has k address lines and n data lines. The bus also includes the control lines Read/$\overline{Write}$ (R/$\overline{W}$) and Memory Function Completed

(MFC) for coordinating data transfers. Other control lines may be added to indicate the number of bytes to be transferred. The connection between the processor and the memory is shown schematically in Figure 4.17.

- The processor reads data from the memory by loading the address of the required memory location into the MAR register and setting the R/$\overline{W}$ line to 1. The memory responds by placing the data from the addressed location onto the data lines, and confirms this action by asserting the MFC signal. Upon receipt of the MFC signal, the processor loads the data on the data lines into the MIDR register.
- The processor writes data into a memory location by loading the address of this location into MAR and loading the data into MDR. It indicates that a write operation is involved by setting the R/$\overline{W}$ line to 0.
- If read or write operations involve consecutive address locations in the main mem-ory, then a "block transfer" operation can be performed in which the only address sent to the memory is the one that identifies the first location.

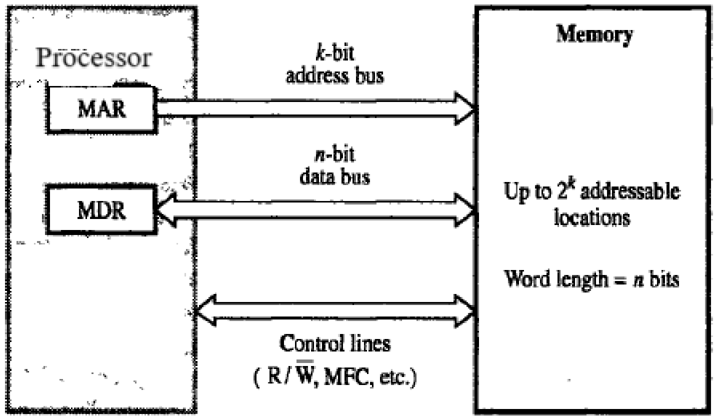

Processor
MAR
MDR
k-bit address bus
n-bit data bus
Control lines (R / $\overline{W}$, MFC, etc.)
Memory
Up to 2^k addressable locations
Word length = n bits

Figure 4.17: Connection of the memory to the processor.

- A useful measure of the speed of memory units is the time that elapses between the initiation of an operation to transfer a word of data and the completion of that operation. This is referred to as the *memory access time.*

- Another important measure is the *memory cycle time,* which is the minimum time delay required between the initiation of two successive memory operations, for example, the time between two successive Read operations. The cycle time is usually slightly longer than the access time, depending on the implementation details of the memory unit.
- A memory unit is called a *Random-Access Memory* (RAM) if the access time to any location is the same, independent of the location's address. This distinguishes such memory units from serial, or partly serial, access storage devices such as magnetic and optical disks. Access time of the latter devices depends on the address or position of the data. The technology for implementing computer memories uses semiconductor integrated circuits.

4.5.1 Cache and Virtual Memory

- The processor of a computer can usually process instructions and data faster than they can be fetched from the main memory. Hence, the memory access time is the bottleneck in the system. One way to reduce the memory access time is to use a *cache memory.* This is a small, fast memory inserted between the larger, slower main memory and the processor. It holds the currently active portions of a program and their data.
- *Virtual memory* is another important concept related to memory organization. With this technique, only the active portions of a program are stored in the main memory, and the remainder is stored on the much larger secondary storage device. Sections of the program are transferred back and forth between the main memory and the secondary storage device in a manner that is transparent to the application program. As a result, the application program sees a memory that is much larger than the computer's physical main memory.

4.6 Semiconductor RAM Memories:

- Semiconductor random-access memories (RAMs) are available in a wide range of speeds. Their cycle times range from 100 ns to less than 10 ns. In this section, we discuss the way that memory cells are organized inside a chip.

4.6.1 Internal Organization of Memory Chips:

- Memory cells are usually organized in the form of an array, in which each cell is capable of storing one bit of information. A possible organization is illustrated in Figure 4.18 below. Each row of cells constitutes a memory word, and all cells of a row are connected to a common line referred to as the *word line,* which is driven by the address decoder on the chip.

- The cells in each column are connected to a Sense/Write circuit by two *bit lines*, and the Sense/Write circuits are connected to the data input/output lines of the chip.
- During a Read operation, these circuits sense, or read, the information stored in the cells selected by a word line and place this information on the output data lines.
- During a Write operation, the Sense/Write circuits receive input data and store them in the cells of the selected word.
- Figure 4.18 is an example of a very small memory circuit consisting of 16 words of 8 bits each i.e; a 16 × 8 organization. A single bidirectional data line, Two control lines, R/W and CS, are provided. The R/$\overline{W}$ (Read/$\overline{Write}$) input specifies the required operation, and the CS (Chip Select) input selects a given chip in a multichip memory system.
- The memory circuit in Figure 4.18 stores 128 bits and requires 14 external connections for address, data, and control lines. It also needs two lines for power supply and ground connections. Consider now a slightly larger memory circuit, one that has 1K (1024) memory cells shown in figure 14. This circuit can be organized as a 128 × 8 memory, requiring a total of 19 external connections.
- Alternatively, the same number of cells can be organized into a 1K×1 format. In this case, a 10-bit address is needed, but there is only one data line, resulting in 15 external connections.
  Figure 4.19 shows such an organization.

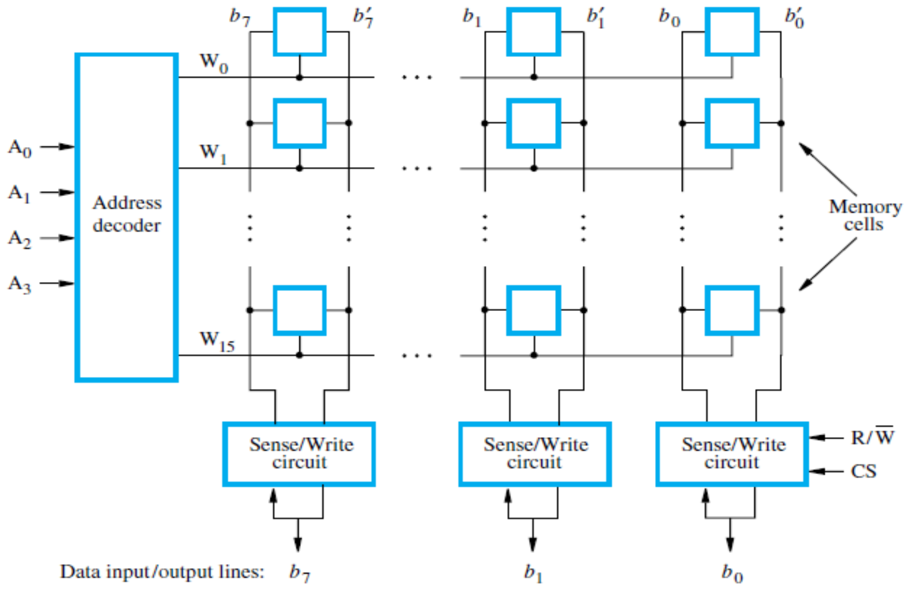


Figure 4.18**:** Organization of bit cells in a memory chip.

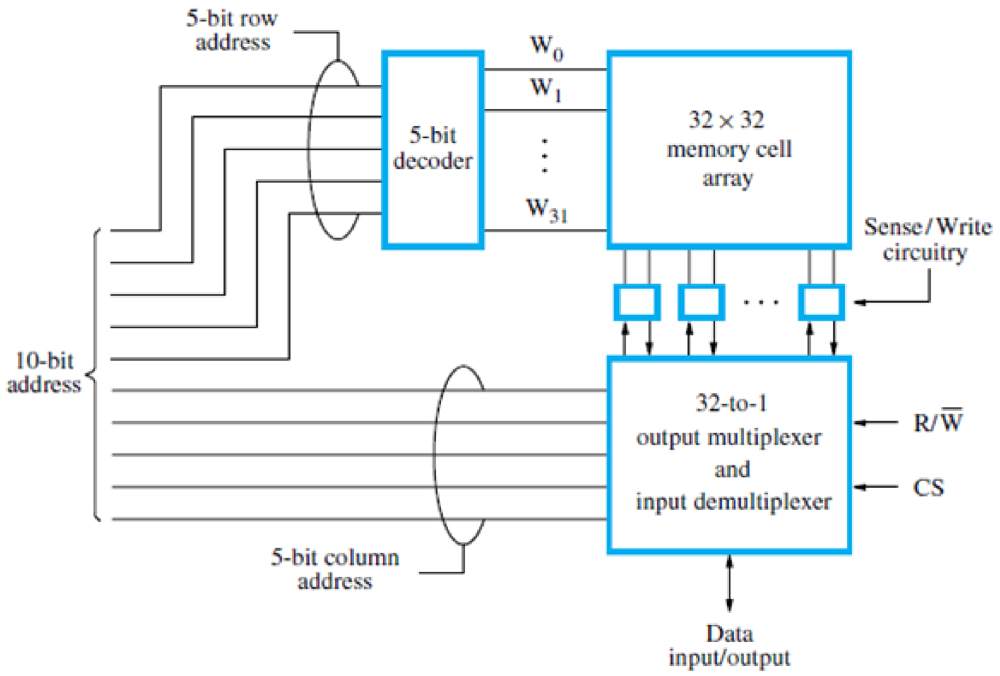


Figure 4.19: Organization of a 1K ×1 memory chip

- The required 10-bit address is divided into two groups of 5 bits each to form the row and column addresses for the cell array. A row address selects a row of 32 cells, all of which are accessed in parallel. But, only one of these cells is connected to the external data line, based on the column address by the output Mutliplexer and input demultiplexer.
- Large chips have essentially the same organization as Figure 4.18 and Figure 4.19, but use a larger memory cell array and have more external connections. For example, a 4M-bit chip may have a 512K × 8 organization, in which case a 19 address is needed and 8 data input/output pins are needed.

4.6.2 Static Memories

- Memories that consist of circuits capable of retaining their state as long as power is applied are known as *static memories*. Figure 4.20 illustrates how a *static RAM* (SRAM) cell may be implemented.

- Two inverters are cross-connected to form a latch. The latch is connected to two bit lines by transistors $T1$ and $T2$. These transistors act as switches that can be opened or closed under control of the word line.
- When the word line is at ground level, the transistors are turned off and the latch retains its state. For example, if the logic value at point X is 1 and at point Y is 0, this state is maintained as long as the signal on the word line is at ground level. Assume that this state represents the value 1.

