UNIT III

Basic Processing Unit

Fundamental concepts, Execution of a complete instruction, Hardwired control, Microprogrammed control, Pipelining, Basic concepts, Data hazards, Instruction hazards, Influence on Instruction sets, Data path and control consideration.

3.1 Basic Processing Unit

3.1.1 Fundamental Concepts

Q1. Explain about instruction execution process with a block diagram.

Ans : **(Imp.)**

To execute an instruction, the processor has to perform the following three steps:

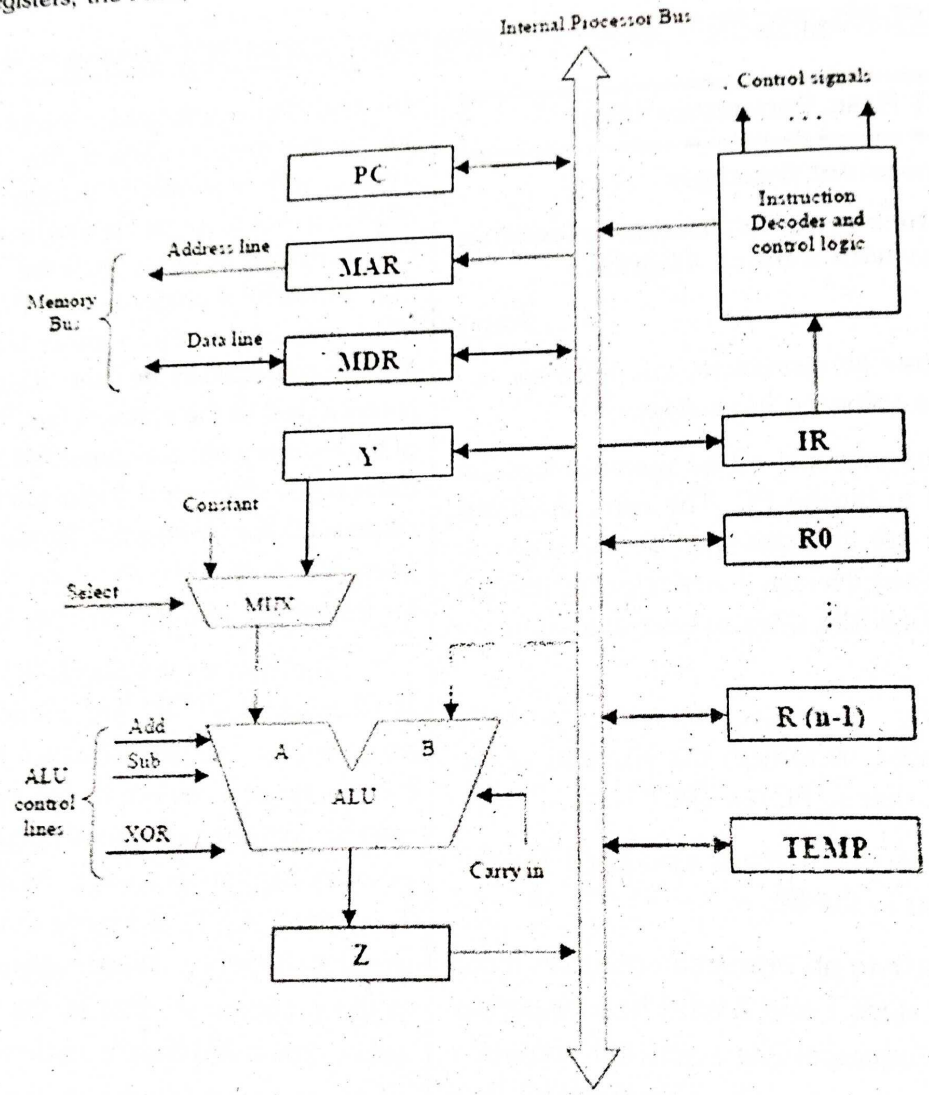
1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are interpreted as an instruction to be executed. Hence, they are loaded into the IR. Symbolically, this can be written as IR ← [[PC]]
2. Assuming that the memory is byte addressable, increment the contents of the PC by 4, that is, PC ← [PC] 4
3. Carry out the actions specified by the instruction in the IR.

In cases where an instruction occupies more than one word, steps 1 and 2 must be repeated as many times as necessary to fetch the complete instruction. These two steps are usually referred to as the fetch phase; step 3 constitutes the execution phase. Figure shows an 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 shown in Figure connected to the internal processor bus via the memory data register, MDR, and the memory address register, MAR, respectively. Register MDR has two inputs and two outputs. Data may be loaded into MDR either from the memory bus or from the internal processor bus. The data stored in MDR may be placed on either bus. The input of MAR is connected to the internal bus, and its output is connected to the external bus. The control lines of the memory bus are connected to the instruction decoder and control logic block. This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus.

The number and use of the processor registers R0 through R(n - 1) vary considerably from one processor to another. Registers may be provided for general-purpose use by the programmer. Some may be dedicated as special-purpose registers, such as index registers or stack pointers. Three registers, Y, Z, and TEMP in Figure 2.1, have not been mentioned before. These registers are transparent to the programmer, that is, the programmer need not be concerned with them because they are never referenced explicitly by any instruction. They are used by the processor for temporary storage during execution of some instructions. These registers are never used for storing data generated by one instruction for later use by another instruction. The multiplexer MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU. The constant 4 is used to increment the contents of the program counter. We will refer to

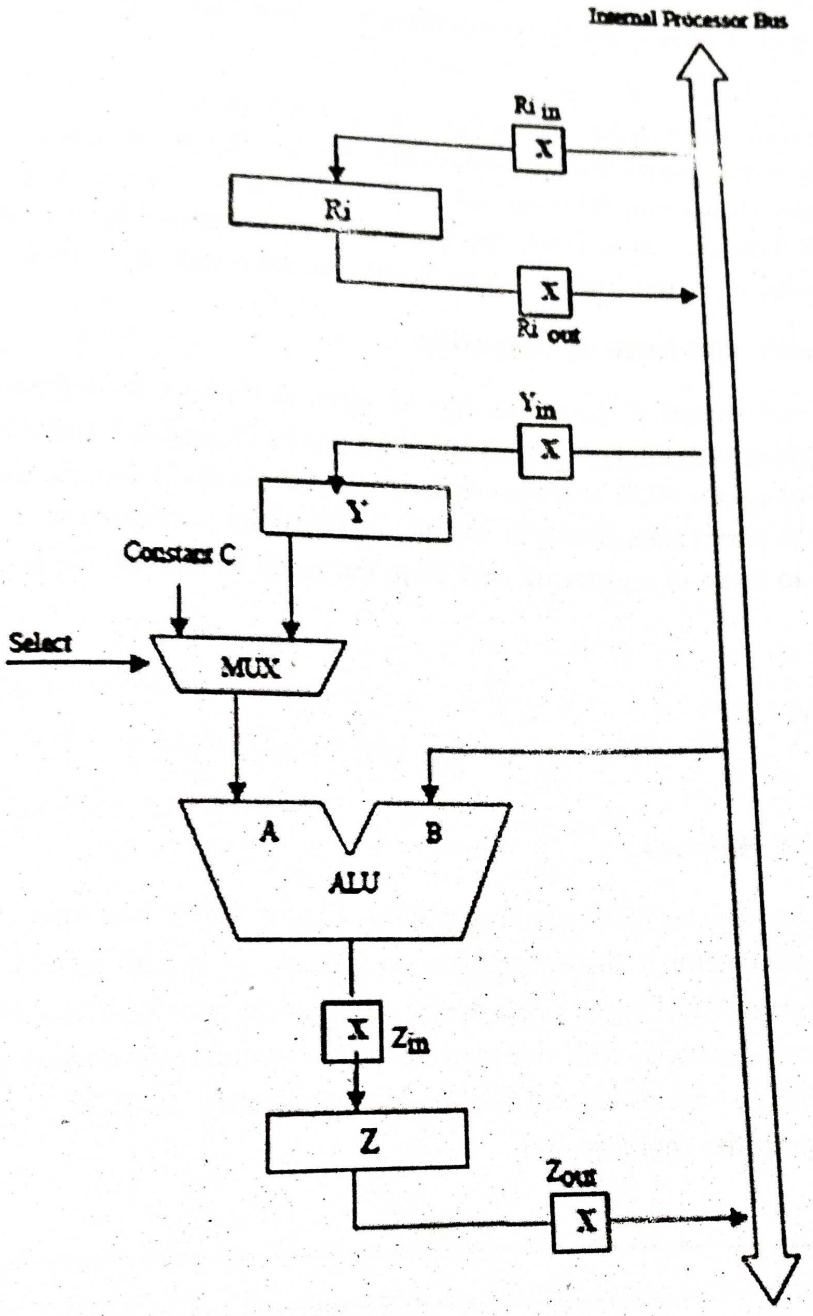
the two possible values of the MUX control input Select as Select4 and SelectY for selecting the constant 4 or register Y, respectively.

As instruction execution progresses, data are transferred from one register to another, often passing through the AL U to perform some arithmetic or logic operation. The instruction decoder and control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register. The decoder generates the control signals needed to select the registers involved and direct the transfer of data. The registers, the ALU, and the interconnecting bus are collectively referred to as the datapath.



Above Figure Single bus organization of the data path inside a processor. 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 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



Register Transfer

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. This is represented symbolically in Figure 2.2. The input and output of register Ri are connected to the bus via switches controlled by the signals Riin and Ri out respectively. When Riin is set to 1, the data on the bus are loaded into Ri. Similarly, when Riout is set to 1, the contents of register Ri are placed on the bus. While Riout is equal to 0, the bus can be used for transferring data from other registers. Suppose that we wish to transfer the contents of register Rl to register R4. This can be accomplished as follows:

- Enable the output of register Rl by setting R1out to 1. This places the contents of R 1 on the processor bus.
- Enable the input of register R4 by setting R4in to 1. 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. The control signals that govern a particular transfer are asserted at the start of the clock cycle. In our example, R1out and R4in are set to 1. The registers consist of edge-triggered flip-flops. Hence, at the next active edge of the clock, the flip-flops that constitute R4 will load the data present at their inputs. At the same time, the control signals R1out and R4in will return to 0.

Performing Arithmetic and Logical 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 2.1 and 2.2, 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 Rl to those of register R2 and store the result in register R3 is

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

Fetching A Word From Memory

The connections for register MDR are illustrated in Figure 2.4. It has four control signals: MDR in and MDR_{out} control the connection to the internal bus, and MDR_{inE} and MDRout E control the connection to the external bus. The circuit in Figure 2.3 is easily modified to provide the additional connections. A three-input multiplexer can be used, with the memory bus data line connected to the third input. This input is selected when $MDR_{inE} = 1$. A second tri-state gate, controlled by MDR_{outE} can be used to connect the output of the flip-flop to the memory bus.

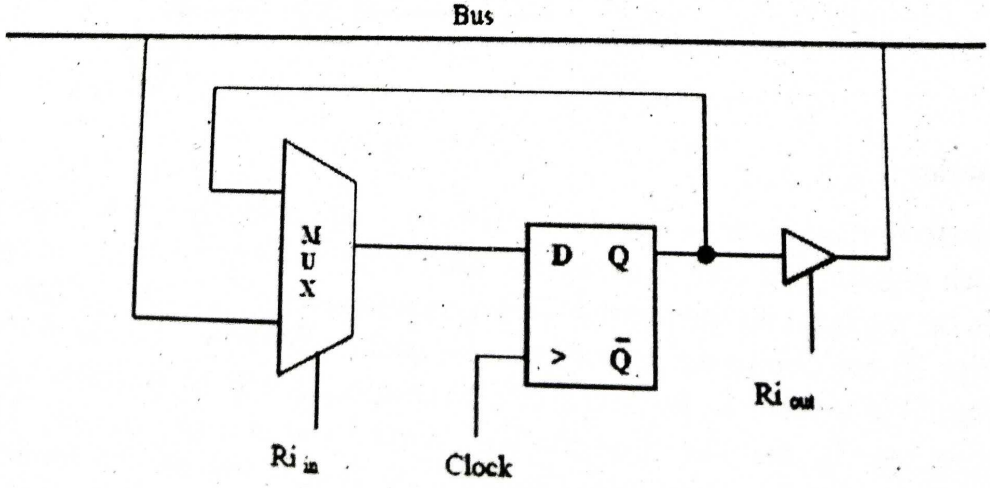


Fig.: Input and output gating for one register bit

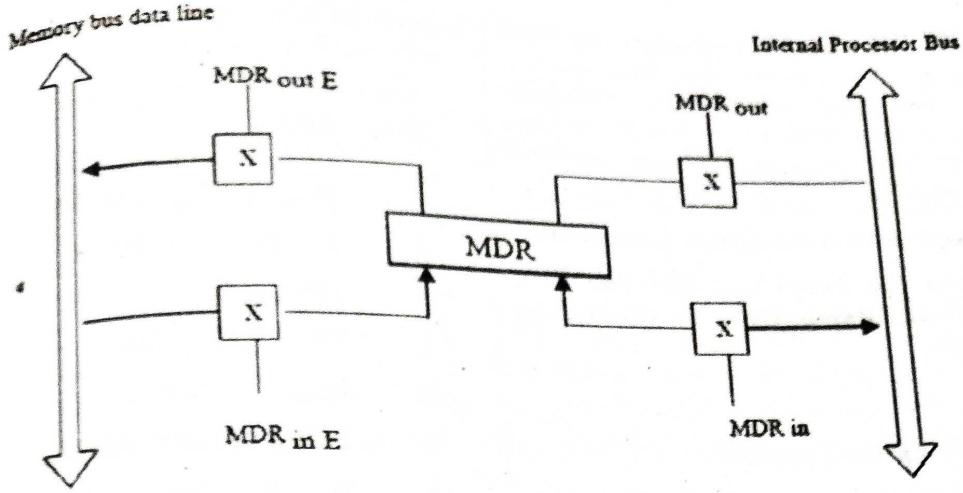


Fig.: Connections and control signals for register MDR

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

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC 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. $R1_{out}$, 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.

