2 Module 2

Processor Organization and Architecture

Syllabus

CPU Architecture, Register Organization, Instruction cycle, Instruction Formats, Control Unit Design-Hardwired and Micro-programmed Control : Vertical and Horizontal Micro-Instructions, Nano-programming, Comparison between CISC and RISC architectures.

2.1 CPU Architecture and Register Organization

University Question

Describe the register organization within the CPU.

MU - May 16, 10 Marks

- Fig. 2.1.1 shows the architecture of microprocessor.
- This architecture is divided in different groups as follows :
  1. Registers
  2. Arithmetic and logic unit
  3. Interrupt control
  4. Timing and control circuitry.

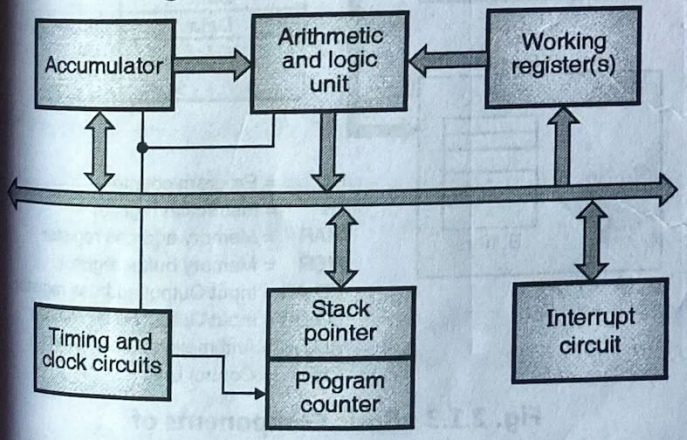

Fig. 2.1.1 : General architecture of a microprocessor

- It consists of PIPO (Parallel in parallel out) register as shown in Fig. 2.1.2.
- This section is also called as scratch pad memory. It stores data and address of memory.
- The register organization affects the length of program, the execution time of program and simplification of the program.
- To achieve better performance, the Number of registers should be large.
- The architecture of microcomputer depends upon the Number and type of the registers used in microprocessor.
- It consists 8-bit registers or 16 bit registers.
- The register section varies from microprocessor to microprocessor.
- The registers are used to store the data and address.
- These registers are classified as :
  (a) Temporary registers
  (b) General purpose registers
  (c) Special purpose registers.

1. Register section

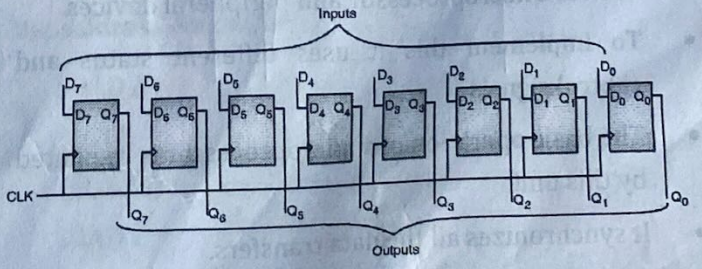

Fig. 2.1.2 : 8 bit register

2. Arithmetic and logical unit

- This section processes data i.e. it performs arithmetic and logical operations.
- It performs arithmetic operations like addition, subtraction and logical operations like ANDing, ORing, EX-ORing, etc.
- The ALU is not available to the user. Its word length depends upon the width of an internal data bus.
- The ALU is controlled by timing and control circuits.
- It accepts operands from memory or register. It stores result of arithmetic and logic operations in register or memory.
- It provides status of result to the flag register. Flag register shows status of result.
- ALU looks after the branching decisions.

3. Interrupt control

- This block accepts different interrupt request inputs.
- When a valid interrupt request is present it informs control logic to take action in response to each signal.

4. Timing and control unit

- This is a control section of microprocessor made up of synchronous sequential logic circuit.
- It controls all internal and external circuits.
- It operates with reference to clock signal.
- This accepts information from instruction decoder and generates micro steps to perform it.
- In addition to this, the block accepts clock inputs, performs sequencing and synchronizing operations.
- The synchronization is required for communication between microprocessor and peripheral devices.
- To implement this it uses different status and control signals.
- The basic operation of a microprocessor is regulated by this unit.
- It synchronizes all the data transfers.
- This unit takes appropriate actions in response to external control signals.

2.1.1 Instruction Formats

University Question

Q. Give different instruction formats.

MU - Dec. 18, May 19, Dec. 19, 5 Marks

- The Control Unit and the ALU (Arithmetic and Logic Unit) along with some registers constitute the Central Processing Unit.
- Fig. 2.1.3 shows the basic components of the computer and their interconnection.
- Also the internal components of the CPU are shown in the Fig. 2.1.3.
- The computer consists of three basic components namely the CPU, memory and I/O devices connected with each other via the buses.
- Input devices are required to give the instructions and data to the system.
- The output devices are used to give the output devices.
- The instructions and the data given by the input device are to be stored, and for storage we require memory.

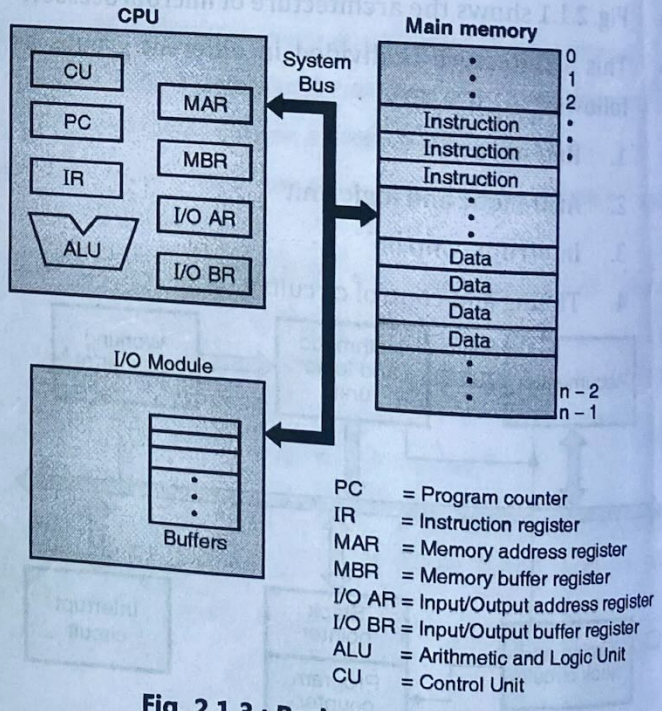

Fig. 2.1.3 : Basic Components of the computer and the CPU

- The memory module receives address and control signals (like read, write, timing etc) and also receives and sends data.