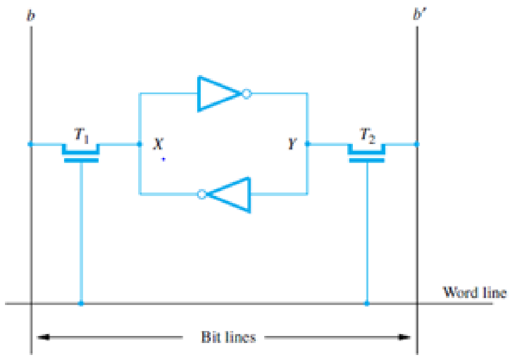


Figure 4.20: A static RAM cell

4.6.2.1 Read Operation

- In order to read the state of the SRAM cell, the word line is activated to close switches $T1$ and $T2$. If the cell is in state 1, the signal on bit line b is high and the signal on bit line b' is low.
- The opposite is true if the cell is in state 0. Thus, b and $b'$are always complements of each other. The Sense/Write circuit at the end of the two bit lines monitors their stateand sets the corresponding output accordingly.

4.6.2.2 Write Operation

- During a Write operation, the Sense/Write circuit drives bit lines b and b', instead of sensing their state. It places the appropriate value on bit line b and its complement on $b'$and activates the word line.

- This forces the cell into the corresponding state, which the cell retains when the word line is deactivated.

4.6.3 Asynchoronous Dynamic RAMs

- Static RAMs are fast, but their cells require several transistors. Less expensive and higher density RAMs can be implemented with simpler cells. But, these simpler cells do not retain their state for a long period, unless they are accessed frequently for Read or Write operations. Memories that use such cells are called *dynamic RAMs* (DRAMs).
- Information is stored in a dynamic memory cell in the form of a charge on a capacitor, but this charge can be maintained for only tens of milliseconds. Since the cell is required to store information for a much longer time, its contents must be periodically *refreshed* by restoring the capacitor charge to its full value. This occurs when the contents of the cell are read or when new information is written into it.
- An example of a dynamic memory cell that consists of a capacitor, *C*, and a transistor, *T*, is shown in Figure 4.21. To store information in this cell, transistor *T* is turned on and an appropriate voltage is applied to the bit line.
- This causes a known amount of charge to be stored in the capacitor. After the transistor is turned off, the charge remains stored in the capacitor, but not for long. The capacitor begins to discharge. This is because the transistor continues to conduct a tiny amount of current, measured in pico amperes, after it is turned off.
- Hence, the information stored in the cell can be retrieved correctly only if it is read before the charge in the capacitor drops below some threshold value. During a Read operation, the transistor in a selected cell is turned on.

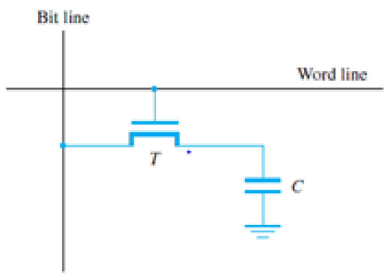


Figure 4.21: A single transistor dynamic memory cell

- A sense amplifier connected to the bit line detects whether the charge stored in the capacitor is above or below the threshold value. If the charge is above the threshold, the sense amplifier drives the bit line to the full voltage representing the logic value 1. As a result, the capacitor is recharged to the full charge corresponding to the logic. value 1.
- If the sense amplifier detects that the charge in the capacitor is below the threshold value, it pulls the bit line to ground level to discharge the capacitor fully. Thus, reading the contents of a cell automatically refreshes its contents. Since the word line is common to all cells in a row, all cells in a selected row are read and refreshed at the same time.
- A 16 -Megabit DRAM chip, configured as 2M × 8, is shown in Figure 4.22 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, forming 512 bytes of data. Therefore, 12 address bits are needed to select a row, and another 9 bits are needed to specify a group of 8 bits in the selected row. In total, 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 the row and column addresses of a byte, respectively.
- During a Read or a Write operation, the row address is applied first. It is loaded into the row address latch in response to a signal pulse on an input control line called the Row Address Strobe (RAS). This causes a Read operation to be initiated, in which all cells in the selected row are read and refreshed.

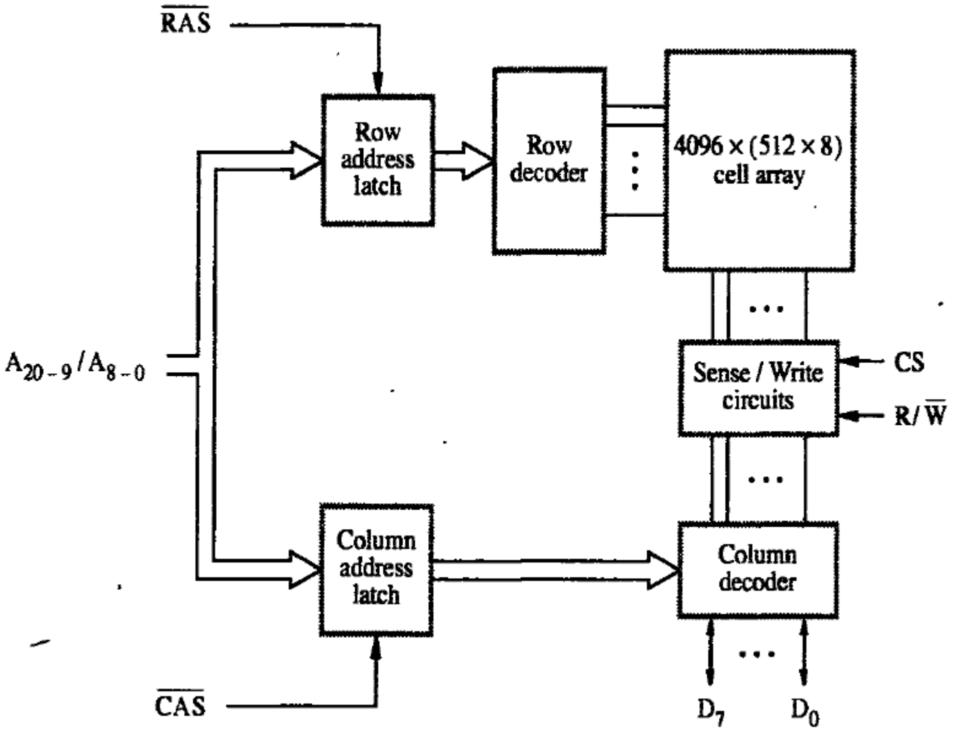


Figure 4.22: Internal organization of a 2M × 8 dynamic memory chip

- Shortly after the row address is loaded, the column address is applied to the address pins and loaded into the column address latch under control of a second control line called the Column Address Strobe (CAS). The information in this latch is decoded and the appropriate group of 8 Sense/Write circuits is selected.
- If the $R/\overline{W}$ control signal indicates a Read operation, the output values of the selected circuits are transferred to the data lines, D_{7-0}.
- For a Write operation, the information on the D_{7-0} lines is transferred to the selected circuits, then used to overwrite the contents of the selected cells in the corresponding 8 columns. We should note that in commercial DRAM chips, the RAS and CAS control signals are active when low. Hence, addresses are latched when these signals change from high to low. The signals are shown in diagrams as RAS and CAS to indicate this fact.
- The timing of the operation of the DRAM described above is controlled by the RAS and CAS

signals. These signals are generated by a memory controller circuit external to the chip when the processor issues a Read or a Write command. During a Read operation, the output data are transferred to the processor after a delay equivalent to the memory's access time. Such memories are referred to as *asynchronous DRAMs*.

- A block of data can be transferred at a much faster rate than can be achieved for transfers involving random addresses. The block transfer capability is referred to as the *fast page mode* feature. (A large block of data is often called a page.) The faster rate attainable in the fast page mode makes dynamic RAMs particularly well suited to this environment.

4.6.4 Synchronous DRAMs:

- More recent developments in memory technology have resulted in DRAMs whose operation is directly synchronized with a clock signal. Such memories are known as synchronous DRAMs (SDRAMs). Figure 4.22 indicates the structure of an SDRAM. The cell array is the same as in asynchronous DRAMs. The address and data connections are buffered by means of registers. We should particularly note that the output of each sense amplifier is connected to a latch.
- A Read operation causes the contents of all cells in the selected row to be loaded into these latches. But, if an access is made for refreshing purposes only, it will not change the contents of these latches; it will merely refresh the contents of the cells. Data held in the latches that correspond to the selected column(s) are transferred into the data output register, thus becoming available on the data output pins.
- SDRAMs have several different modes of operation, which can be selected by writing control information into a mode register. For example, burst operations of different lengths can be specified. The burst operations use the block transfer capability as the fast page mode feature. In SDRAMs, it is not necessary to provide externally generated pulses on the CAS line to select

successive columns. The necessary control signals are provided internally using a column counter and the clock signal. New data can be placed on the data lines in each clock cycle. All actions are triggered by the rising edge of the clock.