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,(R 1) 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.

3.1.2 Execution of A Complete Instruction

Q2. Explain the execution process of a complete instruction by taking an example.

Ans :

Complete Instruction Execution

Consider the instruction Add (R3),Rl , which adds the contents of a memory location pointed to by R3 to register R1. Executing this instruction requires the following actions:

1. Fetch the instruction.
2. Fetch the first operand (the contents of the memory location pointed to by R3).
3. Perform the addition.
4. Load the result into R1.

Figure gives the sequence of control steps required to perform these operations for the singlebus architecture of Figure 2.1. Instruction execution proceeds as follows. In step 1, the instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a Read request to the memory. The Select signal is set to Select4, which causes the multiplexer MUX to select the constant 4. This value is added to the operand at input B, which is the contents of the PC, and the result is stored in register Z. The updated value is moved from register Z back into the PC during step 2, while waiting for the memory to respond. In step 3, the word fetched from the memory is loaded into the IR.

Steps 1 through 3 constitute the instruction fetch phase, which is the same for all instructions. The instruction decoding circuit interprets the contents of the IR at the beginning of step 4. This enables the control circuitry to activate the control signals for steps 4 through 7, which constitute the execution phase. The contents of register R3 are transferred to the MAR in step 4, and a mem read operation is initiated.

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

Fig. : Control sequence for execution of instructio Add (R3), Rl

Then the contents of R1 are transferred register Y in step 5, to prepare for the additi operation. When the Read operation is complete the memory operand is available in register MD and the addition operation is performed in step The contents of MDR are gated to the bus, a thus also to the B input of the ALU, and register Y selected as the second input to the ALU by choosi Select Y The sum is stored in register Z, th transferred to R 1 in step 7. The End signal causes new instruction fetch cycle to begin by returning step 1.

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

Branch Instruction

A branch instruction replaces the contents the PC with the branch target address. This addr

is usually obtain the PC. Figure starts, as usual, The offset valu extension if rec gated onto the target address,

The offs and the addre location 2000 be readily ap before knowi step 4, the instruction ir

Co before lo step 4 in

O

T to load

3.1.3

Q3.

Ans :

Hardw

neede The a

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. 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 reason for this can be readily appreciated from the control sequence in Figure The PC is incremented during the fetch phase, before knowing the type of instruction being executed. Thus, when the branch address is computed in step 4, the PC value used is the updated value, which points to the instruction following the branch instruction in the memory.

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

Fig.: Control sequence for an unconditional instruction

Consider now a conditional branch. In this case, we need to check the status of the condition codes before loading a new value into the PC. For example, for a Branch-on-negative (Branch <0) instruction, step 4 in Figure 2.6 is replaced with

Offset-field-of-IR_{out}, Add, Z_{in}, If N = 0 then End

Thus, if N = 0 the processor returns to step 1 immediately after step 4. If N = 1, step 5 is performed to load a new value into the PC, thus performing the branch operation.

3.1.3 Hardwired Control

Q3. Explain hard wired control architecture. **(Imp.)**

Ans :

Hardwired Control

To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. Computer designers use a wide variety of techniques to solve this problem. The approaches used fall into one of two categories: hardwired control and micro-programmed control.

The required control signals are determined by the following information:

1. Contents of the control step counter
2. Contents of the instruction register
3. Contents of the condition code flags
4. External input signals, such as MFC and interrupt requests

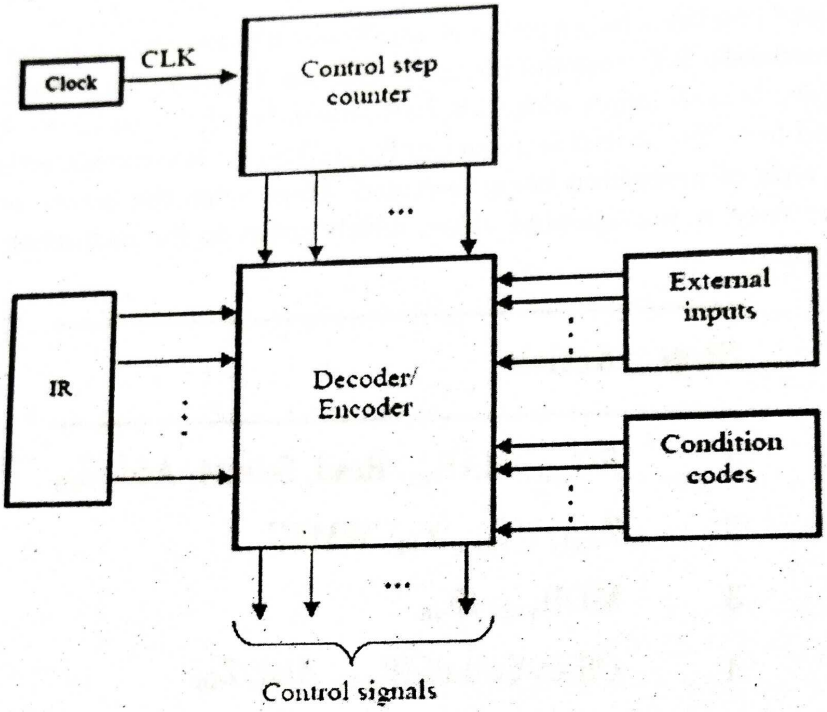


Fig. (a): Control unit organization

To gain insight into the structure of the control unit, we start with a simplified view of the hardware involved. The decoder/encoder block in above Figure (a) 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 the more detailed block diagram in Figure (b). 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. 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 O. (For design details of decoders, refer to Appendix A.) The input signals to the encoder block in Figure are combined to generate the individual control signals Y in , PC OUh Add, End, and so on. An example of how the encoder generates the Zin control signal for the processor organization is given in Figure. This circuitimplements the logic function.

This signal is asserted during time slot Tl for all instructions, during T6 for an Add instruction, during T 4 for an unconditional branch instruction, and so on. Following figure (a) gives a circuit that generates the End control signal from the logic function.

$$Z_{in} = T_1 + T_6 . ADD + T_4 . BR + \ldots \quad (3.1)$$

$$End = T_7 . ADD + T_5 . BR + (T_5 . N + T_4 . \overline{N}) . BRN + \ldots \quad (3.2)$$

The End signal starts a new instruction fetch cycle by resetting the control step counter to its starting value. Figure 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 equal to 0, the counter stops counting. This is needed whenever the WMFC signal is issued, to cause the processor to wait for the reply from the memory.

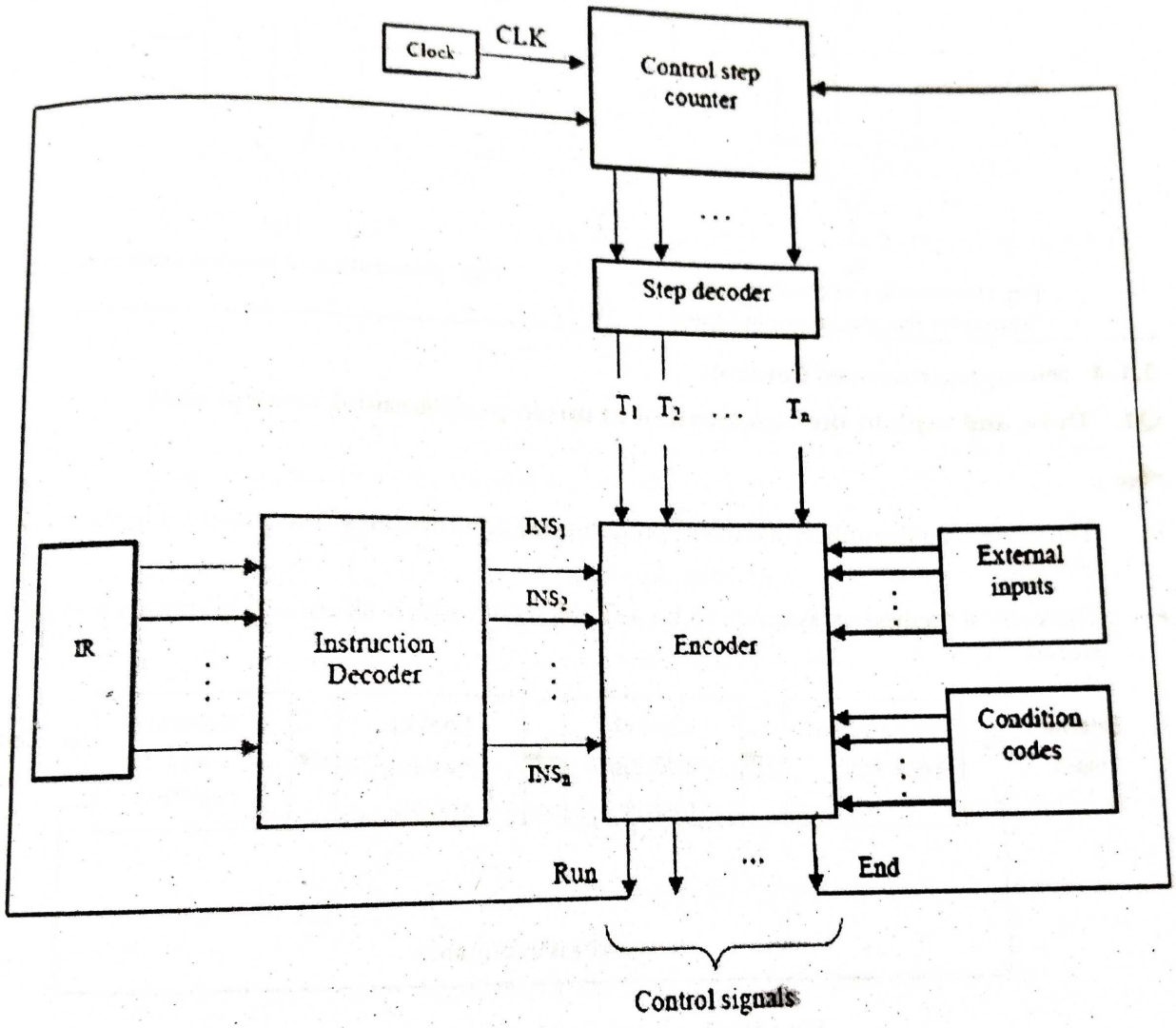


Fig. (b): Separation of decoding and encoding functions

The control hardware shown in Figure a or b can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes, and the external inputs.

The outputs of the state machine are the control signals. The sequence of operations carried out by this machine is determined by the wiring of the logic elements, hence the name "hardwired." A controller that uses this approach can operate at high speed. However, it has little flexibility, and the complexity of the instruction set it can implement is limited.