- The memory is assumed to be of 'n' locations with the addresses from 0 to n – 1.
- CPU module reads instruction and data, writes the data and also sends the control signals.
- It also receives interrupts from the I/O devices and acts accordingly.
- The CPU has the control unit that provides control signals to all the resources inside and outside the CPU.
- The CPU also has the ALU that performs the arithmetic and logical operations.
- The CPU also has some registers as seen in the Fig. 2.1.3.
- The register PC (Program Counter) is a register that always points to the next instruction to be executed.
- Hence the word "Program" in the name.
- It has to increment always to point to the next instruction, which is automatically done. Hence the word "Counter" in the name.
- The MAR (Memory Address Register) is used to store the address to be provided to the memory.
- And MBR (Memory Buffer Register) is used to store the data to be given or taken from the memory.
- Similarly for I/O devices we have IOAR and IOBR.
- Another register named as IR (Instruction Register) is used to store the instruction to be executed by the CPU.
- The use of these registers will be further seen in the next section named as Instruction Cycle.

2.1.2 Instruction Word Format - Number of Addresses

Elements of an Instruction

1. Operation code (Opcode) is that part of the instruction which gives the code for the operation to be performed.
2. Source Operand reference or address 1, gives the reference of the data on which the operation is to be performed. This address could be a register, memory or an input device.
3. Source Operand reference or address 2, gives the reference of the second data on which the operation is to be performed. This address could again be a register, memory or an input device.
4. Result Operand reference gives the reference where the result after performing operation is to be stored. The result could be stored in the register, memory or given to an output device.
5. An instruction may have only one address with the other two fixed, or may have two addresses with one of the source operand address as the result operand address. Hence the instruction can have one, two or three addresses.

- Fig. 2.1.4 shows an example of a simple instruction format with one and two addresses.

Opcode	Operand Address 1

(a) Single address instruction format

Opcode	Operand Address 1	Operand Address 2

(b) Two address instruction format

Fig. 2.1.4 : Instruction Word Formats

- Three address, One address and Zero address instructions.

Zero address instructions

```
PUSH A  ;  ToS <- A
PUSH B  ;  ToS <- B
ADD     ;  ToS <- A + B
PUSH C  ;  ToS <- C
PUSH D  ;  ToS <- D
ADD     ;  ToS < C + D
MUL     ;  ToS <- (A + B)*(C + D)
PUSH E  ;  ToS <- E
PUSH F  ;  ToS <- F
SUB     ;  ToS <- (E – F)
DIV     ;  ToS <- (A + B)*(C + D)/(E – F)
POP X   ;  M[X] <- ToS
```

One address instructions

```
LOAD A  ;  AC <- M[A]
ADD B   ;  AC <- AC + M[B]
STORE P ;  M[P]<- AC
LOAD C  ;  AC<- M[C]
ADD D   ;  AC <- AC + M[D]
MUL P   ;  AC <- AC * M[P]
```

```
i.e. AC <- (A + B) * (B + C)
STORE P  ;  M[P] <- AC
LOAD E   ;  AC <- M[E]
SUB F    ;  AC<- AC – M[F]
STORE Q  ;  M[Q] <- AC
LOAD P   ;  AC <- M[P]
DIV Q    ;  AC <- AC / M[Q]
i.e. AC <- (A + B)*(C+D)/(E – F)
STORE X  ;  M[X] <- X
i.e. X<- (A + B)*(C + D)/(E – F)
```

Accumulator type one address format

```
LOAD A   ;  AC <- M[A]
MUL B    ;  AC <- AC * M[B]
STORE P  ;  M[P]<- AC
LOAD C   ;  AC<- M[C]
MUL D    ;  AC <- AC * M[D]
SUB E    ;  AC<- AC – M[E]
ADD P    ;  AC <- AC + M[P]
i.e. AC <- (A*B) + (C * D – E)
STORE P  ;  M[P] <- AC
LOAD A   ;  AC <- M[A]
ADD B    ;  AC<- AC + M[B]
STORE Q  ;  M[Q] <- AC
LOAD P   ;  AC <- M[P]
DIV Q    ;  AC <- AC / M[Q]
i.e. AC <- ((A*B) + (C*D – E)) / (A + B)
STORE X  ;  M[X] <- X I.E. X<- ((A*B)
            + (C*D – E)) / (A + B)
```

Three address instructions

```
((A*B) + (C*D – E)) / (A + B)
ADD R1,A,B    ;  R1<- M[A] + M[B]
MUL R2,C,D    ;  R2<- M[C] * M[D]
SUB R2,R2,E   ;  R2<- R2 – M[E]
ADD R1,R1,R2  ;  R1< R1 + R2 i.e. R1< ((A*B)
                 + (C*D – E))
ADD R2,A,B    ;  R2 <- A + B
DIV X,R1,R2   ;  M[X] <- R1/ R2 i.e. X < ((A*B)
                 + (C*D – E))/ (A + B)
```

2.1.3 Reverse Polish Notation

- Reverse Polish Notation (RPN) is a mathematical notation wherein the operator follows all of its operands.
- The RPN is also known as Postfix notation and is parenthesis-free.
- The polish notation use to follow the sequence of the operator followed by its operands, and hence the reverse polish notation gets its name.
- The name Polish notation is derived from its origin country.
- The Reverse Polish notation was proposed in 1954 by Burks, Warren, and Wright and was independently reinvented by F. L. Bauer and E. W. Dijkstra in the early 1960s.
- This notation helps to reduce computer memory access and to utilize the stack to evaluate expressions.
- During the 1970s and 1980s, RPN was even known to the general public, as it was widely used in calculators.
- In Reverse Polish notation the operators follow their operands.
- This means that the operands are mentioned first and then their operator, for example, to add 3 and 4, one would write "3 4 +" rather than "3 + 4".
- If there are multiple operations, then the operator is given immediately after its second operand, for example the expression written "3 – 4 + 5" in conventional infix notation would be written "3 4 – 5 +" in RPN.
- An advantage of RPN is that it removes the need for parentheses.