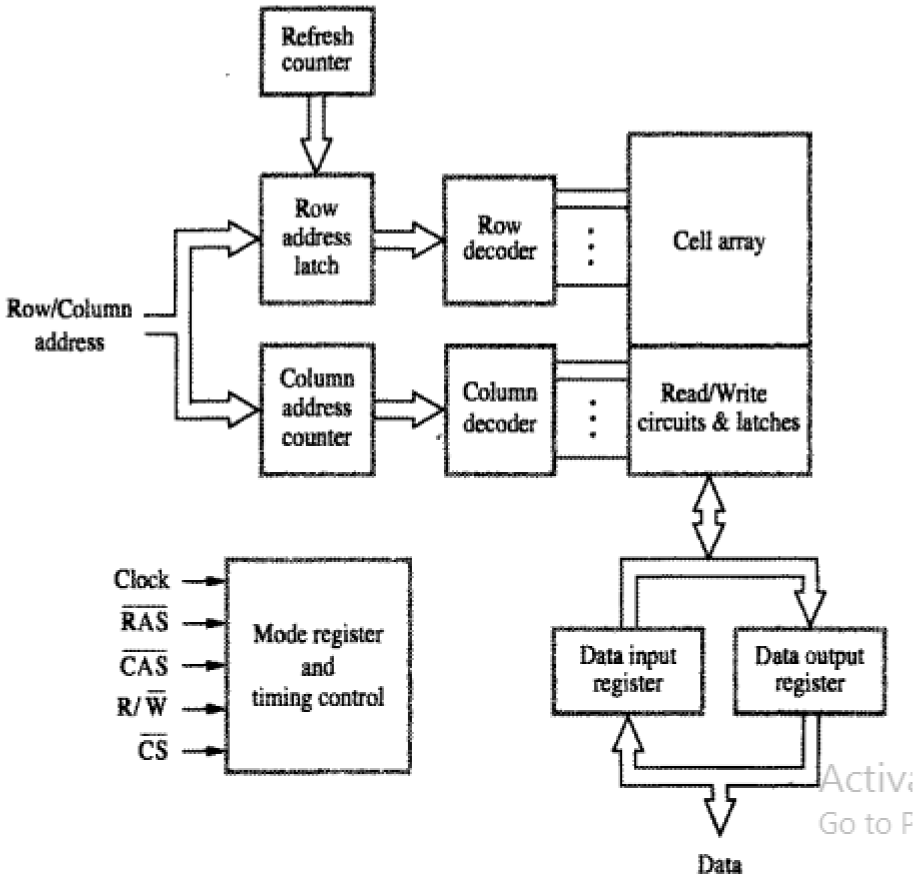


Figure 4.23 Synchronous DRAM

- SDRAMs have built-in refresh circuitry. A part of this circuitry is a refresh counter, which provides the addresses of the rows that are selected for refreshing. In a typical SDRAM, each row must be refreshed at least every 64 ms.
- The term *memory latency* is used to refer to the amount of time it takes to transfer a word of data to or from the memory. In the case of reading or writing a single word of data, the latency provides a complete indication of memory performance.

- When transferring blocks of data, it is of interest to know how much time is needed to transfer an entire block. Since blocks can be variable in size, it is useful to define a performance measure in terms of the number of bits or bytes that can be transferred in one second. This measure is often referred to as the *memory bandwidth*.
- A similar memory device is available, which accesses the cell array in the same way, but transfers data on both edges of the clock. The latency of these devices is the same as for standard SDRAMs. But, since they transfer data on both edges of the clock, their bandwidth is essentially doubled for long burst transfers. Such devices are known as double-data-rate SDRAMs (DDR SDRAMS).

4.7 Read-only Memories

- Both static and dynamic RAM chips are volatile, which means that they retain information only while power is turned on. There are many applications requiring memory devices that retain the stored information when power is turned off. Many embedded applications do not use a hard disk and require nonvolatile memories to store their software.
- A special writing process is needed to place the information into a nonvolatile memory. Since its normal operation involves only reading the stored data, a memory of this type is called a *read-only memory* (ROM). Figure 4.24 shows a possible configuration of a ROM cell.

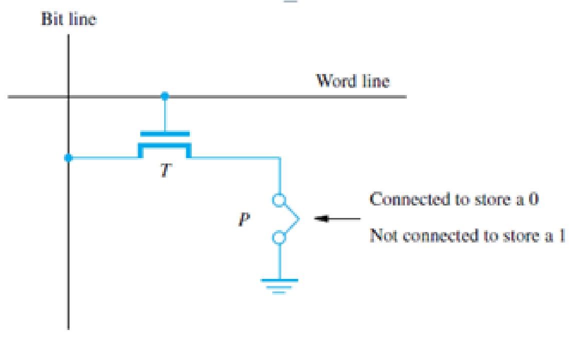


Figure 4.24 A ROM cell

- 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 to close the transistor switch. As a result, 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 level, indicating a 1. A sense circuit at the end of the bit line generates the proper output value. The state of the connection to ground in each cell is determined when the chip is manufactured, using a mask with a pattern that represents the information to be stored.

4.7.1 PROM

- Some ROM designs allow the data to be loaded by the user, thus providing a *programmable ROM* (PROM). Programmability is achieved by inserting a fuse at point *P* Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses at these locations using high-current pulses. Of course, this process is irreversible.
- PROMs provide flexibility and convenience not available with ROMs. The cost of preparing the masks needed for storing a particular information pattern makes ROMs cost effective only in large volumes. The alternative technology of PROMs provides a more convenient and considerably less expensive approach, because memory chips can be programmed directly by the user

4.7.2 EPROM

- Another type of ROM chip provides an even higher level of convenience. It allows the stored data to be erased and new data to be written into it. Such an *erasable,* reprogrammable ROM is usually called an *EPROM*. It provides considerable flexibility during the development phase of digital systems. Since EPROMs are capable of retaining stored information for a long time, they can be used in place of ROMs or PROMs while software is being developed. In this way, memory changes and updates can be easily made.

- An EPROM cell has a structure similar to the ROM cell in Figure 4.24. However, the connection to ground at point *P* is made through a special transistor. The transistor is normally turned off, creating an open switch. It can be turned on by injecting charge into it that becomes trapped inside. Thus, an EPROM cell can be used to construct a memory in the same way as the previously discussed ROM cell. Erasure requires dissipating the charge trapped in the transistors that form the memory cells. This can be done by exposing the chip to ultraviolet light, which erases the entire contents of the chip. To make this possible, EPROM chips are mounted in packages that have transparent windows.

4.7.3 EEPROM

- An EPROM must be physically removed from the circuit for reprogramming. Also, the stored information cannot be erased selectively. The entire contents of the chip are erased when exposed to ultraviolet light. Another type of erasable PROM can be programmed, erased, and reprogrammed electrically. Such a chip is called an *electrically erasable* PROM, or EEPROM. It does not have to be removed for erasure.
- Moreover, it is possible to erase the cell contents selectively. One disadvantage of EEPROMs is that different voltages are needed for erasing, writing, and reading the stored data, which increases circuit complexity. However, this disadvantage is outweighed by the many advantages of EEPROMs. They have replaced EPROMs in practice.

4.7.4 Flash Memory

- An approach similar to EEPROM technology has given rise to *flash memory* devices. A flash cell is based on a single transistor controlled by trapped charge, much like an EEPROM cell. Also like an EEPROM, it is possible to read the contents of a single cell. The key difference is that, in a flash device, it is only possible to write an entire block of cells. Prior to writing, the previous contents of the block are erased. Flash devices have greater density, which leads to higher capacity and a lower cost per bit. They require a single power supply voltage, and consume less power in their operation.

- The low power consumption of flash memories makes them attractive for use in portable, battery-powered equipment. Typical applications include hand-held computers, cell phones, digital cameras, and MP3 music players. In hand-held computers and cell phones, a flash memory holds the software needed to operate the equipment, thus obviating the need for a disk drive. A flash memory is used in digital cameras to store picture data. In MP3 players, flash memories store the data that represent sound. Cell phones, digital cameras, and MP3 players are good examples of embedded systems.
- Single flash chips may not provide sufficient storage capacity for the applications mentioned above. Larger memory modules consisting of a number of chips are used where needed. There are two popular choices for the implementation of such modules: flash cards and flash drives.

4.8 Speed, size and cost

- As an ideal memory would be fast, large, and inexpensive. We know that a very fast memory can be implemented if SRAM chips are used. But these chips are expensive. Thus, for cost reasons, it is impractical to build a large memory using SRAM chips. The alternative is to use Dynamic RAM chips, which have much simpler basic cells and thus are much less expensive. But such memories are significantly slower.
- Although dynamic memory units in the range of hundreds of megabytes can be implemented at a

reasonable cost, the affordable size is still small compared to the demands of large programs with voluminous data. A solution is provided by using secondary storage, mainly magnetic disks, to implement large memory spaces. Very large disks are available at a reasonable price, and they are used extensively in computer systems. However, they are much slower than the semiconductor memory units. So we conclude the following: A huge amount of cost-effective storage can be provided by magnetic disks. A large, yet affordable, main memory can be built with dynamic RAM technology. This leaves SRAMs to be used in smaller units where speed is of the essence, such as in cache memories.

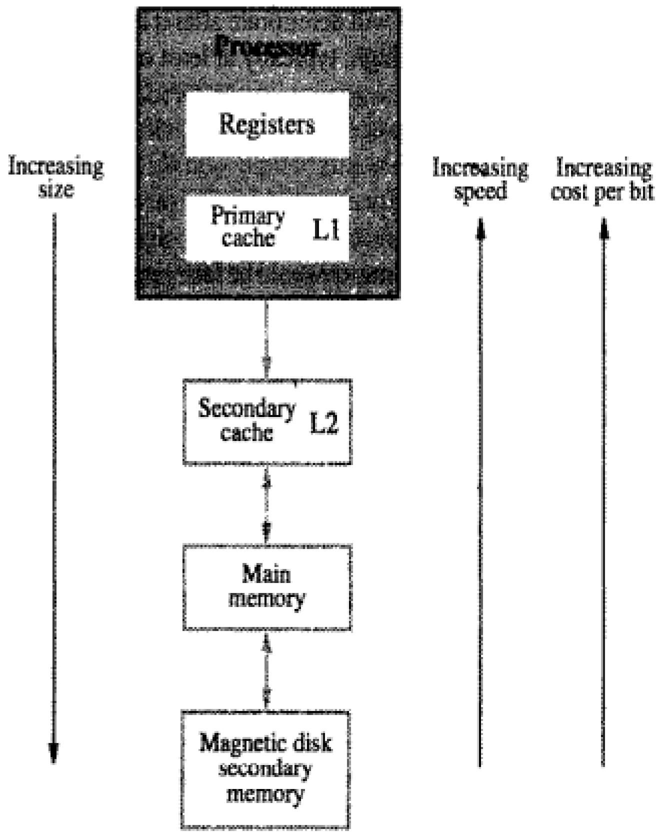


Figure 4.25: Memory hierarchy

- All of these different types of memory units are employed effectively in a computer. The entire computer memory can be viewed as the hierarchy depicted in Figure below. The fastest access is to data held in processor registers. Therefore, if we consider the registers to be part of the memory hierarchy, then the processor registers are at the top in terms of the speed of access. Of course, the registers provide only a minuscule portion of the required memory.