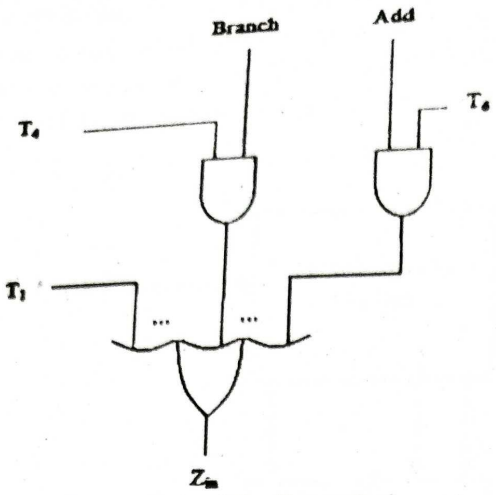


Fig. : Generation of the Z_{in} control signal for the processor in figure.

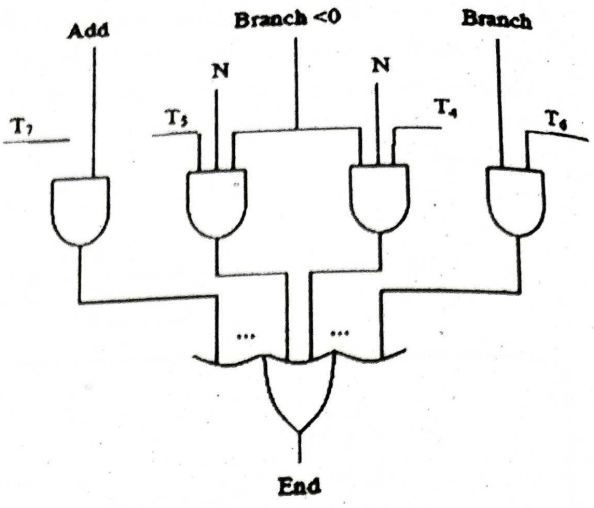


Fig. : Generation of the end control signal

3.1.4 Microprogrammed Control

Q4. Draw and explain the organization of micro programmed control unit.

Ans : **(Imp.)**

- The general configuration of a micro-programmed control unit is demonstrated in the block diagram of Figure.
- The control memory is assumed to be a ROM, within which all control information is permanently stored.

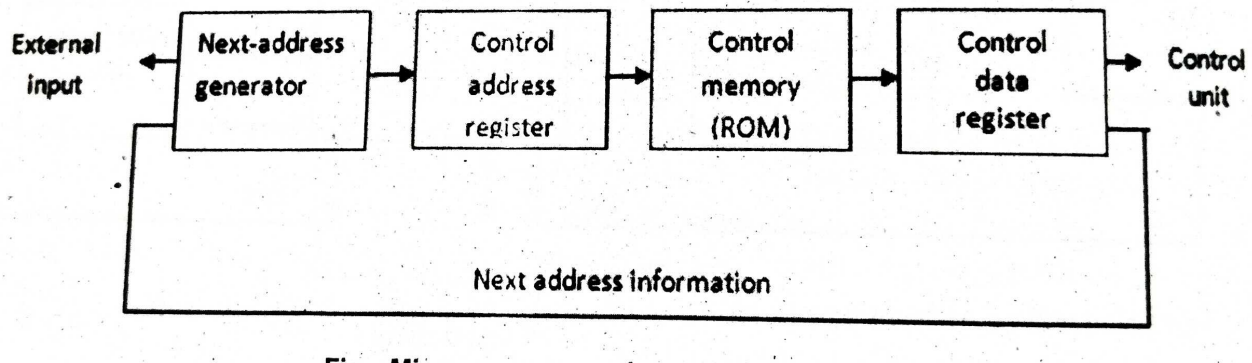


Fig. : Micro-programmed control organization

- The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the next address.
- The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory.
- While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

- Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.
- The next address generator is sometimes called a **micro-program sequencer**, as it determines the address sequence that is read from control memory.
- Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.
- The control data register holds the present microinstruction while the next address is computed and read from memory.
- The data register is sometimes called a pipeline register.
- It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.
- This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.
- The main advantage of the micro programmed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.
- If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.

Q5. Explain the steps of Address Sequencing in detail.

Ans :

Microinstructions are stored in control memory in groups, with each group specifying a routine.

To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

Step-1:

- An initial address is loaded into the control address register when power is turned on in the computer.
- This address is usually the address of the first microinstruction that activates the instruction fetch routine.
- The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.
- At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2:

- The control memory next must go through the routine that determines the effective address of the operand.
- A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.
- The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
- When the effective address computation routine is completed, the address of the operand is availab in the memory address register.

Step-3:

- The next step is to generate the microoperations that execute the instruction fetched from mem
- The microoperation steps to be generated in processor registers depend on the operation code
- of the instruction.
- Each instruction has its own micro-program routine stored in a given location of control mem
- The transformation from the instruction code bits to an address in control memory where
- routine is located is referred to as a mapping process.
- A mapping procedure is a rule that transforms the instruction code into a control memory addr

Step-4:

- Once the required routine is reached, the microinstructions that execute the instruction may
- sequenced by incrementing the control address register.
- Micro-programs that employ subroutines will require an external register for storing the ret
- address.
- Return addresses cannot be stored in ROM because the unit has no writing capability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first addres
- the fetch routine.

Q6. Draw and explain selection of address for control memory.

Ans :

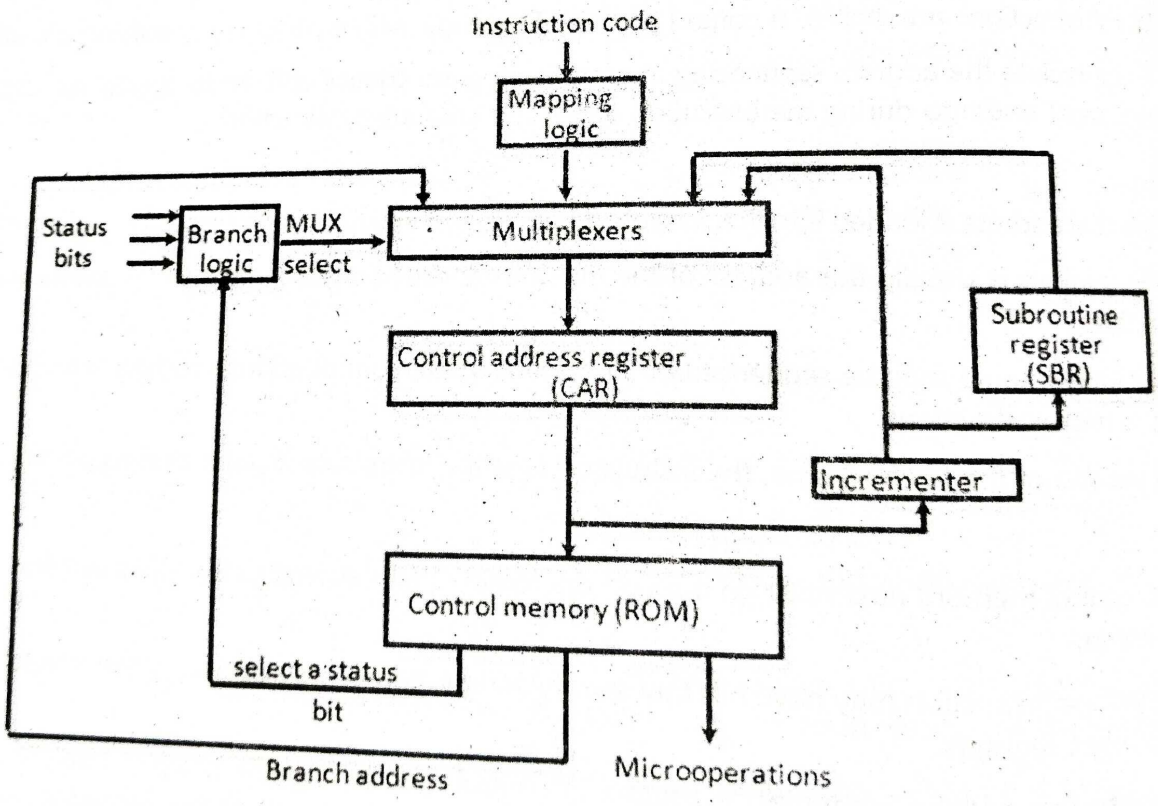


Fig.: Selection of address for control memory

- Above figure shows a block diagram of a control memory and the associated hardware needed
- selecting the next microinstruction address.

- The micr
- registers
- The dia
- address
- The inc
- microin
- Branch
- Condit
- in ord
- An ex
- The r
- micro
- The
- The
- carr
- statu
- The
- the
- A
- mi
- A

Q7. Ex

Ans :

- A
- n
- T
- F

- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.
- The diagram shows four different paths from which the control address register (CAR) receives the address.
- The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.
- Branching is achieved by specifying the branch address in one of the fields of the microinstruction.
- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- An external address is transferred into control memory via a mapping logic circuit.
- The return address for a subroutine is stored in a special register whose value is then used when the micro-program wishes to return from the subroutine.
- The branch logic of figure 5 provides decision-making capabilities in the control unit.
- The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.
- A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register.
- A 0 output in the multiplexer causes the address register to be incremented.

Q7. Explain Mapping of an Instruction.

Ans :

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
- The status bits for this type of branch are the bits in the operation code part of the instruction.
- For example, a computer with a simple instruction format as shown in figure has an operation code of four bits which can specify up to 16 distinct instructions.
- Assume further that the control memory has 128 words, requiring an address of seven bits.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.
- If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

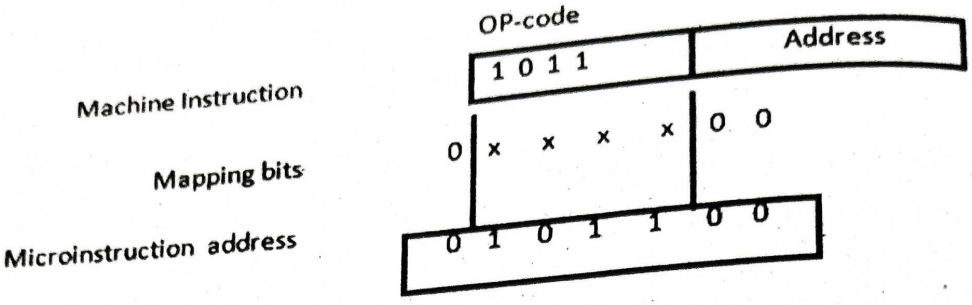


Fig. : Mapping from instruction code to microinstruction address

- One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function.
- The contents of the mapping ROM give the bits for the control address register.
- In this way the microprogram routine that executes the instruction can be placed in any desired location in control memory.
- The mapping concept provides flexibility for adding instructions for control memory as the need arises.

Q8. Draw and explain Computer Hardware Configuration in detail.

Ans :

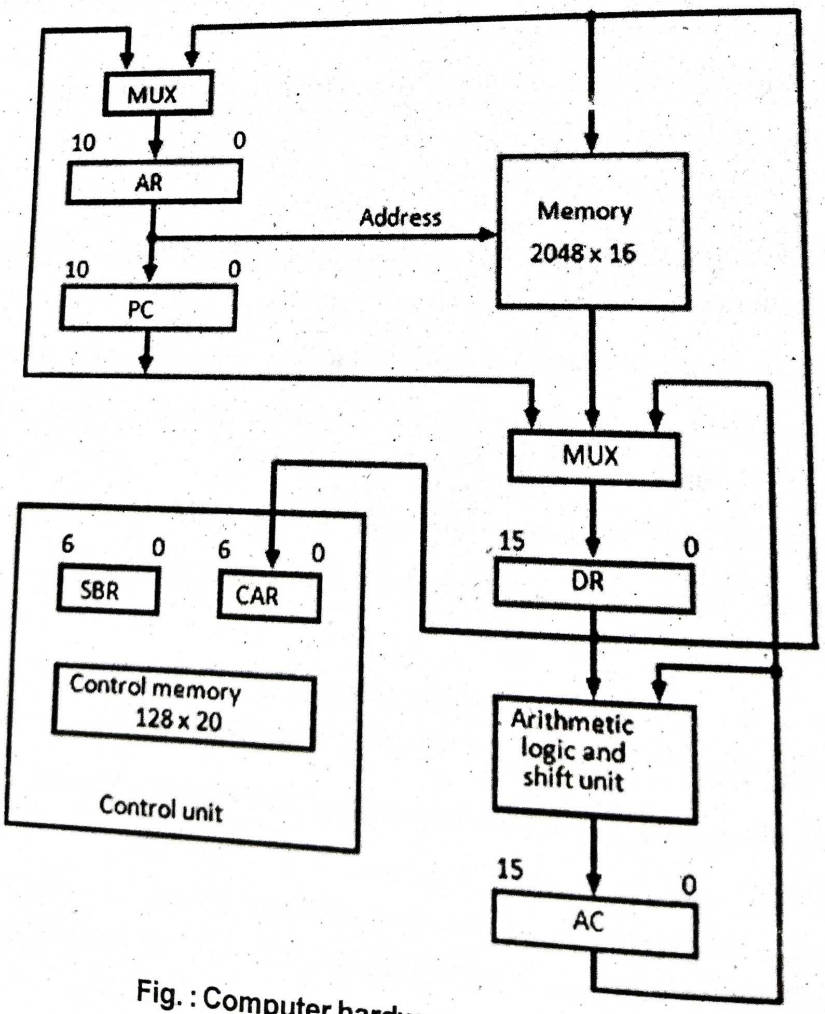


Fig. : Computer hardware configuration

The block diagram of the computer is shown in Figure. It consists ofTwo memory units:
Main memory → for storing instructions and data, and Control memory
→ for storing the microprogram.