2.1.4 Basic Instruction Cycle

University Question

Q. Explain instruction and instruction cycle.

MU - Dec. 18, May 19, 5 Marks

- The instruction cycle is a representation of the states that the computer or the microprocessor performs when executing an instruction.
- The instruction cycle comprises of two main steps to be followed to execute the instruction, namely the

fetch operation in the fetch cycle and the execution operation during the execute cycle.

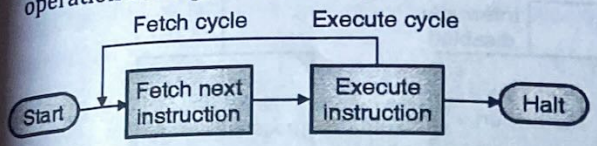

Fig. 2.1.5 : Basic instruction cycle

- Fig. 2.1.5 shows the basic instruction cycle. It comprises of the fetch and executes cycle in a loop to execute huge Number of instructions, until it reaches the halt instruction.
- The fetch cycle comprises of the following operations :

1. Program Counter (PC) holds address of next instruction to fetch; hence the CPU (Processor) fetches instruction from memory location pointed to by PC.
2. This is done by providing the value of the PC to the MAR and giving the Read control signal to the memory.
3. On this the memory provides the value in the given address (which is the instruction) to MBR.
4. The PC value has to be incremented to point to the next instruction (Sometimes the value of PC may have be completely changed in case of some special instructions called as branching instructions).
5. The instruction is loaded into Instruction Register (IR) from the MBR.
6. Finally the processor interprets or decodes the instruction. The processor performs required operations in the execute cycle.

- In the execute cycle the operation asked to be performed by the instruction is done.
- It may comprise of one or more of the following operations :

1. Transfer of data between processor and memory or between processor and IO module.
2. Processing of data like some arithmetic or logical operations on data.
3. Change of the sequence of operation i.e. branching instructions.

2.1.5 Interrupt Cycle

- Fetch and execute are not the only two states in the instruction cycle.
- There is one more state i.e. Interrupt cycle.
- In this subsection we will see the concept of interrupt in short and the interrupt cycle.
- Interrupt is a mechanism by which I/O modules can interrupt normal sequence of processing.
- Interrupt can be because of some request from an I/O device to service that particular device.
- This service may be take or give data or some control operation.
- It may also because of some unexpected operation in the program execution by the CPU itself.
- Interrupt cycle as discussed earlier is added to instruction cycle.
- During this cycle the processor checks for interrupt, and if present and enabled services the same.
- If no interrupt is present then it fetches the next instruction else if interrupt pending then it performs the following operations :

1. Suspend the execution of current program.
2. Save the context of the current program under execution.
3. Set the PC value to start address of interrupt handler routine also called as interrupt service routine. Interrupt service routine is a small program which when executed, services the interrupting source.
4. Process the Interrupt Service Routine (ISR) and then.
5. Restore the context and continue execution of the interrupted program.

- Thus the complete basic instruction cycle with Interrupts can be as shown in the Fig. 2.1.6.

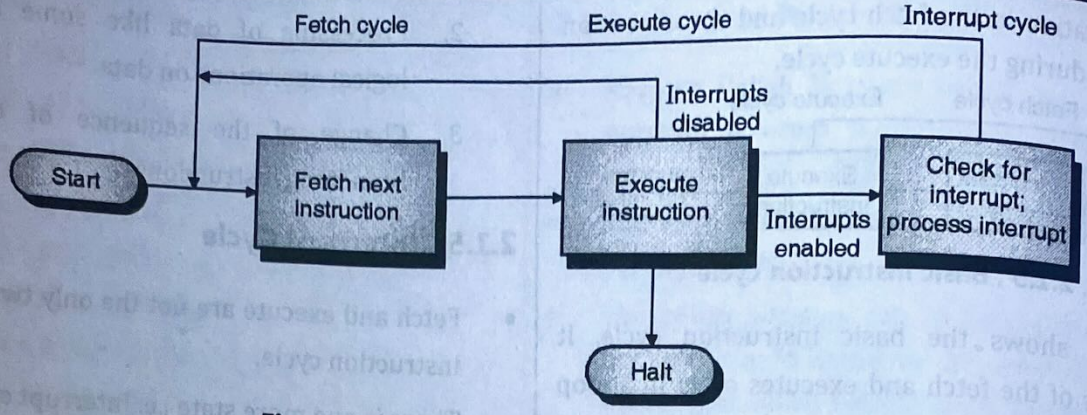

Fig. 2.1.6 : Complete basic instruction cycle

- You will notice in Fig. 2.1.6, the interrupts are checked for, after the execute cycle and processed if enabled and exist; else, it fetches the next instruction.
- The detailed instruction cycle is shown in Fig. 2.1.7.