- At the next level of the hierarchy is a relatively small amount of memory that can be implemented directly on the processor chip. This memory, called a processor cache, holds copies of instructions and data stored in a much larger memory that is provided externally. There are often two levels of caches. A primary cache is always located on the processor chip. This cache is small because it competes for space on the processor chip, which must implement many other functions. The primary cache is referred to as level 1 (L1) cache. A larger, secondary cache is placed between the primary cache and the rest of the memory. It is referred to as level 2 (L2) cache. It is usually implemented using SRAM chips.
- Including a primary cache on the processor chip and using a larger, off-chip, secondary cache is currently the most common way of designing computers. However, other arrangements can be found in practice. It is possible not to have a cache on the processor chip at all. Also, it is possible to have both L1 and L2 caches on the processor chip.
- The next level in hierarchy is called main memory. Thus rather large memory is implemented using dynamic memory components typically in the form of SIMMs, DIMMs, or RIMMs. The main memory is much larger but significantly slower than the cache memory. In a typical computer, the access time for the main memory is about ten times longer than the access time for the L1 cache.
- Disk devices provide a huge amount of inexpensive storage. They are very slow compared to the semiconductor devices used to implement the main memory. During program execution, the speed of memory access is of utmost importance. The key to managing the operation of the

hierarchical memory system is to bring the instructions and data that will be used in the near future as close to the processor as possible.

4.9 Cache Memories

- The cache is a small and very fast memory, interposed between the processor and the main memory. Its purpose is to make the main memory appear to the processor to be much faster than it actually is. The effectiveness of this approach is based on a property of computer programs called *locality of reference*.
- Analysis of programs shows that most of their execution time is spent in routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other. The point is that many instructions in localized areas of the program are executed repeatedly during some time period. This behavior manifests itself in two ways: temporal and spatial.

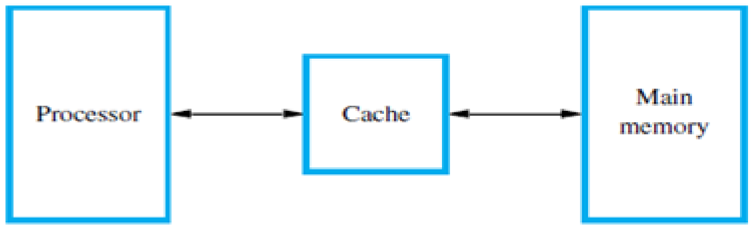


Figure 4.26 Use of Cache memory

- Consider the arrangement in Figure 4.13. When the processor issues a Read request, the contents of a block of memory words containing the location specified are transferred into the cache. Subsequently, when the program references any of the locations in this block, the desired contents are read directly from the cache. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
- The correspondence between the main memory blocks and those in the cache is specified by a

mapping function. When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the cache's *replacement algorithm*.

replacement algorithm.

Cache Hits

- The cache control circuitry determines whether the requested word that is requested by the processor currently exists in the cache. If it does, the Read or Write operation is performed on the appropriate cache location. In this case, a *read* or *write hit* is said to have occurred.
- For a Write operation, the system can proceed in one of two ways. In the first technique, called the *write-through* protocol, both the cache location and the main memory location are updated.
- The second technique is to update only the cache location and to mark the block containing it with an associated flag bit, often called the *dirty* or *modified bit*. The main memory location of the word is updated later, when the block containing this marked word is removed from the cache to make room for a new block. This technique is known as the *write-back*, or *copy-back*, protocol.

Cache Misses

- A Read operation for a word that is not in the cache constitutes a *Read miss*. It causes the block of words containing the requested word to be copied from the main memory into the cache. After the entire block is loaded into the cache, the particular word requested is forwarded to the processor. Alternatively, this word may be sent to the processor as soon as it is read from the main memory. The latter approach, which is called *load-through*, or *early restart*, reduces the processor's waiting time somewhat, at the expense of more complex circuitry.

- When a *Write miss* occurs in a computer that uses the write-through protocol, the information is written directly into the main memory. For the write-back protocol, the block containing the addressed word is first brought into the cache, and then the desired word in the cache is overwritten with the new information.

4.9.1 Mapping Functions

- There are several possible methods for determining where memory blocks are placed in the cache. It is instructive to describe these methods using a specific small example.
- Consider a cache consisting of 128 blocks of 16 words each, for a total of 2048 (2K) words, and assume that the main memory is addressable by a 16-bit address. The main memory has 64K words, which we will view as 4K blocks of 16 words each. For simplicity, we have assumed that consecutive addresses refer to consecutive words.

4.9.1.1 Direct Mapping

- The simplest way to determine cache locations in which to store memory blocks is the *direct-mapping* technique. In this technique, block *j* of the main memory maps onto block *j* modulo 128 of the cache, as depicted in Figure 4.27. Thus, whenever one of the main memory blocks 0, 128, 256, . . . is loaded into the cache, it is stored in cache block 0. Blocks 1, 129, 257, . . . are stored in cache block 1, and so on. Since more than one memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full.
- For example, instructions of a program may start in block 1 and continue in block 129, possibly after a branch. As this program is executed, both of these blocks must be transferred to the block-1 position in the cache. Contention is resolved by allowing the new block to overwrite the currently resident block.

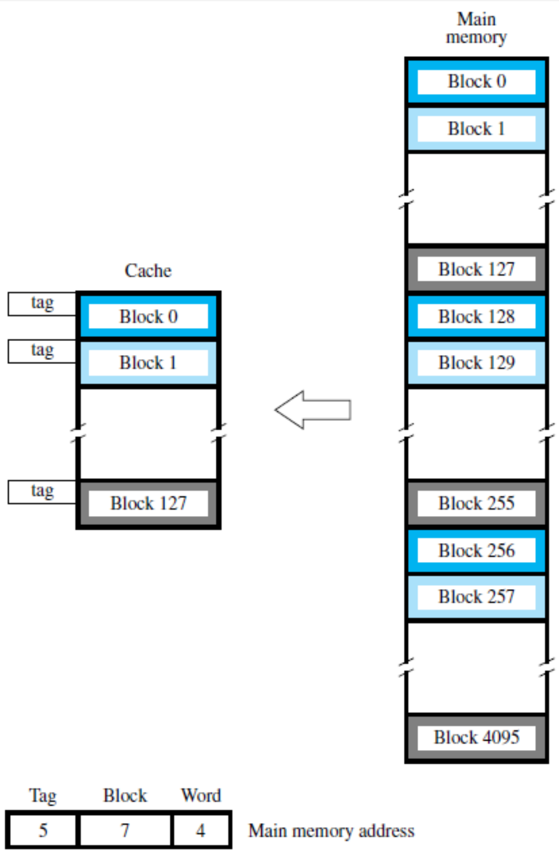


Figure 4.27: Direct-mapped cache

4.9.1.2 Associative Mapping

- Figure 4.28 shows the most flexible mapping method, in which a main memory block can be placed into any cache block position. In this case, 12 tag bits are required to identify a memory block when it is resident in the cache. The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present. This is called the *associative-mapping* technique.
- When a new block is brought into the cache, it replaces (ejects)an existing block only if the cache is full. In this case, we need an algorithm to select the block to be replaced.
- The complexity of an associative cache is higher than that of a direct-mapped cache, because of

the need to search all 128 tag patterns to determine whether a given block is in the cache. To avoid a long delay, the tags must be searched in parallel. A search of this kind is called an *associative search*.

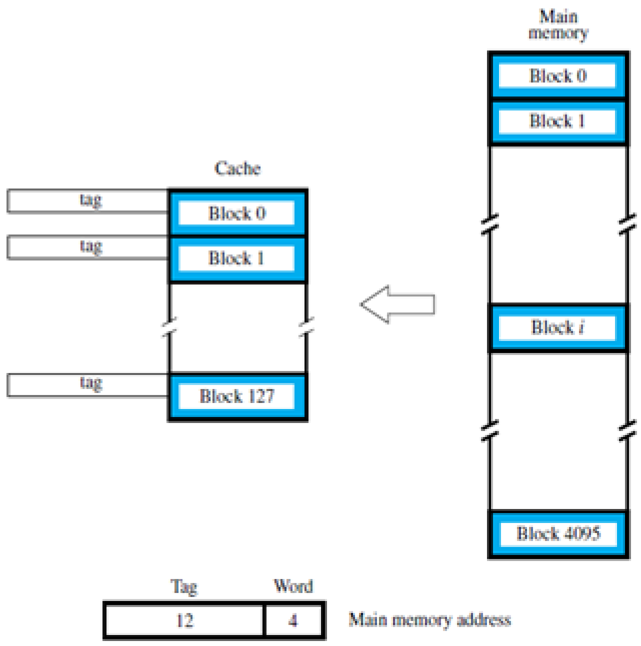


Figure 4.28: Associative-mapped cache.

4.9.1.3 Set-Associative Mapping

- Another approach is to use a combination of the direct- and associative-mapping techniques. The blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set. Hence, the contention problem of the direct method is eased by having a few choices for block placement.
- At the same time, the hardware cost is reduced by decreasing the size of the associative search. An example of this *set-associative-mapping* technique is shown in Figure 8.18 for a cache with two blocks per set. In this case, memory blocks 0, 64, 128, . . . , 4032 map into cache set 0, and they can occupy either of the two block positions within this set. Having 64 sets means that the 6-bit set field of the address determines which set of the cache might contain the desired block.

- The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present. This two-way associative search is simple to implement.
- The number of blocks per set is a parameter that can be selected to suit the requirements of a particular computer. For the main memory and cache sizes in Figure 8.18, four blocks per set can be accommodated by a 5-bit set field, eight blocks per set by a 4-bit set field, and so on.
- The extreme condition of 128 blocks per set requires no set bits and corresponds to the fully-associative technique, with 12 tag bits. The other extreme of one block per set is the direct-mapping method. A cache that has *k* blocks per set is referred to as a *k*-way set-associative cache.

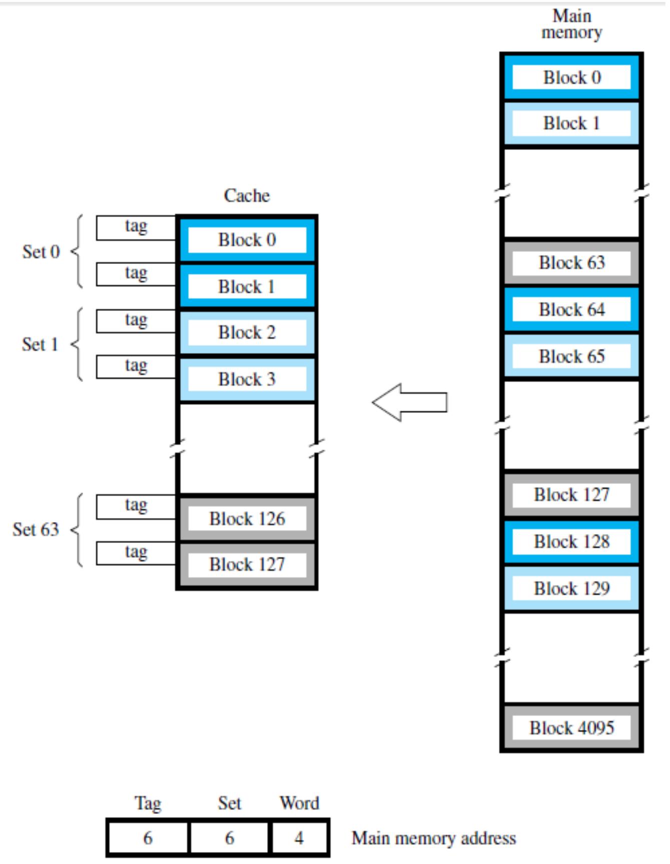


Fig 4.29 Set-associative-mapped cache with two blocks per set.