Six Registers

Processor unit register: AC(accumulator),PC(Program Counter), AR(Address Register), DR(Data Register)

Control unit Register

CAR (Control Address Register), SBR(Subroutine Register)

Multiplexers

The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

ALU

The arithmetic, logic, and shift unit performs microoperations with data from AC and DR and places the result in AC.

DR can receive information from AC, PC, or memory.

AR can receive information from PC or DR.

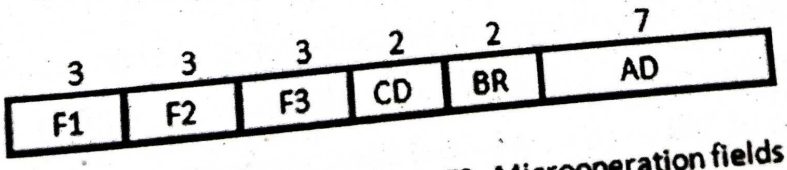
PC can receive information only from AR.

Input data written to memory come from DR, and data read from memory can go only to DR.

Q9. Explain Microinstruction Format in detail.

Ans :

- The microinstruction format for the control memory is shown in figure. The 20 bits of the microinstruction are divided into four functional parts as follows:
- The three fields F1, F2, and F3 specify microoperations for the computer.
- The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. This gives a total of 21 microoperations.
- The CD field selects status bit conditions.
- The BR field specifies the type of branch to be used.
- The AD field contains a branch address. The address field is seven bits wide, since the control memory has $128 = 2^7$ words.



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

Fig. : Microinstruction Format

As an example, a microinstruction can specify two simultaneous microoperations from F2 and F3 and none from F1.

DR → M[AR] with F2 = 100

PC → PC + 1 with F3 = 101

The nine bits of the microoperation fields will then be 000 100 101.

1. Label: The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
2. Microoperations It consists of one, two, or three symbols, separated by commas, from those defined in Table 5.3. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations.

   This will be translated by the assembler to nine zeros.
3. CD The CD field has one of the letters U, I, S, or Z.
4. BR The BR field contains one of the four symbols defined in Table
5. AD The AD field specifies a value for the address field of the microinstruction in one of three possible ways:
   (i) With a symbolic address, this must also appear as a label.
   (ii) With the symbol NEXT to designate the next address in sequence.
   (iii) When the BR field contains a RET or MAP symbol, the AD field is left empty and is converted to seven zeros by the assembler.

The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 1.1

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

Table 1: Condition Field

The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction

Q10. What is Symbolic Microinstruction.

Ans :

Each line of the assembly language microprogram defines a symbolic microinstruction.

Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD.

Q11. Draw the diagram of Micro programmed sequencer for acontrol memory and explain it. **(Imp.)**

Ans :

Microprogram Sequencer

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.

- The address selection part is called a microprogram sequencer.
- A microprogram sequencer can be constructed with digital functions to suit a particular application.
- To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range of applications.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutine calls.
- Some sequencers provide an output register which can function as the address register for the control memory.

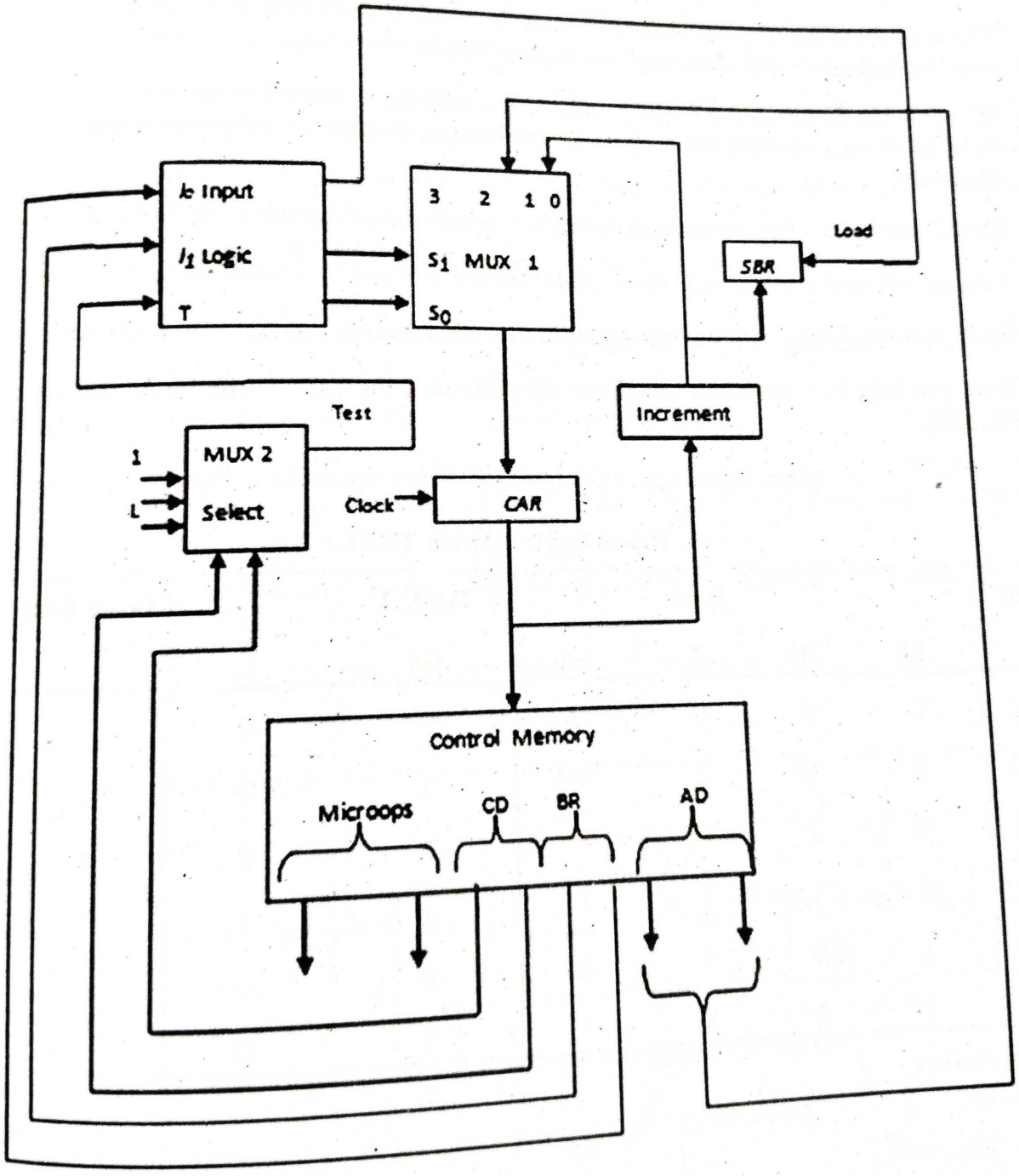


Fig. : Microprogram Sequencer for a control memory

- The block diagram of the microprogram sequencer is shown in figure.

There are two multiplexers in the circuit.

- The first multiplexer selects an address from one of four sources and routes it into a control addr register CAR.
- The second multiplexer tests the value of a selected status bit and the result of the test is applied an input logic circuit.
- The output from CAR provides the address for the control memory.
- The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutin registers SBR.
- The other three inputs to multiplexer 1 come from the address field of the present microinstructio from the output of SBR, and from an external source that maps the instruction.
- Although the figure shows a single subroutine register, a typical sequencer will have a registe stack about four to eight levels deep. In this way, a number of subroutines can be active at th same time.
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexe
- If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise, it is equal to 0.
- The T value together with the two bits from the BR (branch) field goes to an input logic circuit.
- The input logic in a particular sequencer will determine the type of operations that are available in the unit.

Table: Input Logic Truth Table for Microprogram Sequencer

Input Logic : Truth Table

BR	Input				MUX 1		Load SBR
	I1	I0	T	S1	S0	L	
0 0	0	0	0	0	0	0	
0 0	0	0	1	0	1	0	
0 1	0	1	0	0	0	0	
0 1	0	1	1	0	1	1	
1 0	1	0	X	1	0	0	
1 1	1	1	X	1	1	0	

Boolean Function

S0 = I0

S1 = I0I1 + I0'T

L = I0'I1T

- Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations.
- With three inputs, the sequencer can provide up to eight address sequencing operations.
- Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range of operations.

3.2 PIPELINING

Q12. Explain pipelining technique. Draw the general structure of four segment pipeline.

(Imp.)

Ans :

Pipeline is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- Each segment performs partial processing dictated by the way the task is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously.
- Any operation that can be decomposed into a sequence of sub operations of about the same complexity can be implemented by a pipeline processor.
- The technique is efficient for those applications that need to repeat the same task many times with different sets of data.
- The general structure of a four-segment pipeline is illustrated in Figure.

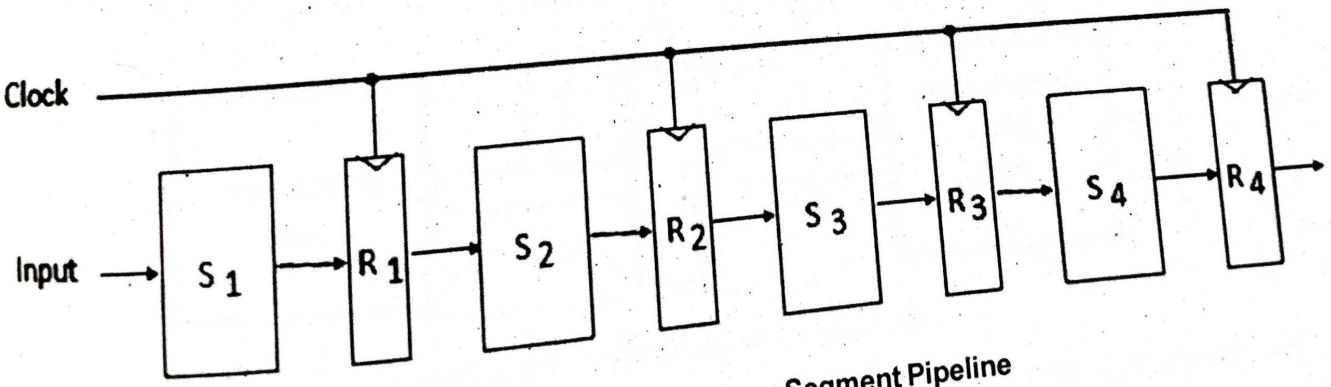


Fig.: General Structure of Four-Segment Pipeline

- The operands pass through all four segments in a fixed sequence.
- Each segment consists of a combinational circuit S, which performs a sub operation over the data stream flowing through the pipe.

- The segments are separated by registers R, which hold the intermediate results between the stages.
- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.
- We define a task as the total operation performed going through all the segments in the pipeline.

3.2.1 Basic Concepts

Q13. Write about the basic concepts of pipelining. **(Imp.)**

Ans :

Pipelining is a particularly effective way of organizing concurrent activity in a computer system. The basic idea is very simple. It is frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly-line operation. The processor executes a program by fetching and executing instructions, one after the other. Let Fi and Ei refer to the fetch and execute steps for instruction Ii. Executions of a program consists of a sequence of fetch and execute steps, as shown in Figure 3.1.