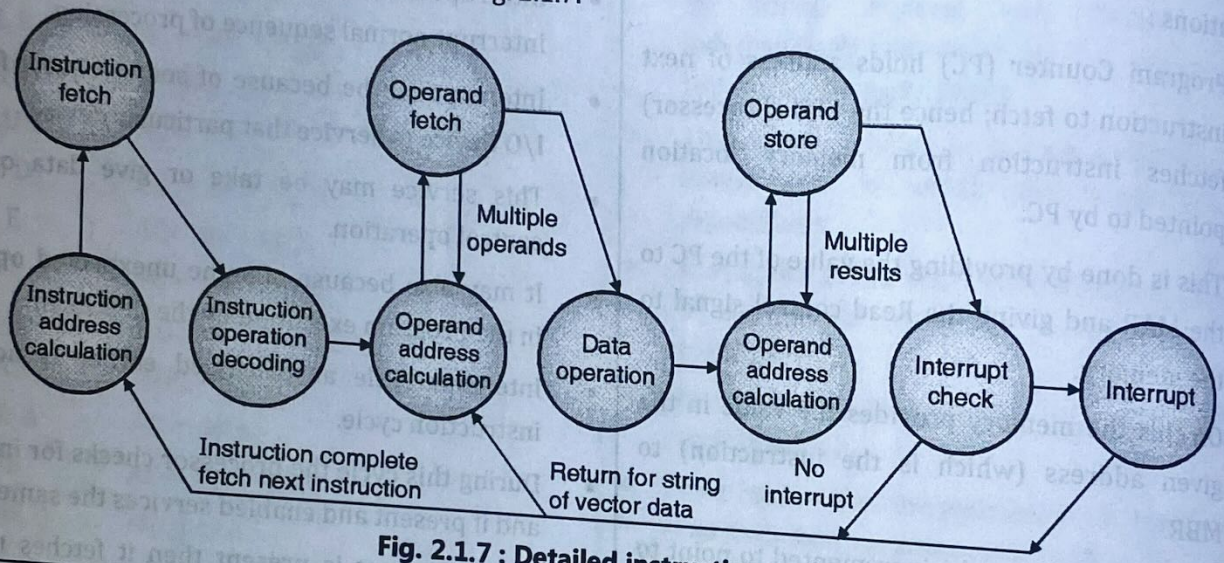

Fig. 2.1.7 : Detailed instruction cycle

- In Fig. 2.1.7, there are some states drawn on the upper side, while some on the lower side.
- The ones on the upper side are the operations carried out on the buses or are external operations, while the ones at the lower level are the operations carried out inside the CPU or are internal operations.
- The instruction cycle begins from the "Instruction address calculation" state, wherein the address of the next instruction is calculated or the value of the PC is updated.
- Then the instruction is fetched, which requires the operation on the buses.
- The instruction fetched is then decoded. Until this state, it is the fetch cycle.
- In the execute cycle, the operand address is calculated and the operands are fetched from the calculated address.
- Again to fetch the operands, we require the buses.
- After fetching the operand, if more operands are required for multiple operand instructions, then the next state is again calculate the operand address i.e. the address of the next operand.
- Once all the operands are fetched, the data operation is carried out as per the operation indicated in the instruction.
- Now for the result storage again the address of operand is calculated and the result is stored in the specified location of the memory.
- In case of multiple operands again the calculation and storage process for the operand continues until all the operands are stored.

- Now begins the interrupt cycle, wherein the first step is to check the presence of an enabled interrupt.
- If there is none, then the next state as seen in the Fig. 2.1.7, is the calculation of next instruction address i.e. execute the next sequential instruction.
- But in case the interrupt is present and enabled then the servicing of the same is done as discussed earlier in this section.
- In the Fig. 2.1.7, you will also notice that there are two paths from the end of the previous instruction.
- The one that goes to the state "Instruction address calculation" for the next instruction; and the one that goes to the "Operand address calculation" for vector instructions.
- Vector instructions are those instructions wherein the operation is same but the data on which the operation is to be performed in a huge block of data or an array of data.
- Hence in the second case, the instruction is already fetched and decoded i.e. the operation is already know, and the operation is to be performed on a block of data.
- After completing the operation on one set of operands, the CPU returns to the next operand address calculation state, wherein it calculates and fetches the next operand.
- Then it performs the operation, stores the result and again for the next set of operand, until all the operands in the array are completed.

2.2 Instruction Interpretation and Sequencing and Micro-Operations with their Sequencing

University Questions

Q. Explain microinstruction sequencing and execution. **MU - May 15, 7 Marks**

Q. Explain microinstructions to execute an instruction MOV [R1], R2. **MU - May 16, 6 Marks**

Q. Explain micro instruction sequencing and execution. **MU - Dec. 16, 10 Marks**

Q. What is Microprogram ? Write microprogram for following operations.

1. ADD R1, M ; Register R1 and Memory location M are added and result store at Register R1.
2. MUL R1, R2 ; Register R1 and Register R2 are multiple and result store at Register R1.

MU - Dec. 18, May 19, Dec. 19, 10 Marks

- The structure of the CPU seen in section 2.1.1 is shown in details in Fig. 2.2.1.
- This structure has a specialty that all the control signals are shown in it.
- Programs are executed as a sequence of instructions.
- As seen in the previous sections of this chapter, each instruction consists of a series of steps that make up the instruction cycle i.e. fetch, decode, etc.
- Each of these steps are, in turn, made up of a smaller series of steps called micro-operations or micro-instructions.
- Control signals are issued to perform these micro-operations and micro-instructions are these control signals.
- Fig. 2.2.1 shows the structure of the CPU with these micro-instructions or the control signals.

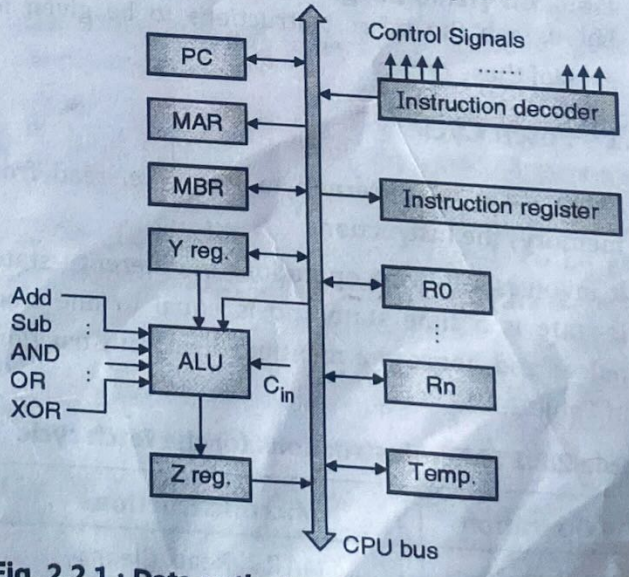

Fig. 2.2.1 : Data path structure with control signals

- It also shows those register as already seen in section 2.1.1 like PC, MAR, MBR, etc.
- There are some registers like the register 'Y' to provide one of the operand to the ALU as shown in the Fig. 2.2.1.