4.9.2 Replacement Algorithms

- In a direct-mapped cache, the position of each block is predetermined by its address; hence, the replacement strategy is trivial. In associative and set-associative caches there exists some flexibility. When a new block is to be brought into the cache and all the positions that bit may occupy are full, the cache controller must decide which of the old blocks to overwrite. This is an important issue, because the decision can be a strong determining factor in system performance.
- In general, the objective is to keep blocks in the cache that are likely to be referenced in the near future. But, it is not easy to determine which blocks are about to be referenced. The property of locality of reference in programs gives a clue to a reasonable strategy. Because program execution usually stays in localized areas for reasonable periods of time, there is a high probability that the blocks that have been referenced recently will be referenced again soon. Therefore, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced. This block is called the *least recently used* (LRU) block, and the technique is called the *LRU replacement algorithm*.
- To use the LRU algorithm, the cache controller must track references to all blocks as computation proceeds. Suppose it is required to track the LRU block of four-block set in a set-associative cache. A 2-bit counter can be used for each block. When a hit occurs, the counter of the block that is referenced is set to 0. Counters with values originally lower than the referenced one are incremented by one, and all others remain unchanged.
- When a *miss* occurs and the set is not full, the counter associated with the new block loaded from the main memory is set to 0, and the values of all other counters are increased by one. When a miss occurs and the set is full, the block with the counter value 3 is removed, the new block is put in its place, and its counter is set to 0. The other three block counters are

incremented by one. It can be easily verified that the counter values of occupied blocks are always distinct.

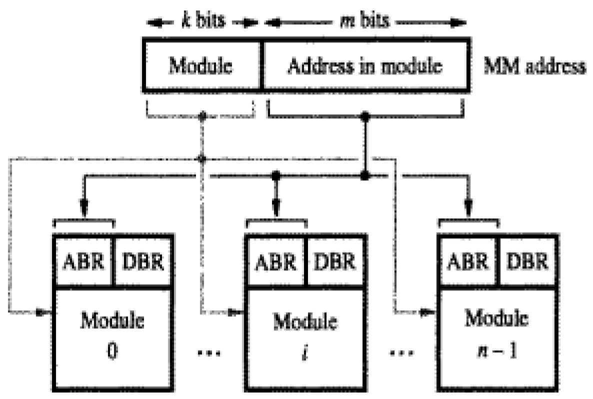
- The simplest algorithm is to randomly choose the block to be overwritten. this simple algorithm has been found to be quite effective in practice.

4.10 Performance Considerations

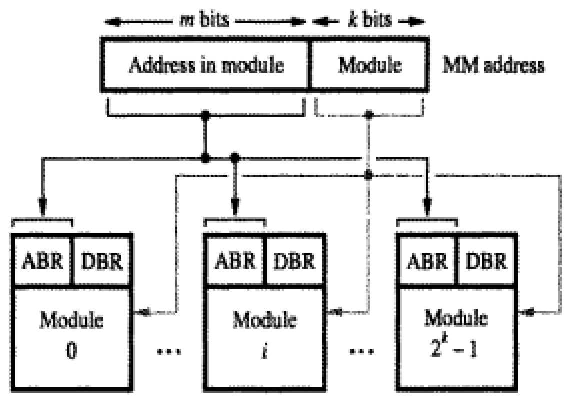
- Two key factors in the commercial success of a computer are performance and cost; the best possible performance for a given cost is the objective. A common measure of success is the *price/performance ratio*. Performance depends on how fast machine instructions can be brought into the processor and how fast they can be executed.
- The main purpose of this hierarchy is to create a memory that the processor sees as having a short access time and a large capacity. When a cache is used, the processor is able to access instructions and data more quickly when the data from the referenced memory locations are in the cache. Therefore, the extent to which caches improve performance is dependent on how frequently the requested instructions and data are found in the cache. In this section, we examine this issue quantitatively.

4.10.1. Interleaving:

- If the main memory of a computer is structured as a collection of physically separate modules, each with its own address buffer register (ABR) and data buffer register (DBR), memory access operations may proceed in more than one module at the same time. Thus, the aggregate rate of transmission of words to and from the main memory system can be increased.
- How individual addresses are distributed over the modules is critical in determining the average number of modules that can be kept busy as computations proceed.
- Two methods of address layout are indicated in Figure 4.30.



(a) Consecutive words in a module



(b) Consecutive words in consecutive modules

Fig 4.30 Addressing multiple-module memory system

- In the first case, the memory address generated by the processor is decoded. The high- order k bits name one of n modules, and the low-order m bits name a particular word in that module. When consecutive locations are accessed, as happens when a block of data is transferred to a cache, only one module is involved. At the same time, however, devices with direct memory access (DMA) ability may be accessing information in other memory modules.
- The second and more effective way to address the modules is shown in Figure. It is called memory interleaving. The low-order k bits of the memory address select a module, and the high-order m bits name a location within that module. In this way, consecutive addresses are located

in successive modules. Thus, any component of the system that generates requests for access to consecutive memory locations can keep several modules busy at any one time. This results in both faster access to a block of data and higher average utilization of the memory system as a whole. To implement the interleaved structure, there must be 24 modules; otherwise, there will be gaps of nonexistent locations in the memory address space.

- The effect of interleaving is substantial. Consider the time needed to transfer a block of data from the main memory to the cache when a read miss occurs. Suppose that a cache with 8-word blocks is used, similar to our examples in Section 5.5. On a read miss, the block that contains the desired word must be copied from the memory into the cache. Assume that the hardware has the following properties. It takes one clock cycle to send an address to the main memory.
- The memory is built with relatively slow DRAM chips that allow the first word to be accessed in 8 cycles, but subsequent words of the block are accessed in 4 clock cycles per word. when consecutive locations in a DRAM are read from a given row of cells, the row address is decoded only once. Addresses of consecutive columns of the array are then applied to access the desired words, which takes only half the time per access.) Also, one clock cycle is needed to send one word to the cache.
- If a single memory module is used, then the time needed to load the desired block into the cache Is 1+8+ (7x4)+1=38 cycles.

4.10.2 Hit Rate and Miss Penalty

- An excellent indicator of the effectiveness of a particular implementation of the memory hierarchy is the success rate in accessing information at various levels of the hierarchy. As a successful access to data in a cache is called a hit. The number of hits stated as a fraction of all attempted accesses is called the *hit rate,* and the *miss rate* is the number of misses stated as a fraction of attempted accesses.

- Ideally, the entire memory hierarchy would appear to the processor as a single memory unit that has the access time of the cache on the processor chip and the size of the magnetic disk. How close we get to this ideal depends largely on the hit rate at different levels of the hierarchy. High hit rates well over 0.9 are essential for high-performance computers.
- Performance is adversely affected by the actions that need to be taken when a miss occurs. A performance penalty is incurred because of the extra time needed to bring a block of data from a slower unit in the memory hierarchy to a faster unit. During that period, the processor is stalled waiting for instructions or data. The waiting time depends on the details of the operation of the cache. For example, it depends on whether or not the load-through approach is used. We refer to the total access time seen by the processor when a miss occurs as the *miss penalty*.
- Consider a system with only one level of cache. In this case, the miss penalty consists almost entirely of the time to access a block of data in the main memory. Let h be the hit rate, M the miss penalty, and C the time to access information in the cache. Thus, the average access time experienced by the processor is

$$t_{avg} = hC + (1 - h)M$$

4.11 Virtual memories:

- In most modern computer systems, the physical main memory is not as large as the address space of the processor. For example, a processor that issues 32-bit addresses has an addressable space of 4G bytes. The size of the main memory in a typical computer with a 32-bit processor may range from 1G to 4G bytes. If a program does not completely fit into the main memory, the parts of it not currently being executed are stored on a secondary storage device, typically a magnetic disk. As these parts are needed for execution, they must first be brought into the main memory, possibly replacing other parts that are already in the memory. These actions are performed automatically by the operating system, using a scheme known as *virtual memory*.

4.11.1 Virtual Memory Organization

- Figure 8.24 shows a typical organization that implements virtual memory.

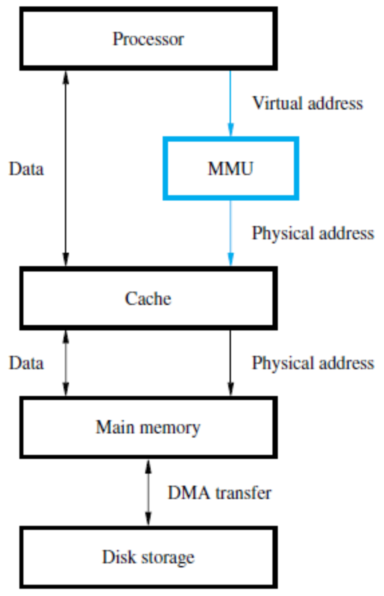


Figure 4.31 Virtual memory organization

- Under a virtual memory system, programs, and hence the processor, reference instructions and data in an address space that is independent of the available physical main memory space.
- The binary addresses that the processor issues for either instructions or data are called *virtual* or *logical addresses*. These addresses are translated into physical addresses by a combination of hardware and software actions.
- If a virtual address refers to a part of the program or data space that is currently in the physical memory, then the contents of the appropriate location in the main memory are accessed immediately. Otherwise, the contents of the referenced address must be brought into a suitable

location in the memory before they can be used.

- A special hardware unit, called the *Memory Management Unit* (MMU), keeps track of which parts of the virtual address space are in the physical memory. When the desired data or instructions are in the main memory, the MMU translates the virtual address into the corresponding physical address.
- If the data are not in the main memory, the MMU causes the operating system to transfer the data from the disk to the memory.