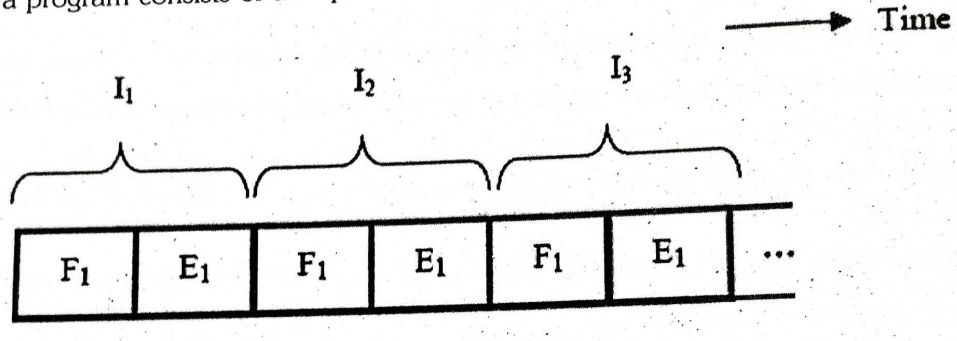


Fig.: Fettch and Execute steps

Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them, as shown in Figure 10. The instruction fetched by the fetch unit is deposited in an intermediate storage buffer, B1. This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction. The results of execution are deposited in the destination location specified by the instruction. The data can be operated by the instructions are inside the block labeled "Execution unit".

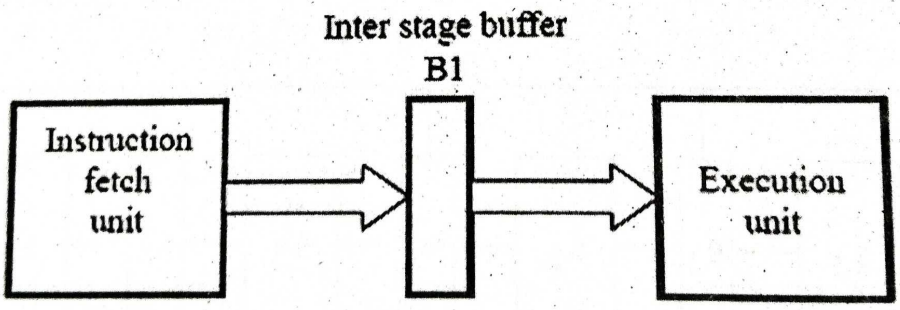


Fig.: Hardware organization of pipelining.

The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle. Operation of the computer proceeds as in Figure In the first clock cycle, the fetch unit fetches an instruction I1 (step F1) and stores it in buffer Bl at the end of the clock cycle. In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I2 (step F2). Meanwhile, the execution unit performs the operation specified by instruction I1,

which is available to it in buffer B1 (step E1). By the end of the second clock cycle, the execution of instruction I1 is completed and instruction I2 is available. Instruction I2 is stored in B1, replacing I1, which is no longer needed. Step E2 is performed by the execution unit during the third clock cycle, while instruction I3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time.

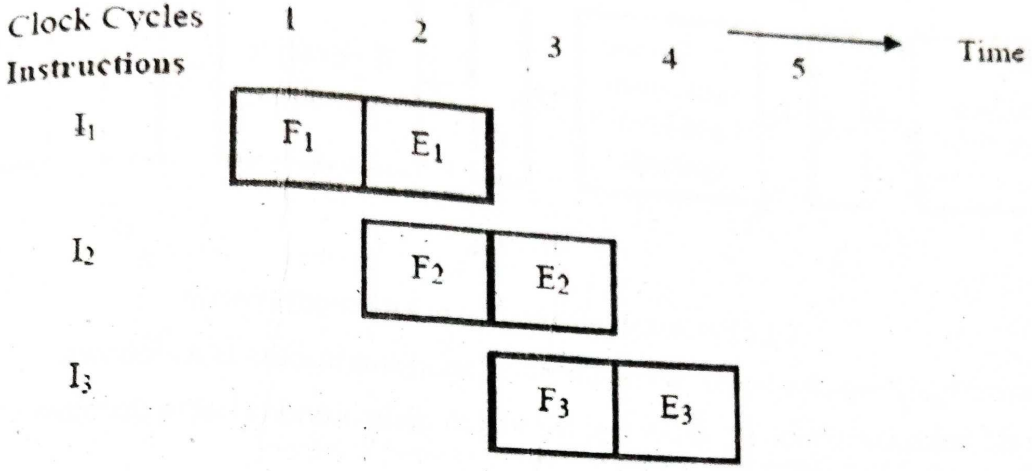


Fig.: Pipelined executions of instructions (Instructions Pipelining)

A pipelined processor may process each instruction in four steps, as follows:

F Fetch: read the instruction from the memory.

D Decode: decode the instruction and fetch the source operand(s).

E Execute: perform the operation specified by the instruction.

W Write: store the result in the destination location.

The sequence of events for this case is shown in Figure. Four instructions are in progress at any given time. This means that four distinct hardware units are needed, as shown in Figure. These units must be capable of performing their tasks simultaneously and without interfering with one another. Information is passed from one unit to the next through a storage buffer.

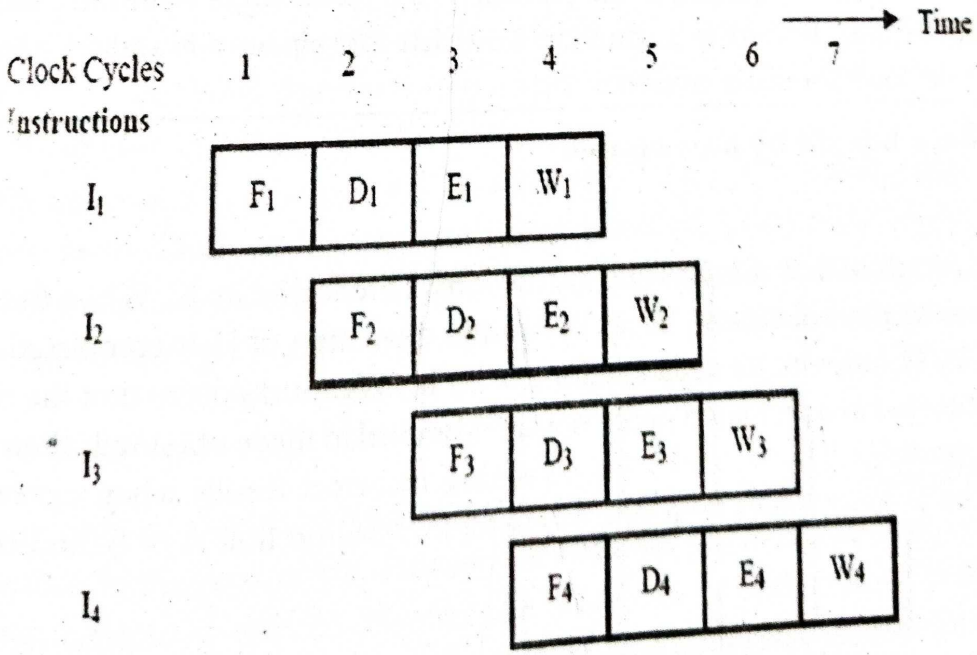


Fig.: Instruction execution divided into four steps

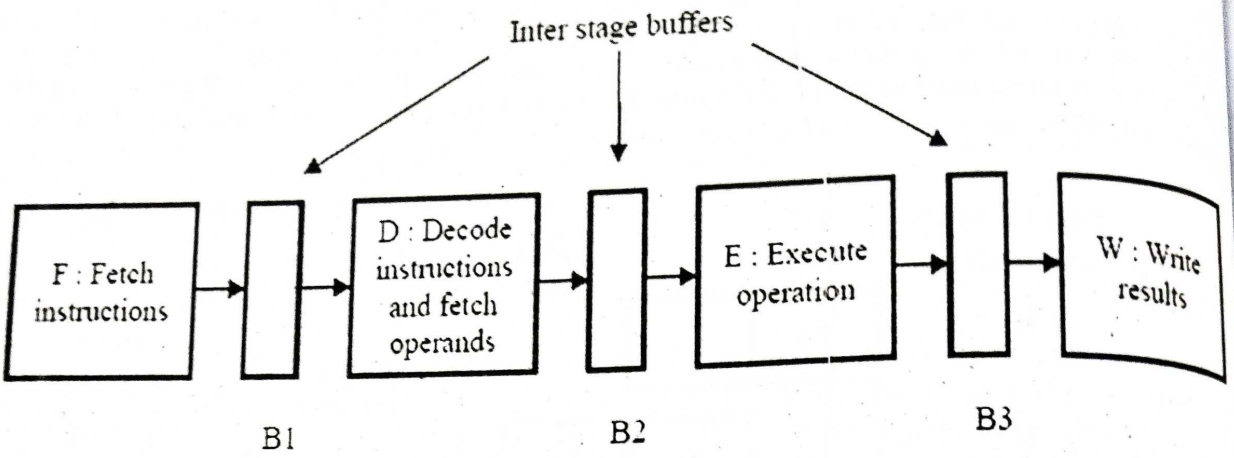


Fig.: Hardware organization of a 4-stage pipeline

For example, during clock cycle 4, the information in the buffers is as follows:

- Buffer B1 holds instruction I3, which was fetched in cycle 3 and is being decoded by the instruction decoding unit.
- Buffer B2 holds both the source operands for instruction I2 and the specification of the operation to be performed.
- Buffer B3 holds the results produced by the execution unit and the destination information for instruction 11.

3.2.2 Data Hazards

Q14. What are data hazards.

Ans :

Data hazards

A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result some operation has to be delayed, and the pipeline stalls. A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason.

Q15. Explain data hazard by an example.

Ans :

Consider a program that contains two instructions, I1 followed by I2. When this program is executed in a pipeline, the execution of 12 can begin before the execution of I1 is completed. This means that the results generated by I1 may not be available for use by I2. We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially. The potential for obtaining incorrect results when operations are performed concurrently can be demonstrated by a simple example. Assume that A = 5, and consider the following two operations:

A ← 3 A

B ← 4 * A

When these operations are performed in the order given, the result is B == 32. But if they are performed concurrently, the value of A used in computing B would be the original value, 5, leading to an incorrect result. If these two operations are performed by instructions in a program, then the instructions must be executed one after the other, because the data used in the second instruction depend on the result of the first instruction. On the other hand, the two operations

$A \leftarrow 5 \times C$

$B \leftarrow 20\ C$

can be performed concurrently, because these operations are independent.

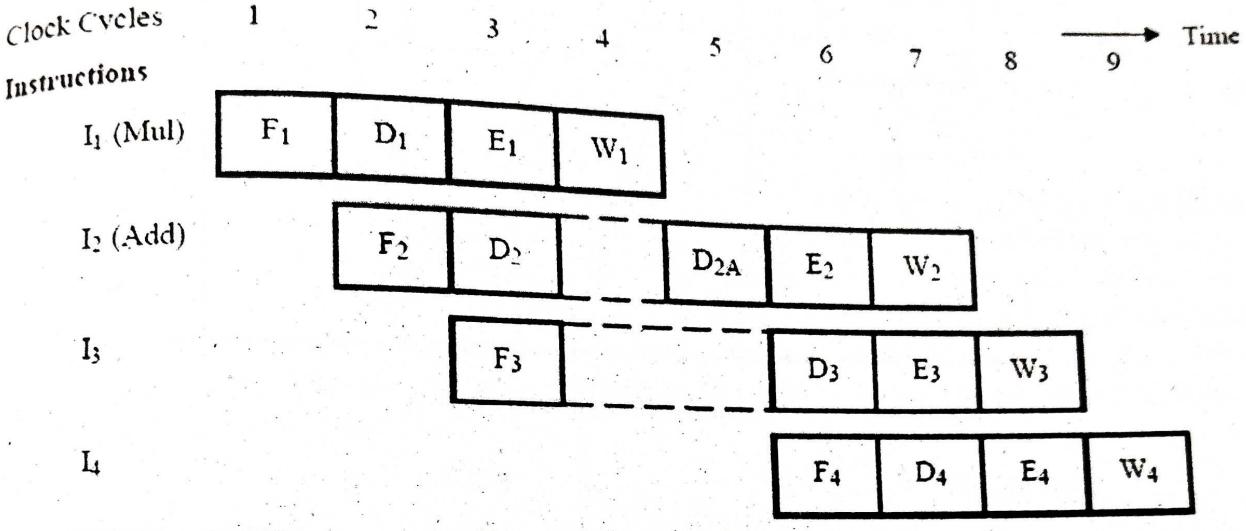
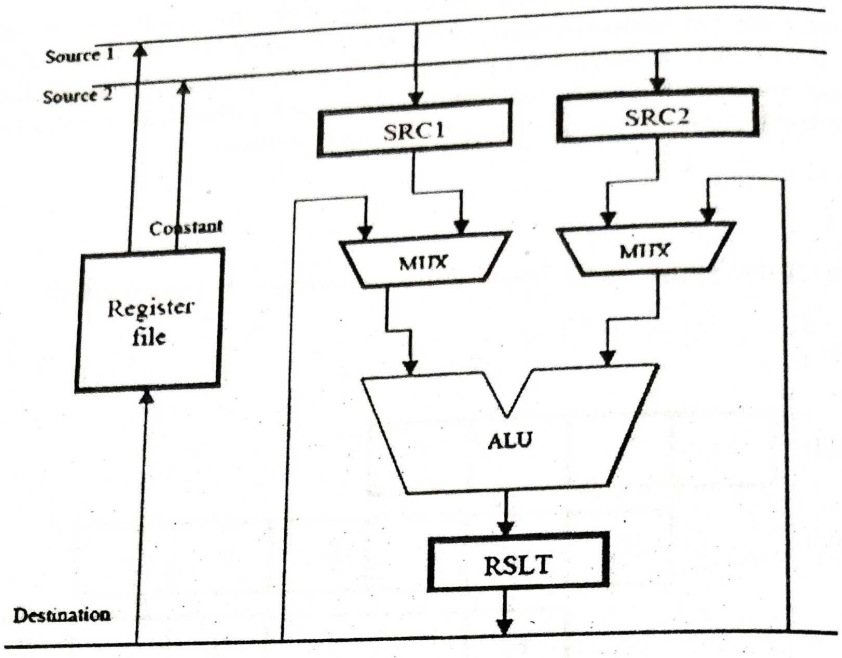