- Another register is the 'Z' register, which is used to store the result given by the ALU.
- A "temp" register or the temporary register to store some temporary data.
- The set of registers R0 to Rn (the value of 'n' depends on the registers in the CPU) for general purpose operations.
- There is also an instruction decoder for decoding the instructions stored in the instruction register and in turn provides the micro-instructions or the control signals for the resources inside and outside the CPU.
- The ALU also gets the control signals from this decoder indicating the operation to be performed like Add, Sub, etc.
- The ALU also has an extra input called as C_{in} i.e. the carry input as required for adder.
- To execute any instruction as seen earlier it is to be divided into three cycles viz. fetch, execute and interrupt cycles.
- The execute cycle will differ based on the operation to be carried out in the instruction, but the fetch and interrupt cycle will be common for all the cycles.
- Let us see the micro-instructions to be given for each of these cycles.

2.2.1 Fetch Cycle

- Fetch cycle is concerned to fetch (i.e. read from memory) the instruction.
- It involves following operations in different t-states (t-state is a time state and is equal to one clock pulse) and hence the mentioned microinstructions in Table 2.2.1.

Table 2.2.1 : Microinstructions for the fetch cycle

	Operation	Microinstructions
t1	$PC \rightarrow MAR$	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
t2	$M \rightarrow MBR$ $PC \leftarrow PC + 1$	Z_{out}, PC_{in}, Wait for memory fetch cycle
t3	$MBR \rightarrow IR$	MBR_{out}, IR_{in}

- As seen in the table, three clock pulses or t-states are required for the fetch cycle.
- Note, the control unit is an organizational part of the CPU, hence the design can vary from processor to processor.
- In the first t-state, the address of the instruction to be executed is given to the MAR register from the PC register.
- To perform this operation the control signals given are PC_{out} and MAR_{in}.
- This will make the PC register give out its data and the MAR register accept this data.
- Also the memory is indicated to perform a read operation from memory hence the signal "Read".
- To increment the value of PC, the various operations are performed on ALU signals i.e. Clear Y, Set C_{in}, Add, Z_{in}.
- The 'Y' register is cleared and the carry flag is set.
- Now when the ALU is said to perform the "ADD" operation it will add the contents of the 'Y' register, carry flag and the contents of the internal data bus.
- The contents of the internal data bus are nothing but the value given out by the PC register.
- Hence the PC is added with '1' i.e. the carry flag and hence incremented value of PC is given to the 'Z' register.
- In the second clock pulse the CPU has to wait for the memory operation, but in the same time it can transfer the result in 'Z' register to the PC register with the control signals namely Z_{out} and PC_{in}.
- This could not be done in the previous t-state, as two data cannot be given simultaneously on the data bus, else it will get mixed up.
- Only one data can be given on the data bus in any clock pulse, but as many as required can accept the data.
- In the final t-state, the contents received from the memory i.e. the instruction is transferred to its correct place i.e. the instruction register.
- This is done by the control signals namely MBR_{out} and IR_{in}.

- This also completes the entire fetch operation of the instruction.

2.2.2 Execute Cycle

- Execute cycle as discussed can be of various types based on the operation to be performed in the instruction and the location of the operand.
- We will see some examples in this subsection.
- The first example we will take for the execution of a direct addressed operand.
- In this case the address of the operand is directly given in the instruction.
- It involves different operations in various t-states as shown in Table 2.2.2 assuming the instruction ADD R1, [X].

Table 2.2.2 : Microinstructions for the execute cycle of direct addressed mode of operand access

	Operation	Microinstructions
t1	IR → MAR	IR_{out}(address), MAR_{in}, Read, Clear C_{in}
t2	M → MBR	$R1_{out}$, Y_{in}, Wait for memory read cycle
t3	MBR + R1 → R1	MBR_{out}, Add, Z_{in}
t4		Z_{out}, $R1_{in}$

- In this case of direct addressing mode, the address of the memory operand is in the instruction itself.
- The instruction as we have seen in the fetch cycle reaches the IR register.
- Hence the IR register is given a signal to give out the address part and the MAR register to accept this address input value by giving the control signals IR_{out}(address) and MAR_{in}.
- At the same time, since the memory is to be read from the control signal is given to the memory i.e. "Read".
- Also the carry flag is cleared to get ready for the addition operation.
- Since the instruction expects addition of the register 'R1' and the data at memory location with address 'X', the contents of register 'R1' are transferred to the 'Y' register, which is one of the operands for any ALU operation.
- To perform this transfer operation the control signals given are $R1_{out}$ and Y_{in}.
- Also by the end of the second t-state, the data operand required from the memory will be available in the MBR register.
- In the third t-state the contents of the MBR, which is the content of memory location with the address 'X', is placed on the internal data bus and the ALU is indicated to perform the addition operation.
- It adds the contents of the 'Y' register and the contents of the internal data bus, and the result is given to the 'Z' register.
- An extra t-state is required to send the data from the 'Z' register to the register R1, as seen earlier two data cannot be given simultaneously on the data bus in the same t-state.
- And the contents of memory location with the address 'X' are already put on the data bus in the third t-state.
- The fourth t-state is thus required to transfer the data from register 'Z' to register R1 using the signals Z_{out}, $R1_{in}$.
- Another execute cycle we will be studying in this sub-section is for the indirect addressed operand.
- In this case, the address given in the instruction is the memory location that contains the address of the operand.
- Table 2.2.3 shows the micro-operations required for such an execute cycle for an example instruction ADD R1, [[X]]
- Table 2.2.3 shows the control signals to be given exactly similar to that of the Table 2.2.2, with a minor difference i.e. the value received in the MBR on first memory read is the operand address and hence is to be given back to the memory to fetch the actual operand.