4.11.2 Address Translation

- A simple method for translating virtual addresses into physical addresses is to assume that all programs and data are composed of fixed-length units called *pages*, each of which consists of a block of words that occupy contiguous locations in the main memory.
- Pages commonly range from 2K to 16K bytes in length. They constitute the basic unit of information that is transferred between the main memory and the disk whenever the MMU determines that a transfer is required.
- Pages should not be too small, because the access time of a magnetic disk is much longer (several milliseconds) than the access time of the main memory.
- A virtual-memory address-translation method based on the concept of fixed-length pages is shown schematically in Figure 4.32. Each virtual address generated by the processor, whether it is for an instruction fetch or an operand load/store operation, is interpreted as a *virtual page number* (high-order bits) followed by an *offset* (low-order bits) that specifies the location of a particular byte (or word) within a page.
- Information about the main memory location of each page is kept in a *page table*. This information includes the main memory address where the page is stored and the current status

of the page.

- An area in the main memory that can hold one page is called a *page frame*. The starting address of the page table is kept in a *page table base register*. By adding the virtual page number to the contents of this register, the address of the corresponding entry in the page table is obtained. The contents of this location give the starting address of the page if that page currently resides in the main memory.

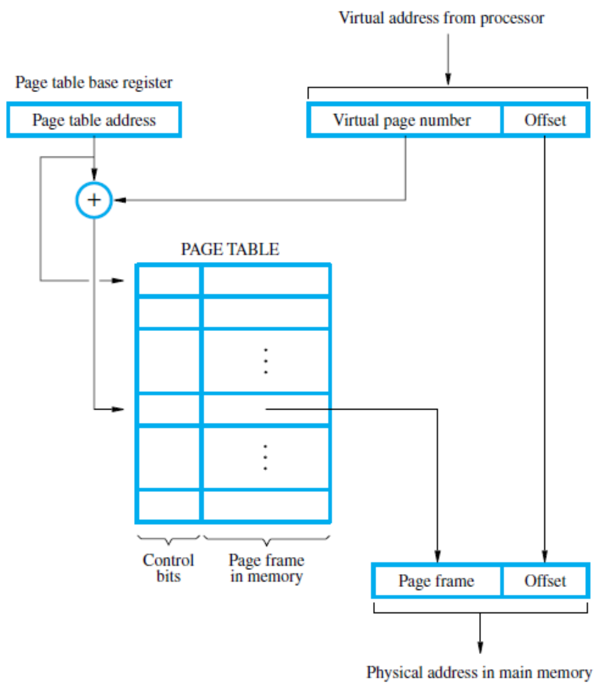


Figure 4.32 Virtual-memory address translation

- Each entry in the page table also includes some control bits that describe the status of the page while it is in the main memory. One bit indicates the validity of the page, that is, whether the page is actually loaded in the main memory. It allows the operating system to invalidate the page without actually removing it.
- Another bit indicates whether the page has been modified during its residency in the memory. As in cache memories, this information is needed to determine whether the page should be written

back to the disk before it is removed from the main memory to make room for another page.

4.11.3 Translation Lookaside Buffer

- The page table information is used by the MMU for every read and write access. Ideally, the page table should be situated within the MMU. Unfortunately, the page table may be rather large. Since the MMU is normally implemented as part of the processor chip, it is impossible to include the complete table within the MMU. Instead, a copy of only a small portion of the table is accommodated within the MMU, and the complete table is kept in the main memory.
- The portion maintained within the MMU consists of the entries corresponding to the most recently accessed pages. They are stored in a small table, usually called the Translation Lookaside Buffer (TLB).
- The TLB functions as a cache for the page table in the main memory. Each entry in the TLB includes a copy of the information in the corresponding entry in the page table. In addition, it includes the virtual address of the page, which is needed to search the TLB for a particular page.
- Figure 4.33 shows a possible organization of a TLB that uses the associative mapping technique. Set-associative mapped TLBs are also found in commercial products. Address translation proceeds as follows. Given a virtual address, the MMU looks in the TLB for the referenced page. If the page table entry for this page is found in the TLB, the physical address is obtained immediately. If there is a miss in the TLB, then the required entry is obtained from the page table in the main memory and the TLB is updated. It is essential to ensure that the contents of the TLB are always the same as the contents of page tables in the memory. When the operating system changes the contents of a page table, it must simultaneously invalidate the corresponding entries in the TLB. One of the control bits in the TLB is provided for this

purpose. When an entry is invalidated, the TLB acquires the new information from the page table in the memory as part of the MMU's normal response to access misses.

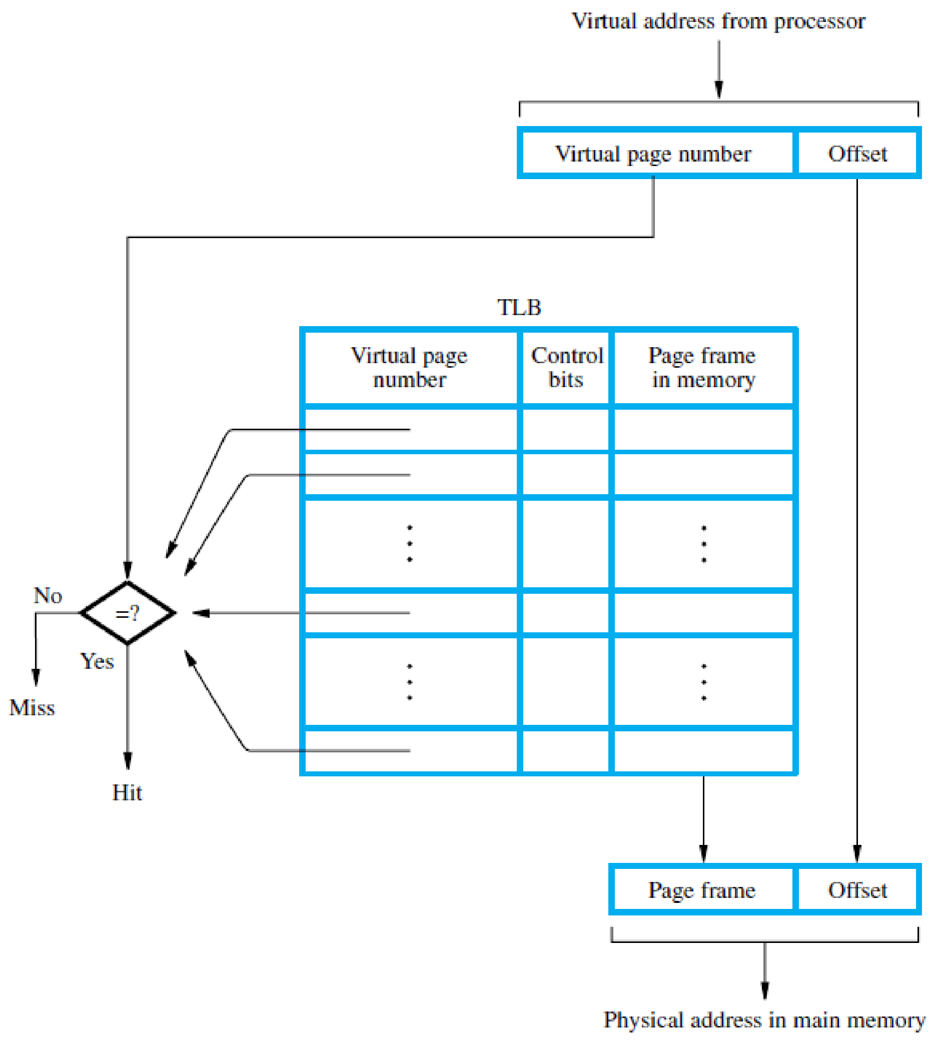


Figure 4.33 Use of an associative-mapped TLB.

4.11.4 Page Faults

- When a program generates an access request to a page that is not in the main memory, a page fault is said to have occurred. The entire page must be brought from the disk into the memory before access can proceed. When it detects a page fault, the MMU asks the operating system to intervene by raising an exception (interrupt). Processing of the program that generated the page fault is interrupted, and control is transferred to the operating system.

- The operating system copies the requested page from the disk into the main memory. Since this process involves a long delay, the operating system may begin execution of another program whose pages are in the main memory. When page transfer is completed, the execution of the interrupted program is resumed.
- When the MMU raises an interrupt to indicate a page fault, the instruction that requested the memory access may have been partially executed. It is essential to ensure that the interrupted program continues correctly when it resumes execution. There are two options. Either the execution of the interrupted instruction continues from the point of interruption, or the instruction must be restarted.
- The design of a particular processor dictates which of these two options is used. If a new page is brought from the disk when the main memory is full, it must replace one of the resident pages. The problem of choosing which page to remove is just as critical here as it is in a cache, and the observation that programs spend most of their time in a few localized areas also applies. Because main memories are considerably larger than cache memories, it should be possible to keep relatively larger portions of a program in the main memory. This reduces the frequency of transfers to and from the disk.
- Concepts similar to the LRU replacement algorithm can be applied to page replacement, and the control bits in the page table entries can be used to record usage history. One simple scheme is based on a control bit that is set to 1 whenever the corresponding page is referenced (accessed). The operating system periodically clears this bit in all page table entries, thus providing a simple way of determining which pages have not been used recently.

4.12 Memory Management Requirements

- Memory management routines are part of the operating system of the computer. It is convenient to assemble the operating system routines into a virtual address space, called the *system space*, that is separate from the virtual space in which user application programs reside. The latter space is called the *user space*. There may be a number of user spaces, one for each user. This is arranged by providing a separate page table for each user program.
- The MMU uses a page table base register to determine the address of the table to be used in the translation process. Hence, by changing the contents of this register, the operating system can switch from one space to another.
- In any computer system in which independent user programs coexist in the main memory, the notion of *protection* must be addressed. No program should be allowed to destroy either the data or instructions of other programs in the memory. The needed protection an be provided in several ways.
- Let us first consider the most basic form of protection. Most processors can operate in one of two modes, the *supervisor mode* and the *user mode*. The processor is usually placed in the supervisor mode when operating system routines are being executed and in the user mode to execute user programs.
- In the user mode, some machine instructions cannot be executed. These are *privileged instructions*. They include instructions that modify the page table base register, which can only be executed while the processor is in the supervisor mode. Since a user program is executed in the user mode, it is prevented from accessing the page tables of other users or of the system space.
- It is sometimes desirable for one application program to have access to certain pages belonging

to another program. The operating system can arrange this by causing these pages to appear in both spaces. The shared pages will therefore have entries in two different pagetables. The control bits in each table entry can be set to control the access privileges granted to each program.