Fig. Pipeline stalled by data dependency between D2 and W1

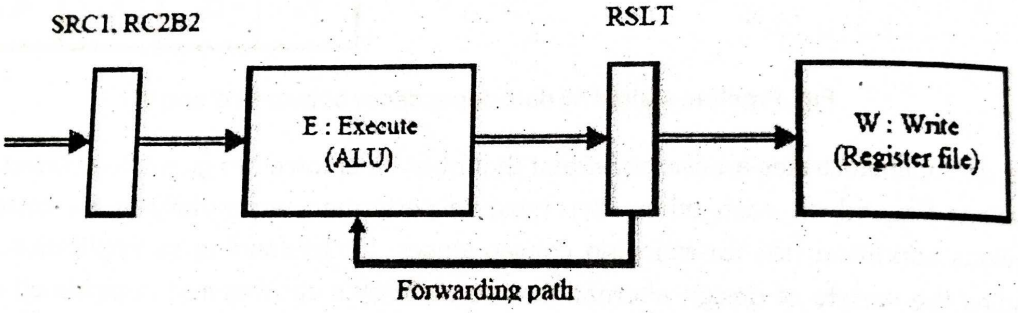
This example illustrates a basic constraint that must be enforced to guarantee correct results. When two operations depend on each other, they must be performed sequentially in the correct order. This rather obvious condition has far-reaching consequences. Understanding its implications is the key to understanding the variety of design alternatives and trade-off's encountered in pipelined computers. For example, the two instructions Mul R2,R3,R4 and Add RS,R4,R6 give rise to a data dependency. The result of the multiply instruction is placed into register R4, which in turn is one of the two source operands of the Add instruction. Assuming that the multiply operation takes one clock cycle to complete, execution would proceed as shown in Figure 12. As the Decode unit decodes the Add instruction in cycle 3, it realizes that R4 is used as a source operand. Hence, the D step of that instruction cannot be completed until the W step of the multiply instruction has been completed. Completion of step D2 must be delayed to clock cycle 5, and is shown as step D2A in the figure. Instruction h is fetched in cycle 3, but its decoding must be delayed because step D3 cannot precede D2. Hence, pipelined execution is stalled for two cycles.

Operand Forwarding

The data hazard just described arises because one instruction, instruction I2 in Figure 3.9, is waiting for data to be written in the register file. However, these data are available at the output of the ALU once the Execute stage completes step El. Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I1 to be forwarded directly for use in step E2.



(a) Forwarding data path



(b) Posision of the source and result registers in the processor pipeline

Fig.: Operand forwarding in a pipelined processor

Figure (a) shows a part of the processor datapath involving the ALU and theregister file. This arrangement is similar to the three-bus structur, except that registers SRCl, SRC2, and RSLT have been added. These registers constitute interstage buffers needed for pipelined operation, as illustrated in Figure 3.10b. With reference to Figure (b), registers SRC1 and SRC2 are part of buffer B2 and RSLT is part of B3. The data forwarding mechanism is provided by the blue connection lines. The two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of either the SRCI or SRC2 register. When the instructions in Figure 12 are executed in the datapath of Figure 13 the operations performed in each clock cycle are as follows. After decoding instruction I2 and detecting the data dependency, a decision is made to use data forwarding. The operand not involved in the dependency, register R2, is read and loaded in register SRCI in clock cycle 3. In the next clock cycle, the product produced by instruction I1 is available in register RSLT, and because of the forwarding connection, it can be used in step E2. Hence, execution of I2 proceeds without interruption.

3.2.3 Instruction Hazards

Q16. What are called instruction hazards?

Ans :

Instruction Hazards

Pipeline execution of instructions will reduce the time and improves the performance. Whenever this stream is interrupted, the pipeline stalls, as figure illustrates for the case of a cache miss. A branch instruction may also cause the pipeline to stall. The effect of branch instructions and the techniques that can be used for mitigating their impact are discussed with unconditional branches and conditional branches.

Q17. Explain, how branching will be done in instruction hazards. **(Imp.)**

Ans :

Unconditional Branches

A sequence of instructions being executed in a two-stage pipeline is shown in Figure. Instructions I1 to I3 are stored at successive memory addresses, and I2 is a branch instruction. Let the branch target be instruction Ik. In clock cycle 3, the fetch operation for instruction 13 is in progress at the same time that the branch instruction is being decoded and the target address computed. In clock cycle 4, the processor must discard I3, which has been incorrectly fetched, and fetch instruction Ik. In the meantime, the hardware unit responsible for the Execute (E) step must be told to do nothing during that clock period. Thus, the pipeline is stalled for one clock cycle.

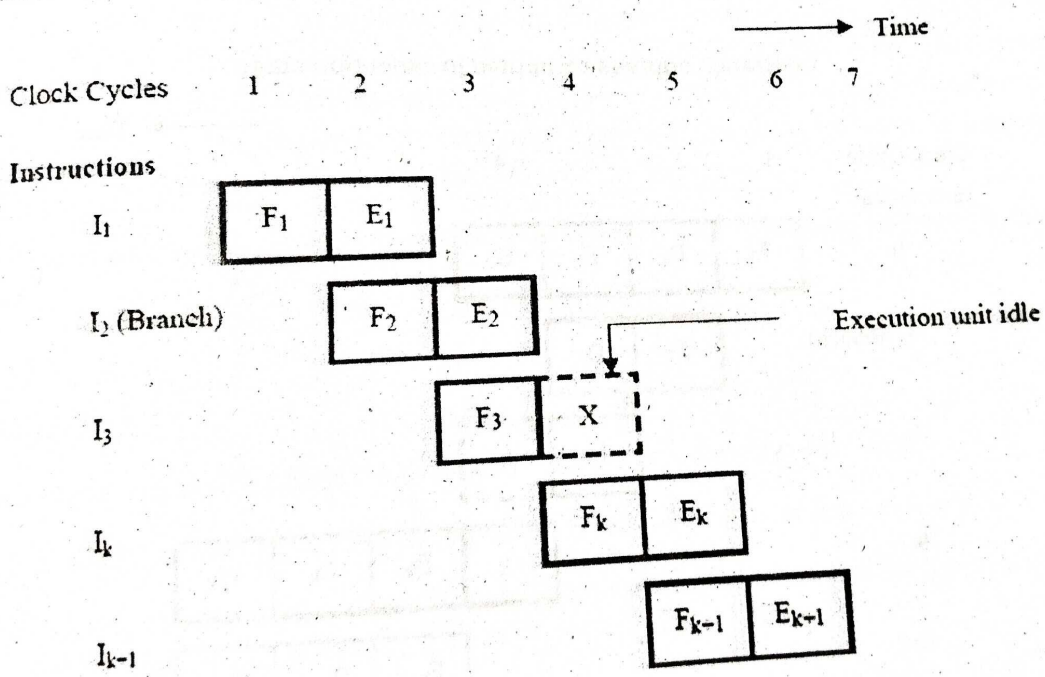
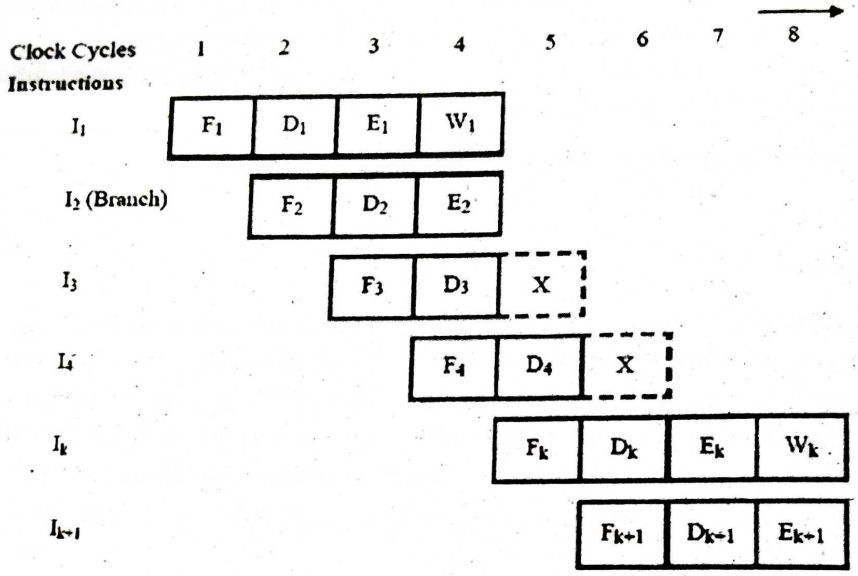


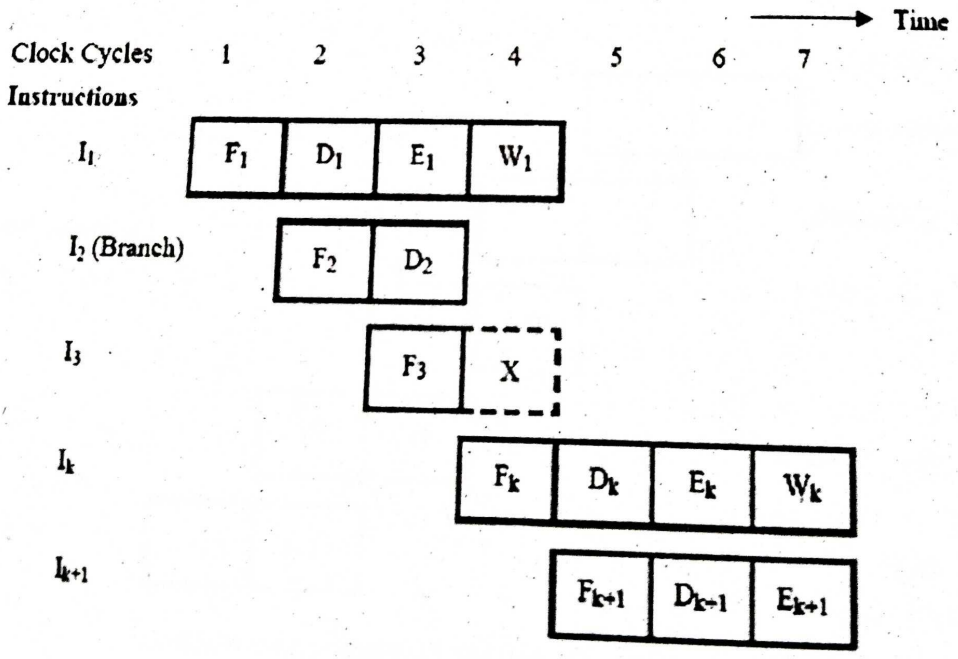
Fig.: An idle cycle caused by a branch instruction

The time lost as a result of a branch instruction is often referred to as the branch penalty (Time loss). In Figure, the branch penalty is one clock cycle. For a longer pipeline, the branch penalty may be higher. For example, Figure a shows the effect of a branch instruction on a four stage pipeline. The branch address is computed in step E2. Instructions I3 and I4 must be discarded, and the target instruction, Ik, is fetched in clock cycle 5. Thus, the branch penalty is two clock cycles.