Table 2.2.3 : Microinstructions of the execute cycle of an indirect addressed operand instruction

	Operation	Microinstructions
t1	IR → MAR	IR_{out}(address), MAR_{in}, Read, Clear C_{in}
t2	M → MBR	$R1_{out}$, Y_{in}, Wait for memory read cycle

	Operation	Microinstructions
t3	MBR → MAR	MBR_{out}(address), MAR_{in}, Read
t4	M → MBR	Wait for memory read cycle
t5	MBR + R1 → R1	MBR_{out}, Add, Z_{in}
t6		Z_{out}, $R1_{in}$

2.2.3 Interrupt Cycle

- It is concerned to perform the test for any pending interrupts at the end of every instruction execution and if an interrupt occurs.
- It involves the different micro-operations for various t-states as shown in Table 2.2.4.
- Here you will notice a special register used called as the stack pointer (SP), which always points to the top of the stack.
- This stack is used to store the return address of the interrupted program.

Table 2.2.4 : Microinstructions for the interrupt cycle

	Operation	Microinstructions
t1	SP ← SP – 1	SP_{out}(address), Decrement, Z_{in}
t2	SP → MAR	Z_{out}, MAR_{in}, SP_{in}
t3	PC → MBR	PC_{out}(return address), MBR_{in}, Write
t4	ISR address → PC	ISR address out, PC_{in} (new address), Wait for memory write cycle

- The control signals are to be generated using the control unit.
- The design of this control unit can be done in two ways namely : Hardwired Control Unit and Microprogrammed Control Unit.
- We will see these two methods in the subsequent sections.

2.2.4 Applications of Microprogramming

University Question

Q. What are applications of microprogramming ?

MU - May 14, May 15, 3 Marks

- The applications of microprogramming are as given below :

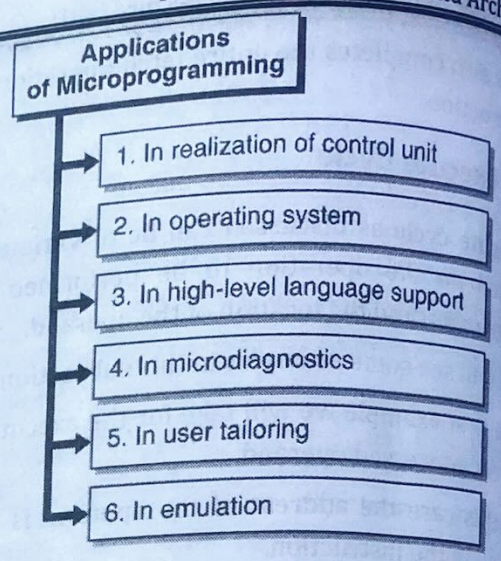

Fig. 2.2.2 : Applications of microprogramming

1. In realization of control unit

- Microprogramming is used widely now f implementing the control unit of computers.

2. In operating system

- Microprograms can be used to implement some the primitives of operating system.
- This simplifies operation system implementatio and also improves the performance of the operatin system.

3. In high-level language support

- In high-level language various sub functions an data types can be implemented using microprogramming.
- This makes compilation into an efficient machine language from possible.

4. In microdiagnostics

- Microprogramming can be used for detection isolation monitoring and repair of system errors.
- This known as microdiagnostics and they significant enhance system maintenance.

5. In user tailoring

- By using RAM for implementing control memory (CM), it is possible to tailor the machine to different applications.

6. In emulation

- Emulation refers to the use of a microprogram on one machine to execute programs originally written for another machine.

- This is used widely as an aid for users in migrating from one computer to another.

2.2.5 Examples of Microprograms

University Question

Q. What is microprogram ? Write microprogram for following operations.

1. ADD R1, M ; Register R1 and Memory location M are added and result store at Register R1.
2. MUL R1, R2 ; Register R1 and Register R2 are multiple and result store at Register R1.

MU - Dec. 18, May 19, 10 Marks

1. Write a microprograms for the instruction :
MOV R_3, R_4

T-state	Operation	Microinstructions
T1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	MBR → IR	MBR_{out}, IR_{in}
T4	R_3 ← R_4	$R_{4\ out}$, $R_{3\ in}$
T5	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T6	SP ← SP – 1	Z_{out}, SP_{in}, MAR_{in}
T7	PC → MDR	PC_{out}, MDR_{in}, WRITE
T8	MDR → [SP]	Wait for mem access
T9	PC ← IS Raddr	PC_{in} IS Raddr out

2. Write a microprogram for the instruction :
ADD R_3, R_4

T-state	Operation	Microinstructions
T1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	MBR → IR	MBR_{out}, IR_{in}
T4	R_3 → x	$R_{3\ out}$, X_{in}, CLRC
T5	R_4 → ALU	$R_{4\ out}$, ADD, Z_{in}
T6	Z → R_3	Z_{out}, $R_{3\ in}$
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T8	SP ← SP – 1	Z_{out}, SP_{in}, MAR_{in}
T9	PC → MDR	PC_{out}, MDR_{in}, WRITE
T10	MDR → [SP]	Wait for mem access
T11	PC ← IS Raddr	PC_{in} IS Raddr out

3. Write a microprogram for the instruction :
MOV R_3 , [R_4] OR LOAD R_3 , [R_4]

T-state	Operation	Microinstructions
T1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	MBR → IR	MBR_{out}, IR_{in}
T4	R_4 → MAR	$R_{4\ out}$, MAR_{in}, READ
T5	Mem → MDR	Wait for mem access
T6	MDR → R_3	MDR_{out} , $R_{3\ in}$
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T8	SP ← SP – 1	Z_{out}, SP_{in}, MAR_{in}
T9	PC → MDR	PC_{out}, MDR_{in}, WRITE
T10	MDR → [SP]	Wait for mem access
T11	PC ← IS R_{addr}	PC_{in} IS Raddr out