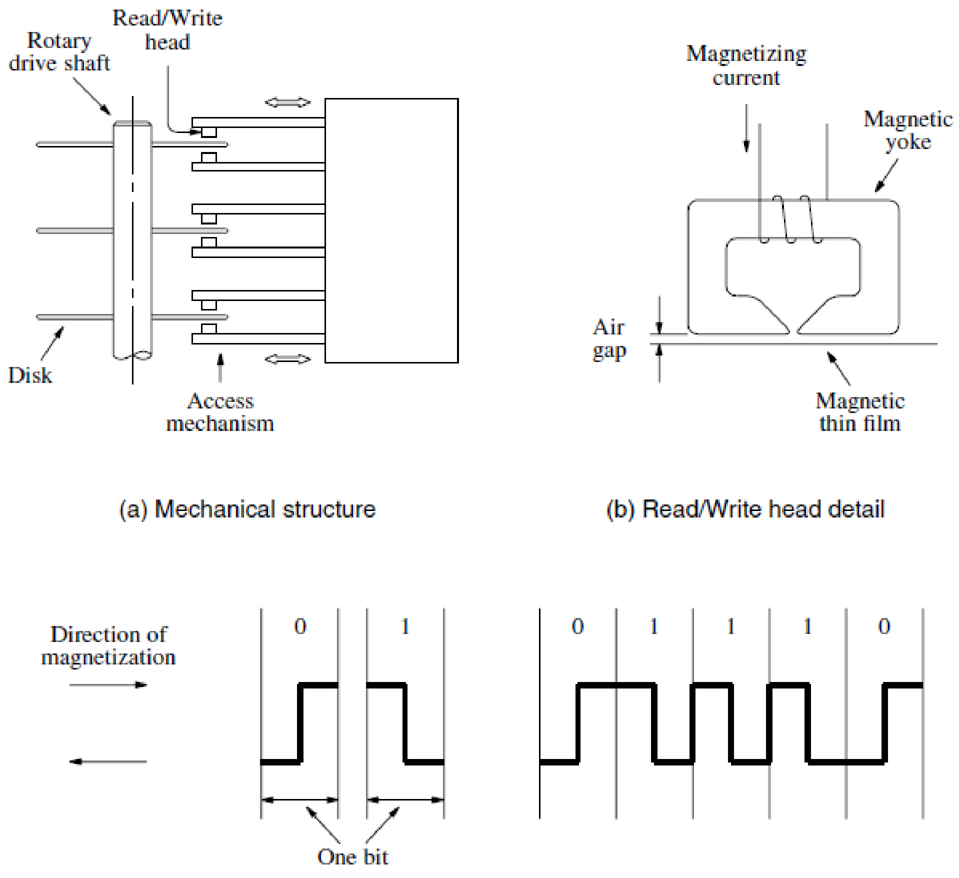
4.13 Secondary memory:

- The large storage requirements of most computer systems are economically realized in the form of magnetic and optical disks, which are usually referred to as secondary storage devices.

4.13.1 Magnetic Hard Disks

- The storage medium in a magnetic-disk system consists of one or more disk platters mounted on a common spindle. A thin magnetic film is deposited on each platter, usually on both sides. The assembly is placed in a drive that causes it to rotate at a constant speed.
- The magnetized surfaces move in close proximity to read/write heads, as shown in Figure 4.34 a. Data are stored on concentric tracks, and the read/write heads move radially to access different tracks.
- Each read/write head consists of a magnetic yoke and a magnetizing coil, as indicated in Figure 4.34b. Digital information can be stored on the magnetic film by applying current pulses of suitable polarity to the magnetizing coil.
- This causes the magnetization of the film in the area immediately underneath the head to switch to a direction parallel to the applied field. The same head can be used for reading the stored information. In this case, changes in the magnetic field in the vicinity of the head caused by the movement of the film relative to the yoke induce a voltage in the coil, which now serves as a sense coil.
- The polarity of this voltage is monitored by the control circuitry to determine the state of magnetization of the film. Only changes in the magnetic field under the head can be sensed during the Read operation. Therefore, if the binary states 0 and 1 are represented by two opposite states of magnetization, a voltage is induced in the head only at 0-to-1 and at 1-to-0 transitions in the bit stream.

- A long string of 0s or 1s causes an induced voltage only at the beginning and end of the string. Therefore, to determine the number of consecutive 0s or 1s stored, a clock must provide information for synchronization.



(a) Mechanical structure (b) Read/Write head detail

(c) Bit representation by phase encoding

Figure 4.34 Magnetic disk principles.

- Read/write heads must be maintained at a very small distance from the moving disk surfaces in order to achieve high bit densities and reliable Read and Write operations. When the disks are moving at their steady rate, air pressure develops between the disk surface and the head and forces the head away from the surface.
- The read/write heads of a disk system are movable. There is one head per surface. All heads are mounted on a comb-like arm that can move radially across the stack of disks to provide access to individual tracks, as shown in Figure 4.34a.
- To read or write data on a given track, the read/write heads must first be positioned over

that track

- The disk system consists of three key parts. One part is the assembly of disk platters, which is usually referred to as the disk.
- The second part comprises the electromechanical mechanism that spins the disk and moves the read/write heads; it is called the disk drive.
- The third part is the disk controller, which is the electronic circuitry that controls the operation of the system.
- The disk controller may be implemented as a separate module, or it may be incorporated into the enclosure that contains the entire disk system. We should note that the term disk is often used to refer to the combined package of the disk drive and the disk it contains.

4.13.1.1 Organization and Accessing of Data on a Disk

- The organization of data on a disk is illustrated in Figure 4.35.
- Each surface is divided into concentric tracks, and each track is divided into sectors. The set of corresponding tracks on all surfaces of a stack of disks forms a logical cylinder.
- All tracks of a cylinder can be accessed without moving the read/write heads. Data are accessed by specifying the surface number, the track number, and the sector number. Read and Write operations always start at sector boundaries.

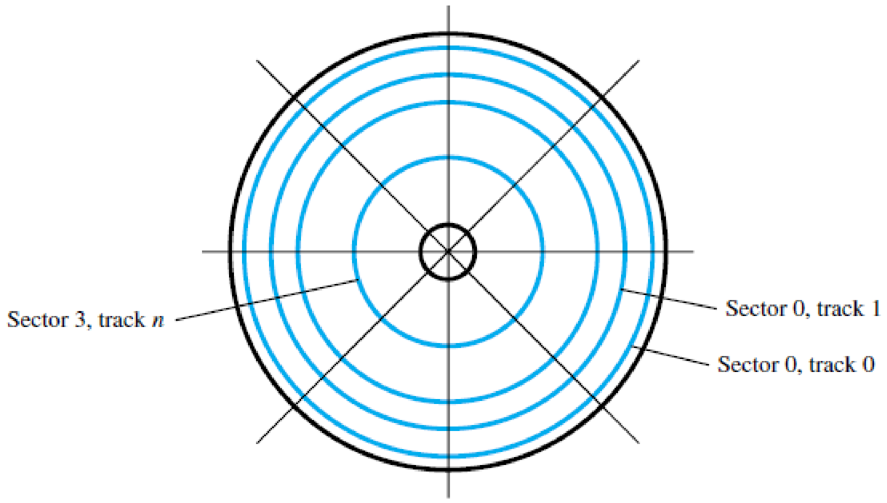


Fig 4.35 Organization of one surface of a disk

- Data bits are stored serially on each track. Each sector may contain 512 or more bytes. The data are preceded by a sector header that contains identification (addressing) information used to find the desired sector on the selected track.
- Following the data, there are additional bits that constitute an error-correcting code (ECC). The ECC bits are used to detect and correct errors that may have occurred in writing or reading the data bytes.
- There is a small inter-sector gap that enables the disk control circuitry to distinguish easily between two consecutive sectors.

4.13.2 Optical Disks

- Storage devices can also be implemented using optical means. The familiar compact disk (CD), used in audio systems, was the first practical application of this technology. Soon after, the optical technology was adapted to the computer environment to provide a high capacity read-only storage medium known as a CD-ROM.
- The first generation of CDs was developed in the mid-1980s by the Sony and Philips companies. The technology exploited the possibility of using a digital representation for analog sound signals.
- To provide high-quality sound recording and reproduction, 16-bit samples of the analog signal are taken at a rate of 44,100 samples per second. Initially, CDs were designed to hold up to 75 minutes, requiring a total of about 3 × 109 bits (3 gigabits) of storage. Since then, higher-capacity devices have been developed.

4.13.2.1 CD ROM

- As the information is stored in binary form in CDs, they are suitable for use as a storage medium in computer systems.
- Stored data are organized on CD-ROM tracks in the form of blocks that are called *sectors*. There are different forms for a sector.
- One format, known as Mode 1, uses 2352- byte sectors. There is a 16-byte header that contains a synchronization field used to detect the beginning of the sector and addressing information used to identify the sector.

- This is followed by 2048 bytes of stored data. At the end of the sector, there are 288 bytes used to implement the error-correcting scheme. The number of sectors per track is variable; there are more sectors on the longer outer tracks. With the Mode 1 format, a CD-ROM has a storage capacity of about 650 Mbytes.
- Error detection and correction is done at more than one level. each byte of information stored on a CD is encoded using a 14-bit code that has some error-correcting capability. This code can correct single-bit errors.
- CD-ROM drives operate at a number of different rotational speeds. The basic speed, known as 1X, is 75 sectors per second. This provides a data rate of 153,600 bytes/s (150 Kbytes/s), using the Mode 1 format.
- Higher speed CD-ROM drives are identified in relation to the basic speed. Thus, a 56X CD-ROM has a data transfer rate that is 56 times that of the 1X CD-ROM, or about 6 Mbytes/s. This transfer rate is considerably lower than the transfer rates of magnetic hard disks, which are in the range of tens of megabytes per second.

4.13.2.2 CD-Rewritable

- The most flexible CDs are those that can be written multiple times by the user. They are known as CD-RWs (CD-ReWritables). The basic structure of CD-RWs is similar to the structure of CD-Rs.
- CD drives designed to read and write CD-RW disks can usually be used with other compact disk media. They can read CD-ROMs and can read and write CD-Rs.
- CD-RW disks provide low-cost storage media. They are suitable for archival storage of information that may range from databases to photographic images. They can be used for low-volume distribution of information, just like CD-Rs, and for backup purposes.
- The CD-RW technology has made CD-Rs less relevant because it offers superior capability at only slightly higher cost.