Reducing the branch penalty requires the branch address to be computed earlier in the pipeline. Typically, the instruction fetch unit has dedicated hardware to identify a branch instruction and compute the branch target address as quickly as possible after an instruction is fetched. With this additional hardware, both of these tasks can be performed in step D2, leading to the sequence of events shown in Figure b. In this case, the branch penalty is only one clock cycle.



(a) Branch address computed in execution stage



(b) Branch address computed in decode stage

Fig.: Branch timing

Either a cache miss or a branch instruction stalls the pipeline for one or more clock cycles. To reduce effect of these interruptions, many processors employ sophisticated fetch units that can fetch instructions re they are needed and put them in a queue. Typically, the instruction queue can store several

instructions. A separa and sends them to th also performs the de processing capability queue filled at all tim delay in fetching ins instructions from the queue.

To be effe and execute bra impact of occasi of a branch or a The fetch unit c

Q18. What is in detai

Ans :

Pipeline Con

- There a
- Resourc
- Data de but this
- Branch

Data Deper

- A collis certain
- A data

instructions. A separate unit, which we call the dispatch unit, takes instructions from the front of the queue and sends them to the execution unit. This leads to the organization shown in Figure. The dispatch unit also performs the decoding function. To be effective, the fetch unit must have sufficient decoding and processing capability to recognize and execute branch instructions. It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions. If there is a delay in fetching instructions because of a branch or a cache miss, the dispatch unit continues to issue instructions from the instruction queue. The fetch unit continues to fetch instructions and add them to the queue.

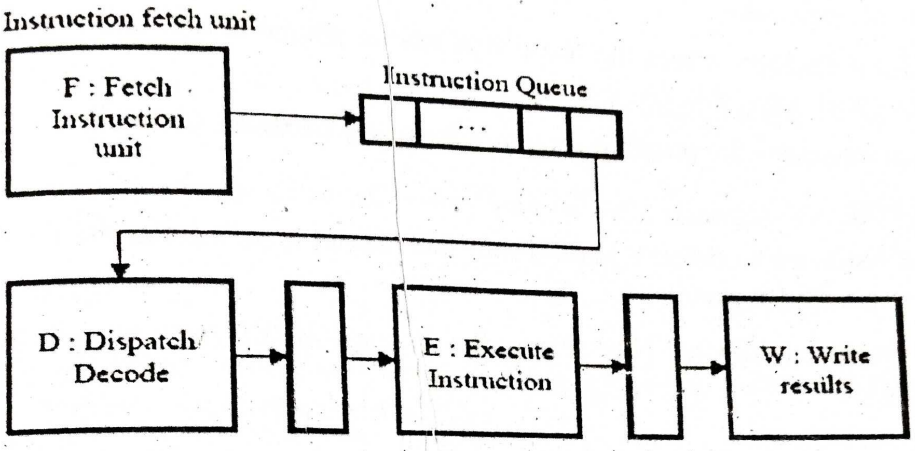


Fig.: Use of instruction queue in hardware organization.

To be effective, the fetch unit must have sufficient decoding and processing capability to recognize and execute branch instructions. It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions. If there is a delay in fetching instructions because of a branch or a cache miss, the dispatch unit continues to issue instructions from the instruction queue. The fetch unit continues to fetch instructions and add them to the queue.

Q18. What is pipeline conflict? Explain data dependency and handling of branch instruction in detail.

Ans :

Pipeline Conflict

- There are three major difficulties that cause the instruction pipeline conflicts.
- Resource conflicts caused by access to memory by two segments at the same time.
- Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
- Branch difficulties arise from branch and other instructions that change the value of PC.

Data Dependency

- A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations.
- A data dependency occurs when an instruction needs data that are not yet available.

- Similarly, an address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available.
- Pipelined computers deal with such conflicts between data dependencies in a variety of ways as follows.

Hardware Interlocks

- An interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline.
- Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict.
- This approach maintains the program sequence by using hardware to insert the required delays.

Operand Forwarding

- It uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments.
- This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.

Delayed Load

- Sometimes compiler has the responsibility for solving data conflicts problems.
- The compiler for such computers is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operation instruction, this method is called delayed load.

Handling of Branch Instructions

- One of the major problems in operating an instruction pipeline is the occurrence of branch instructions.
- A branch instruction can be conditional or unconditional.
- The branch instruction breaks the normal sequence of the instruction stream, causing difficulties in the operation of the instruction pipeline.
- Various hardware techniques are available to minimize the performance degradation caused by instruction branching.

Pre-fetch target

- One way of handling a conditional branch is to prefetch the target instruction in addition to the instruction following the branch.
- If the branch condition is successful, the pipeline continues from the branch target instruction.
- An extension of this procedure is to continue fetching instructions from both places until the branch decision is made.

Branch Target Buffer

- Another possibility is the use of a branch target buffer or BTB.

- The BTB is an associative memory included in the fetch segment of the pipeline.
- Each entry in the BTB consists of the address of a previously executed branch instruction and the target instruction for that branch.
- It also stores the next few instructions after the branch target instruction.
- The advantage of this scheme is that branch instructions that have occurred previously are readily available in the pipeline without interruption.

Loop Buffer

- A variation of the BTB is the loop buffer. This is a small very high speed register file maintained by the instruction fetch segment of the pipeline.
- When a program loop is detected in the program, it is stored in the loop buffer in its entirety, including all branches.

Branch Prediction

- A pipeline with branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed.
- The pipeline then begins pre-fetching the instruction stream from the predicted path.
- A correct prediction eliminates the wasted time caused by branch penalties.

Delayed Branch

- A procedure employed in most RISC processors is the delayed branch.
- In this procedure, the compiler detects the branch instructions and rearranges the machine language code sequence by inserting useful instructions that keep the pipeline operating without interruptions.

3.2.4 Influence on Instruction Sets

Q19. Explain, influence of instruction sets in pipeline. **(Imp.)**

Ans :

Influence on Instruction Sets

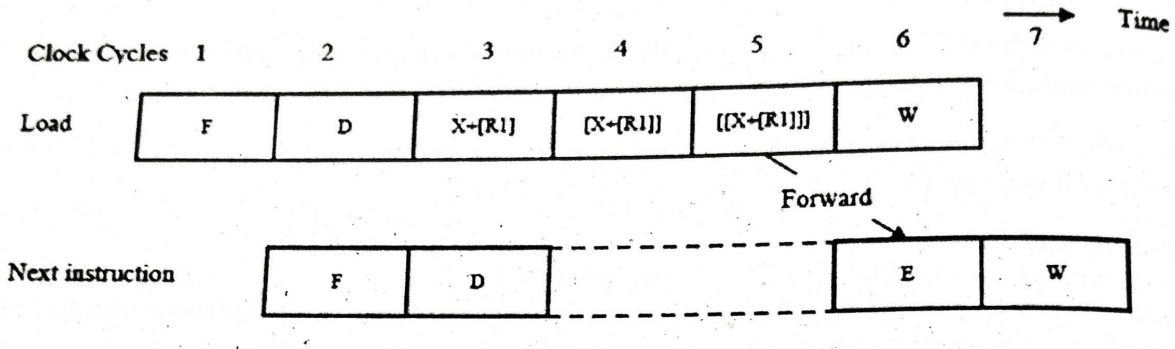
Some instructions are much better suited to pipelined execution than other instructions. For example, instruction side effects can lead to undesirable data dependencies. The machine instructions are influenced by addressing modes and condition code flags.

I. Addressing Modes

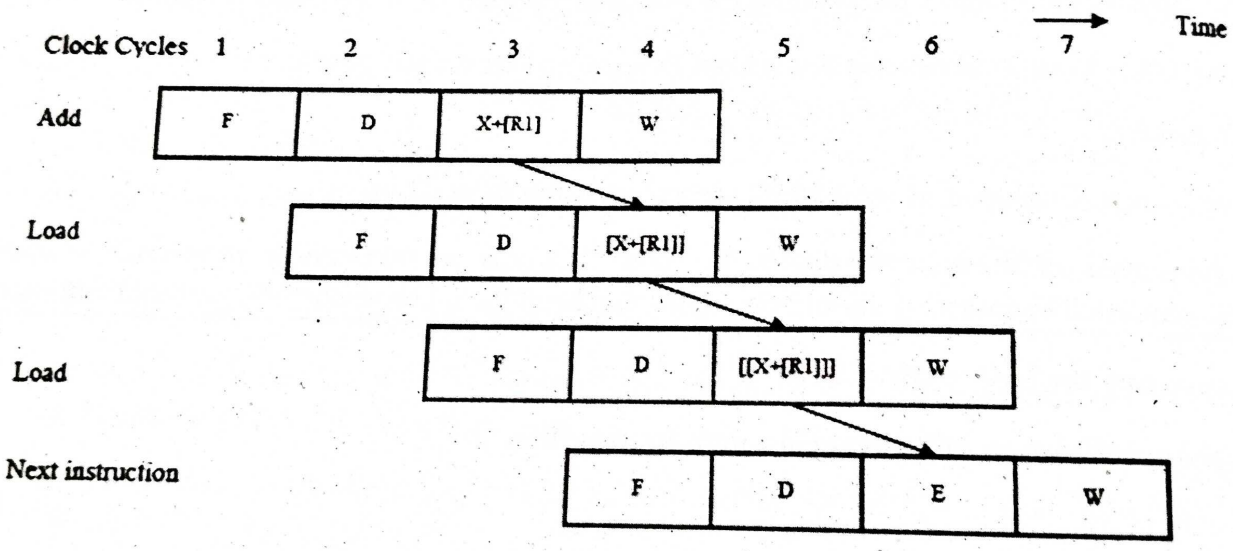
Addressing modes should provide the means for accessing a variety of data structures simply and efficiently. Useful addressing modes include index, indirect, autoincrement, and autodecrement. Many processors provide various combinations of these modes to increase the flexibility of their instruction sets. Complex addressing modes, such as those involving double indexing, are often encountered.

Two important considerations in this regard are the side effects of addressing modes such as autoincrement and autodecrement and the extent to which complex addressing modes cause the pipeline to stall. Another important factor is whether a given mode is likely to be used by compilers.

Assume a simple model for accessing operands in the memory. The load instruction Load X(RI),R2 takes five cycles to complete execution, However, the instruction Load (RI),R2 can be organized to fit a four-stage pipeline because no address computation is required. Access to memory can take place in stage E. A more complex addressing mode may require several accesses to the memory to reach the named operand.



(a) Complex addressing mode



(b) Simple addressing mode

Fig.: Equivalent operations using complex and simple addressing mode

For example, the instruction Load (X(RI)),R2

It may be executed as shown in Figure (a), assuming that the index offset, X, is given in the instruction, word. Mter computing the address in cycle 3, the processor needs to access memory twice - first to read location X [RI] in clock cycle 4 and then to read location [X [RI]] in cycle 5. If R2 is a source operand in the next instruction, that instruction would be stalled for three cycles, which can be reduced to two cycles with operand forwarding, as shown in figure . To implement the same Load operation using only simple addressing modes requires several instructions. For example, on a computer that allows three operand addresses like

```
Add #X,Rl,R2
Load (R2),R2
Load (R2),R2
```

The Add instruction performs the operation R2 ← X [RI], The two Load instructions fetch the address and then the operand from the memory. This sequence of instructions takes exactly the same number of clock cycles as the original, single Load instruction, as shown in Figure (b).

This example indicates that, in a pipelined processor, complex addressing modes that involve several accesses to the memory do not necessarily lead to faster execution. The main advantage of such modes is that they reduce the number of instructions needed to perform a given task and thereby reduce the program space needed in the main memory. Their main disadvantage is that their long execution times cause the pipeline to stall, thus reducing its effectiveness. They require more complex hardware to decode and execute them. Also, they are not convenient for compilers to work with.

The instruction sets of modem processors are designed to take maximum advantage of pipelined hardware. The addressing modes used in modem processors often have the following features:

- Access to an operand does not require more than one access to the memory.
- Only load and store instructions access memory operands.
- The addressing modes used do not have side effects.

Three basic addressing modes that have these features are register, register indirect, and index. The first two require no address computation. In the index mode, the address can be computed in one cycle, whether the index value is given in the instruction or in a register. Memory is accessed in the following cycle. None of these modes has any side effects, with one possible exception. Some architecture, such as ARM, allow the address computed in the index mode to be written back into the index register. This is a side effect that would not be allowed under the guidelines above.