4. Write a microprogram for the instruction : ADD R_3, $[R_4]$

T-state	Operation	Microinstructions
T1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	MBR → IR	MBR_{out}, IR_{in}
T4	R_4 → MAR	R_4 out, MAR_{in}, READ, CLRC
T5	Mem → MDR	Wait for men access
T5	MDR → ALU	MD R_{out}, Z_{in} ADD
T6	$R_3 \rightarrow X_1$	R_{3out}, X_{in}
T7	Z → R_3	Z_{out}, $R_{3\ in}$
T8	Check for intr	Assumption enabled intr pending
		CLRX, SETC, SP_{out}, SUB, Z_{in},
T9	SP ← SP – 1	Z_{out}, SP_{in}, MAR_{in}
T10	PC → MDR	PC_{out}, MDR_{in}, WRITE
T11	MDR → [SP]	Wait for mem access
T12	PC ← IS Raddr	PC_{in} IS Raddr out

5. Write a microprogram for the instruction : ADD R_3, $[[R_4]]$

T-state	Operation	Microinstructions
T1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	MBR → IR	MBR_{out}, IR_{in}
T4	mem → MDR	Wait for mem access
T5	MDR → ALU	MDR_{out}, Z_{in}, ADD
T6	Z → R_3	Z_{out}, $R_{3\ in}$
T7	Check for intr	Assumption enabled intr pending
		CLRX, SETC, SP_{out}, SUB,
T8	SP ← SP – 1	Z_{out}, SP_{in}, MAR_{in}
T9	PC → MDR	PC_{out}, MDR_{in}, WRITE
T10	MDR → [SP]	Wait for mem access
T11	PC ← IS Raddr	PC_{in} IS Raddr out

6. Write a microprogram for the instruction : ADD R_3, 45H

T-state	Operation	Microinstructions
T1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	MBR → IR	MBR_{out}, IR_{in}
T4	R_3 → X	R_{out}, X_{in}, CLRC
T5	IRdata → ALU	IR_{out}, ADD, Z_{in}
T6	Z → R_3	Z_{out}, $R_{3\ in}$
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T8	SP ← SP – 1	Z_{out}, SP_{in}, MAR_{in}
T9	PC → MDR	PC_{out}, MDR_{in}, WRITE
T10	MDR → [SP]	Wait for mem access
T11	PC ← IS Raddr	PC_{in} IS Raddr out

7. Write a microprogram for the instruction : ADD R_3, [45H]

T-state	Operation	Microinstructions
T1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	MBR → IR	MBR_{out}, IR_{in}
T4	IR addr → MAR	IR_{out}, MAR_{in}, D, CLRC
T5	R_3 → X	$R_{3\ out}$, X_{in}

T-state	Operation	Microinstructions
T5	mem → MDR	Wait for mem access
T6	MDR → ALU $[R_3 + 45]$	MDR_{out} Z_{in}, ADD
T7	$Z \rightarrow R_3$	Z_{out}, $R_{3\,in}$
T8	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T9	$SP \leftarrow SP - 1$	Z_{out}, SP_{in}, MAR_{in}
T10	$PC \rightarrow MDR$	PC_{out}, MDR in, WRITE
T11	$MDR \rightarrow [SP]$	Wait for mem access
T12	$PC \leftarrow$ IS Raddr	PC_{in} IS Raddr out

8. Write a microprogram for the instruction ADDX, [Y]

T-state	Operation	Microinstruction
T1	$PC \rightarrow MAR$	PC_{out}, MAR_{in}, READ, CLRT, STC, ADD, Z
T2	$mem \rightarrow MDR$ $PC \leftarrow PC + 1$	Wait for mem access Z_{out}, PC_{in}
T3	$MDR \rightarrow IR$	MDR_{out}, IR_{in}
T4	$Y \rightarrow MAR$	y_{out}, MAR_{in}, READ, CLRC
T5	$mem \rightarrow MDR$ $X \rightarrow Temp$	Wait for mem access X_{out}, T_{in}
T6	$MDR \rightarrow ALU$	MDR_{out}, Z_{in}, ADD
T7	$Z \rightarrow X$	Z_{out}, X_{in}
T8	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T9	$SP \leftarrow Sp - 1$	Z_{out}, SP_{in}, MAR_{in}
T10	$PC \rightarrow MDR$	PC_{out}, MDR_{in}, WRITE
T11	$MDR \rightarrow [SP]$	Wait for mem access
T12	$PC \leftarrow$ IS Raddr	PC_{in} IS Raddr out

9. Write a microprogram for the instruction : ADD X, [[400]]

T-state	Symbolic operations	Microinstruction
T1	$PC \rightarrow MAR$	PC_{out}, MAR_{in}, READ, CLRT, SETC, ADD, Z
T2	$mem \rightarrow MDR$ $PC \leftarrow PC + 1$	Wait for mem access Z_{out}, PC_{in}
T3	$MDR \rightarrow IR$	MDR_{out}, IR_{in}
T4	$IRaddr \rightarrow MAR$	IR_{out}, MAR_{in}, READ, CLRC
T5	$mem \rightarrow MDR$ $X \rightarrow Temp$	Wait for mem access X_{out}, T_{in}
T6	$MDR \rightarrow MAR$	MDR_{out}, MAR_n, READ
T7	$mem \rightarrow MDR$	Wait for mem access
T8	$MDR \rightarrow ALU$	MDR_{out}, ADD, Z_{in}
T9	$Z \rightarrow X$	Z_{out}, X_{in}
T8	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T9	$SP \leftarrow SP - 1$	Z_{out}, SP_{in}, MAR_{in}
T10	$PC \rightarrow MDR$	PC_{out}, MDR_{in}, WRITE
T11	$MDR \rightarrow [SP]$	Wait for mem access
T12	$PC \leftarrow$ IS Raddr	PC_{in} IS Raddr out