4.13.2.3 DVD Technology

- The success of CD technology and the continuing quest for greater storage capability has led to the development of DVD (Digital Versatile Disk) technology.

- The first DVD standard as defined in 1996 by a consortium of companies, with the objective of being able to store a full-length movie on one side of a DVD disk. The physical size of a DVD disk is the same as that of CDs. The disk is 1.2 mm thick, and it is 120 mm in diameter. Its storage capacity is made much larger than that of CDs by several design changes.
- Access times for DVD drives are similar to CD drives. However, when the DVD disks rotate at the same speed, the data transfer rates are much higher because of the higher density. Rewritable versions of DVD devices have also been developed, providing large storage capacities.

4.13.3 Magentic tape System

- Magnetic tapes are suited for off-line storage of large amounts of data. They are typically used for backup purposes and for archival storage.
- Magnetic-tape recording uses the same principle as magnetic disks. The main difference is that the magnetic film is deposited on a very thin 0.5- or 0.25-inch wide plastic tape. Seven or nine bits (corresponding to one character) are recorded in parallel across the width of the tape, perpendicular to the direction of motion.
- A separate read/write head is provided for each bit position on the tape, so that all bits of a character can be read or written in parallel. One of the character bits is used as a parity bit.
- To help users organize large amounts of data, a group of related records is called a file. The beginning of a file is identified by a file mark, as shown in Figure 4.36.

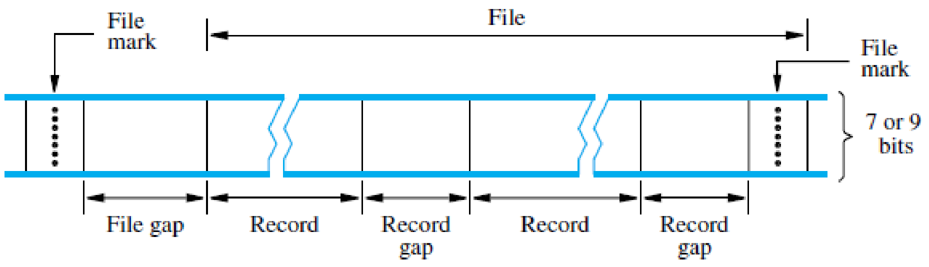


Fig 4.36 organization of data on magnetic tape

- The file mark is a special single- or multiple-character record, usually preceded by a gap longer than the inter-record gap.

- The first record following a file mark can be used as a header or identifier for the file. This allows the user to search a tape containing a large number of files for a particular file.

Assignment-Cum-Tutorial Questions Part– A

Processor Organization

Objective Questions

1. A sequence of microinstructions constitute a_______________ [ ]

 a) Micro-program b)micro-operation

 c) micro-instruction d)microprocessor

2. The instruction,Add#45,R1does______________ [ ]

 a) Adds the value of 45 to the address of R1 and stores 45 in that address

 b) Adds 45 to the value of R1 and stores it in R1

 c) Finds the memory location 45 and adds that content to that of R1

 d) None of the mentioned.

3. A sequence of microinstructions constitutes a______________ [ ]

 a) Micro-program b)micro-operation

 c) micro-instruction d)micro-processor

4. A memory that is part of control unit is referred to as________ [ ]

 a) Cache memory b)control memory

 c) main memory d) virtual memory

5. Whenthecontrolsignalsaregeneratedbyhardwareusingconventionallogicdesign techniques, the control unit is said to be_________________________ [ ]

 a) programmed b)micro-programmed

 c) hardwired d)none of the above

6. The next address generator is also called_____________ [ ]

 a) Micro-program sequencer b)control unit

 c)micro-instruction sequencer d)micro-programmed control

7. The goals of both hard wired control and micro-programmed control units are to______
   a) Access memory   b)access ALU   [ ]
   c) none   d)generate control signals
8. Each computer instruction has its own____________in control memory to generate the micro-operations that execute the instruction.   [ ]
   a) microinstruction   b)branch instruction
   c)mapping logic   d)micro-program routine
9. The control_________register specifies the address of the microinstruction and the control _______Register holds the microinstruction read from memory.   [ ]
   a) Address, data   b)data, memory
   c)memory, instruction   d) data, instruction
10. A control memory has 4096words of 24bitseach.Howmanybits are there in the control address register?   [ ]
    a) 12   b)24   c) 32   d)25
11. Themicroprogramwrittenasstringsof0'sand 1'sisa_______________________[ ]
    a) Symbolic microinstruction   b)binary microinstruction
    c)symbolic micro-program   d)binary micro-program
12. Address information in the microinstruction ***cannot*** be_________   [ ]
    a) Single address field   b)two address field
    c)variable format   d)a control signal
13. Micro instruction execution is____________   [ ]
    a) To generate the control signals needed to execute the microinstruction
    b) To get the next microinstruction from the control memory
    c) To get the next instruction from the main memory
    d) To get the microinstruction from the main memory
14. Microinstruction sequencing is_________   [ ]
    a) To get the next instruction from the main memory
    b) To get the next microinstruction from the control memory
    c) To get the microinstruction from the main memory
    d) To generate the control signals needed to execute the microinstruction
15. A micro-program sequencer

a) Generates the address of next microinstruction to be executed

b) Generates the control signals to execute a microinstruction.

c) Sequentially averages all microinstructions in the control memory.

d) Enables the efficient handling of a micro-program subroutine.

Descriptive Questions

S.No.	Name of the Question	BL
1.	What are the main components of a processor, and how do they contribute to its operation?	L1
2.	Describe the role of registers in processor organization and list some common types.	L2
3.	Outline the steps involved in the instruction execution cycle in a typical CPU.	L2
4.	How does the CPU's Control Unit facilitate the execution of an instruction?	L3
5.	Explain how control signals are generated using hardwired control signals.	L2
6.	Explain how control signals are generated using microprogrammed control signals.	L2
7.	Differentiate between hardwired control unit and microprogrammed control unit.	L2
8.	Explain how a typical multiple-bus system is structured in a processor and the role of each bus.	L2

Part-B

Memory Organization

Objective Questions

1. How many unique memory locations can a computer with a 16-bit addressing scheme access? [ ]

   A) 64K B) 4G C) 16E D) 2^64

2. In a byte-addressable memory system with 32-bit addressing, how many bytes can be addressed? [ ]

   A) 64K B) 4G C) 16E D) 2^32

3.A computer with a 64-bit address bus can theoretically address up to: [ ]

A) 4G locations B) 64K locations C) 16E locations D) 2^16 locations

4. What is the term used to describe the time from initiation of a memory request to when the data is available? [ ]

A) Memory Cycle Time B) Memory Access Time

C) Data Transfer Time D) Cache Hit Time

5. Which of the following is typically longer in duration than the memory access time? [ ]

A) Memory Access Time B) Memory Cycle Time

C) Cache Access Time D) Bus Transfer Time

6.Which type of memory has uniform access time regardless of the address being accessed? [ ]

A) Serial Access Memory B) Magnetic Disk

C) Random-Access Memory (RAM) D) Optical Disk

7.Which of the following types of storage typically has varying access times depending on the data location? [ ]

A) RAM B) Cache Memory C) Optical Disk D) Solid State Drive (SSD)

8.What is the primary purpose of cache memory in a computer system? [ ]

A) To increase the total storage capacity of the computer

B) To reduce the time the processor needs to access frequently used data

C) To extend the apparent size of physical memory using disk storage

D) To store long-term data permanently

9.Virtual memory allows a computer to [ ]

A) Run programs that are larger than physical memory

B) Store data on removable disks

C) Increase the physical size of RAM

D) Improve data transfer rates between CPU and RAM

10. In modern computer systems, data are typically transferred in [ ]

A) Single bytes

B) Contiguous blocks of data

C) Random access patterns

D) Single-word increments

11. The ability to efficiently transfer large blocks of data is crucial for: [ ]

A) Reducing cache misses

B) Optimizing CPU clock speed

C) Enhancing performance of memory and I/O operations

12. Which of the following is a primary characteristic of Static RAM (SRAM)? [ ]

A) It requires periodic refreshing to maintain data.

B) It uses a flip-flop circuit to store each bit of data.

C) It has slower access times compared to DRAM.

D) It is used primarily for long-term storage.

13. Which characteristic best describes Asynchronous DRAMs (ADR)? [ ]

A) They operate synchronously with the system clock.

B) They do not require a clock signal for timing operations.

C) They have faster access times compared to Synchronous DRAMs.

D) They are used primarily for high-speed cache memory.

14. What is the primary advantage of Synchronous DRAM (SDRAM) over Asynchronous

DRAM?

[ ]

A) SDRAM does not require periodic refreshing.

B) SDRAM operates in sync with the system clock, allowing for faster and more predictable data access.

C) SDRAM has a simpler memory cell structure.

D) SDRAM offers larger storage capacities compared to other DRAM types.

15. Which type of memory allows data to be electrically erased and reprogrammed?

[ ]

A) ROM B) PROM C) EPROM D) EEPROM

16. What is a key feature of Flash memory that differentiates it from other types of non-volatile memory? [ ]

A) It requires power to maintain data.

B) It can be electrically erased and reprogrammed in blocks.

C) It has slower write speeds compared to SRAM.

D) It is used only for long-term data storage in CPUs.

17. Which of the following statements accurately reflects the typical trade-offs between speed, size, and cost in memory systems? [ ]

A) Faster memory is always cheaper than slower memory and can store more data.

B) Larger memory sizes generally result in higher costs and slower access speeds.

C) High-speed memory, such as SRAM, is typically more expensive per bit and offers less capacity compared to slower, larger memory types like DRAM.

D) Cost is independent of memory speed and size, and all memory types provide equal storage capacities.

18. Which of the following replacement algorithms selects the cache block to be replaced based on the block that has not been used for the longest period? [ ]

A) Least Recently Used (LRU) B) First-In-First-Out (FIFO)

C) Optimal Page Replacement D) Random Replacement

Descriptive Questions

S.No.	Name of the Question	BL
1.	Explain the different cache memory mapping functions using suitable diagrams.	L2
2.	Describe the organization of virtual memory in a computer system.	L2
3.	Differentiate between magnetic hard disks, optical disks, and magnetic tape systems.	L2
4.	Compare and contrast ROM, PROM, EPROM, and EEPROM.	L2
5.	Describe the organization of a semiconductor RAM memory chip.	L2