II. Condition Modes

In many processors, the condition code flags are stored in the processor status register. They are either set or cleared by many instructions, so that they can be tested by subsequent conditional branch instructions to change the flow of program execution. An optimizing compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or data dependencies between successive instructions occur. In doing so, the compiler must ensure that reordering does not cause a change in the outcome of a computation. The dependency introduced by the condition-code flags reduces the flexibility available for the compiler to reorder instructions. Consider the sequence of instructions

```
Add Rl,R2
Compare R3,R4
Branch=0 ...
```

Assume that the execution of the Compare and Branch = 0 instructions proceeds as in Figure. The branch decision takes place in step E3 rather than D2 because it must await the result of the Compare instruction. The execution time of the Branch instruction can be reduced by interchanging the Add and Compare instructions,

Compare R3,R4

Add Rl,R2

Branch=0 ...

This will delay the branch instruction by one cycle relative to the Compare instruction. As a result, at the time the Branch instruction-'is being decoded the result of the Compare instruction will be available and a correct branch decision will be made. There would be no need for branch prediction. However, interchanging the Add and Compare instructions can be done only if the Add instruction does not affect the condition codes.

3.2.5 Data Path And Control Consideration

Q20. Explain data path for pipelined execution architecture.

Ans: **(Imp.)**

Datapath and Control Considerations

Consider the three-bus structure suitable for pipelined execution with a slight modification to support a 4-stage pipeline as shown in figure.

Several important changes are

1. There are separate instruction and data caches that use separate address and data connections to the processor. This requires two versions of the MAR register, IMAR for accessing tile instruction cache and DMAR for accessing the data cache.
2. The PC is connected directly to the IMAR, so that the contents of the PC can be transferred to IMAR at the same time that an independent ALU operation is taking place.
3. The data address in DMAR can be obtained directly from the register file or from the ALU to support the register indirect and indexed addressing modes.
4. Separate MDR registers are provided for read and write operations. Data can be transferred directly between these registers and the register file during load and store operations without the need to pass through the ALU.
5. Buffer registers have been introduced at the inputs and output of the ALU. These are registers SRCl, SRC2, and RSLT. Forwarding connections may be added if desired.
6. The instruction register has been replaced with an instruction queue, which is loaded from the instruction cache.
7. The output of the instruction decoder is connected to the control signal pipeline. This pipeline holds the control signals in buffers B2 and B3.

The following operations can be performed independently in the processor of Figure:

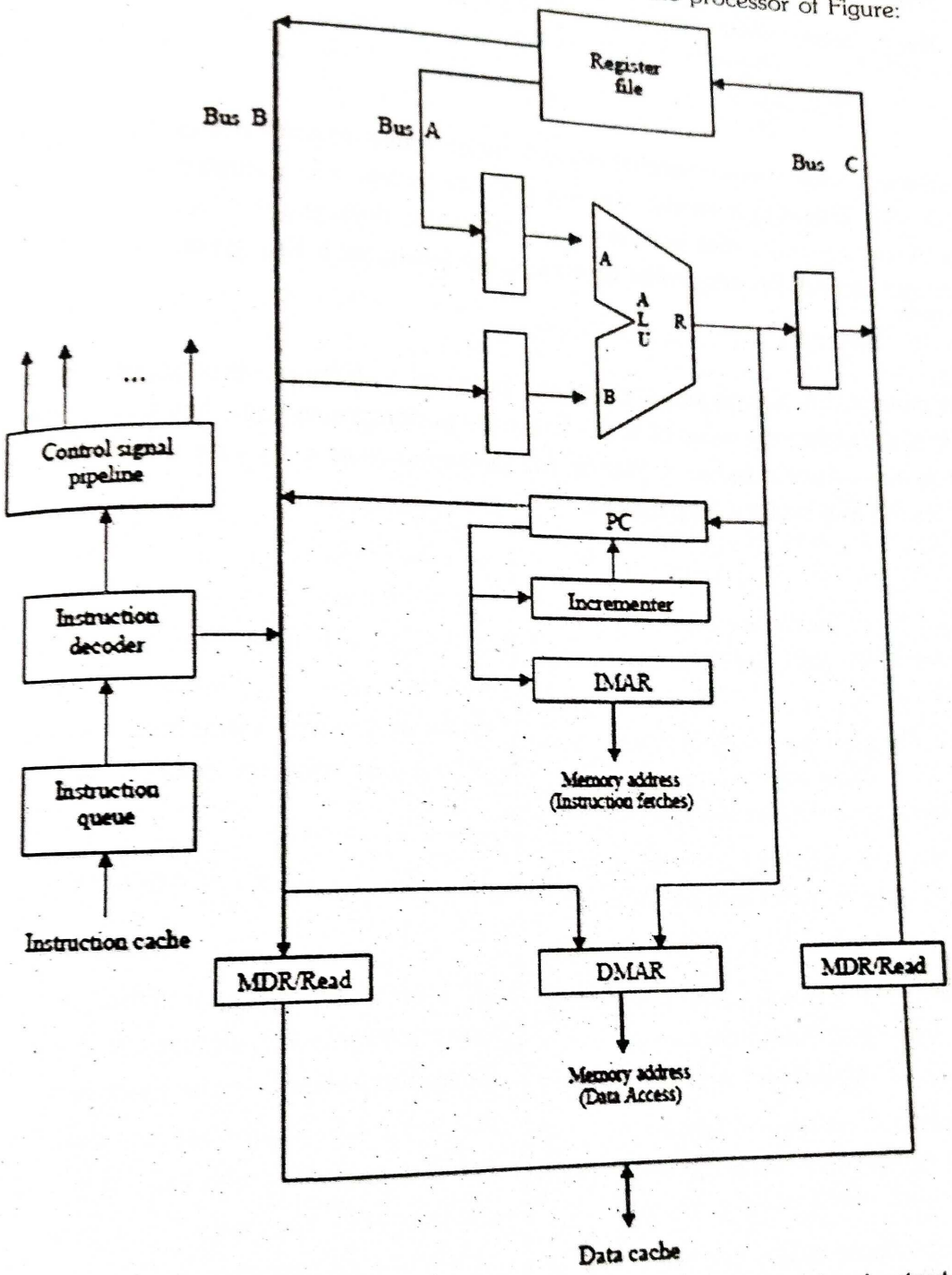


Fig.: Datapath modified for pipelined execution with interstage buffers at the input and output of the ALU.

- Reading an Instruction from the instruction cache
- Incrementing the PC
- Decoding an instruction
- Reading from or writing into the data cache
- Reading the contents of up to two registers from the register file
- Writing into one register in the register file

➢ Performing an ALU operation

The processor execution time T, of a program that has a dynamic instruction count N is given by

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

Where S is the average number of clock cycles it takes to fetch and execute one instruction and R is the clock rate. This simple model assumes that instructions are executed one after the other, with no overlap. A useful performance indicator is the instruction throughput, which is the number of instructions executed per second. For sequential execution, the throughput, Ps is given by

$$P_s = R/s$$

In general, an n-stage pipeline has the potential to increase throughput n times. Thus, it would appear that the higher the value of n, the larger the performances gain. Any time a pipeline is stalled, the instruction throughput is reduced. Hence, the performance of a pipeline is highly influenced by factors such as branch and cache miss penalties.

Short Question and Answers

1. Branch Instruction

Ans :

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.

2. Hardwired Control

Ans :

To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. Computer designers use a wide variety of techniques to solve this problem. The approaches used fall into one of two categories: hardwired control and micro-programmed control. The required control signals are determined by the following information:

i) Contents of the control step counter

ii) Contents of the instruction register

iii) Contents of the condition code flags

iv) External input signals, such as MFC and interrupt requests

3. Address Sequencing.

Ans :

Microinstructions are stored in control memory in groups, with each group specifying a routine.

To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

Step-1

- An initial address is loaded into the control address register when power is turned on in the computer.
- This address is usually the address of the first microinstruction that activates the instruction fetch routine.
- The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.
- At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2

- The control memory next must go through the routine that determines the effective address of the operand.
- A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.
- The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
- When the effective address computation routine is completed, the address of the operand is available in the memory address register.

Step-3

- The next step is to generate the microoperations that execute the instruction fetched from memory.
- The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
- Each instruction has its own micro-program routine stored in a given location of control memory.
- The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a mapping process.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.

Step-4

- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.
- Micro-programs that employ subroutines will require an external register for storing the return address.
- Return addresses cannot be stored in ROM because the unit has no writing capability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.

4. Mapping of an Instruction.

Ans :

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
- The status bits for this type of branch are the bits in the operation code part of the instruction.
- For example, a computer with a simple instruction format as shown in figure has an operation code of four bits which can specify up to 16 distinct instructions.
- Assume further that the control memory has 128 words, requiring an address of seven bits.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.

Q5. What is Symbolic Microinstruction.

Ans :

Each line of the assembly language microprogram defines a symbolic microinstruction.

Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD.

Q6. Microprogram Sequencer

Ans :

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a microprogram sequencer.
- A microprogram sequencer can be constructed with digital functions to suit a particular application.
- To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range of applications.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutine calls.
- Some sequencers provide an output register which can function as the address register for the control memory.

7. Explain pipelining

Ans :

Pipeline is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- Each segment performs partial processing dictated by the way the task is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

8. Data Hazards

Ans :

A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result some operation has to be delayed, and the pipeline stalls. A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason.

9. Instruction Hazards

Ans :

Pipeline execution of instructions will reduce the time and improves the performance. Whenever this stream is interrupted, the pipeline stalls, as figure illustrates for the case of a cache miss. A branch instruction may also cause the pipeline to stall. The effect of branch instructions and the techniques that can be used for mitigating their impact are discussed with unconditional branches and conditional branches.

10. Pipeline conflict.

Ans :

- There are three major difficulties that cause the instruction pipeline conflicts.
- Resource conflicts caused by access to memory by two segments at the same time.
- Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
- Branch difficulties arise from branch and other instructions that change the value of PC.

Choose the Correct Answers

Any condition that causes a processor to stall is called as _____ [a]

(a) Hazard (b) Page fault

(c) System error (d) None of the mentioned

The periods of time when the unit is idle is called as _____ [d]

(a) Stalls (b) Bubbles

(c) Hazards (d) Both Stalls and Bubbles

The stalling of the processor due to the unavailability of the instructions is called as [a]

(a) Control hazard (b) structural hazard

(c) Input hazard (d) None of the mentioned

The fetch and execution cycles are interleaved with the help of _____ [b]

(a) Modification in processor architecture (b) Clock

(c) Special unit (d) Control unit

_____ are the different type/s of generating control signals. [d]

(a) Micro-programmed (b) Hardwired

(c) Micro-instruction (d) Both Micro-programmed and Hardwired

A word whose individual bits represent a control signal is _____ [b]

(a) Command word (b) Control word

(c) Co-ordination word (d) Generation word

Highly encoded schemes that use compact codes to specify a small number of functions in each micro instruction is _____ [b]

(a) Horizontal organisation (b) Vertical organisation

(c) Diagonal organisation (d) None of the mentioned

Individual control words of the micro routine are called as _____ [c]

(a) Micro task (b) Micro operation

(c) Micro instruction (d) Micro command

The ALU performs the indicated operation on the operands prepared in the prior cycle and store the result in the specified destination operand location. [c]

(a) Fetch instruction (b) Decode instruction

(c) Execute instruction (d) Fetch operand

The special memory used to store the micro routines of a computer is _____ [b]

(a) Control table (b) Control store

(c) Control mart (d) Control shop

Fill in the Blanks

1. The disadvantage/s of the hardwired approach is _____
2. In micro-programmed approach, the signals are generated by _____
3. A sequence of control words corresponding to a control sequence is called _____
4. The situation wherein the data of operands are not available is called _____
5. The time lost due to the branch instruction is often referred to as _____
6. _____ method is used in centralized systems to perform out of order execution.
7. _____ compilers have been developed specifically for pipelined systems.
8. To increase the speed of memory access in pipelining, we make use of _____
9. _____ instruction starts a new instruction fetch cycle and resets the counter
10. Register renaming is done in pipelined processors to _____

Answers

1. Less flexible & cannot be used for complex instructions
2. Machine instructions
3. Micro routine
4. Data hazard
5. Branch penalty
6. Redundancy
7. Optimizing compilers
8. Cahce
9. End
10. To eliminate certain kinds of hazards