- Write a microprogram for the instruction : MUL R1, R2

T-state	Operation	Microinstructions
T1	$PC \rightarrow MAR$	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T2	$M \rightarrow MBR$ $PC \leftarrow PC + 1$	Z_{out}, PC_{in}, Wait for memory fetch cycle
T3	$MBR \rightarrow IR$	MBR_{out}, IR_{in}
T4	$R_3 \rightarrow x$	$R_{3\,out}$, X_{in}, CLRC
T5	$R_4 \rightarrow ALU$	$R_{4\,out}$, MUL, Z_{in}
T6	$Z \rightarrow R_3$	Z_{out}, $R_{3\,in}$

T-state	Operation	Microinstructions
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP_{out}, SUB, Z_{in},
T8	SP ← SP – 1	Z_{out}, SP_{in}, MAR_{in}
T9	PC → MDR	PC_{out}, MDR_{in}, WRITE
T10	MDR → [SP]	Wait for mem access
T11	PC ← IS Raddr	PC_{in} IS Raddr out

2.3 Control Unit : Hardwired Control Unit Design Methods

University Question

Q. Describe hardwire control unit and specify its advantages.
MU - May 14, Dec. 15, Dec. 16, May 17, 10 Marks

Q. Explain with diagram functioning of Hardwired Control unit.
MU - May 15, 8 Marks

Q. Explain different technique for design of control unit of computer.
MU - Dec. 18, 10 Marks

- The hardwired control unit is viewed as a sequential an combinational logic circuit.
- It is used to generate a set of fixed sequences of control signals. It is implemented using any of a variety of "standard" digital logic circuits.
- The major advantages of hardwired control units are higher speed of operation and smaller space required for implementation on silicon wafer i.e. the IC (Integrated Circuit), since the components required are lesser.
- The only disadvantage is that modifications to the design are slightly difficult.
- The use of hardwired control unit is majorly found in the RISC designs.
- There are different methods to implement hardwired control unit :
  1. State table method.
  2. Delay-Element method.
  3. Sequence counter method.
  4. PLA method.

2.3.1 State Table Method

- In this method state transition for each instructi is made and hence a state table is obtained.
- This state table is then combine to form instruction set state table, where all the instructio (OPCODE) are considered as inputs and accordi to this the next state is being determined.
- Each state with a set of microinstructions to issued to various components of the processor well as external control signals.
- This state table is then implemented using flip-flo and combinational circuit to generate differe control signals.
- An example state table implementation is shown Fig. 2.3.1.

	Inputs			
State	I	I_2	.....	I_m
S_1	$S_{1,1}, O_{1,2}$	$S_{1,2}, O_{1,2}$	.....	$S_{1,m}, O_{1,m}$
S_2	$S_{2,1}, O_{2,1}$	$S_{2,2}, O_{2,2}$	.....	$S_{2,m}, O_{2,m}$
: :			.....	
S_n	$S_{n,1}, O_{n,1}$	$S_{n,2}, O_{n,2}$	.....	$S_{n,m}, O_{n,m}$

Fig. 2.3.1(a) : Mealy

	Inputs				
State	I	I_2	.....	I_m	Outputs
S_1	$S_{1,1}$	$S_{1,2}$	.....	$S_{1,m}$	O_1
S_2	$S_{2,1}$	$S_{2,2}$	.....	$S_{2,m}$	O_2
: :			.....		
S_n	$S_{n,1}$	$S_{n,2}$	.....	$S_{n,m}$	O_n

(b) Moore Type

Fig. 2.3.1 : State tables for a finite-state machine

2.3.2 Delay Element Method

- This method is implemented using delay elements i.e. D-flipflops.
- A flipflop is made to give output logic '1' after the specific event or in a t-state in sequence and the outputs of these flipflops are used to generate control signals or the micro-instructions i.e. two

operations that require a delay of 1 t-state between them are separated by a D flipflop between them.

- Fig. 2.3.2 shows this implementation.

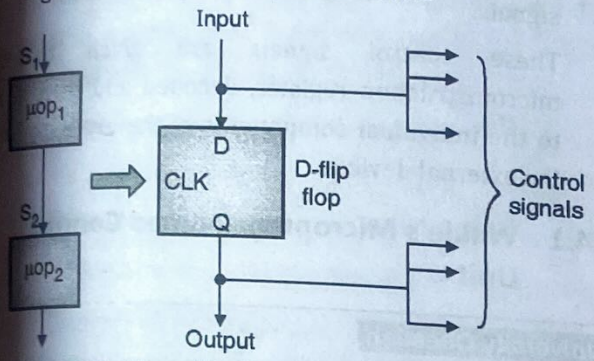

Fig. 2.3.2 : Use of D flip flop as a delay element between two sets of control signals

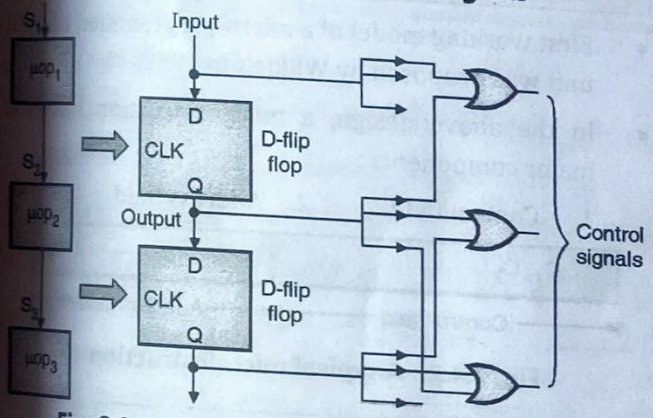

Fig. 2.3.3 : Use of OR gate in delay element method of hardwired control unit

- The signals that activate the same control signal are ORed together i.e. if a signal has to be activated from the outputs of multiple flipflops then an OR gate is used as shown in Fig. 2.3.3.
- In case if a decision is to be made then it is implemented using a If-Then-Else circuit i.e. two AND gates coupled to a OR gate. This is shown in Fig. 2.3.4.

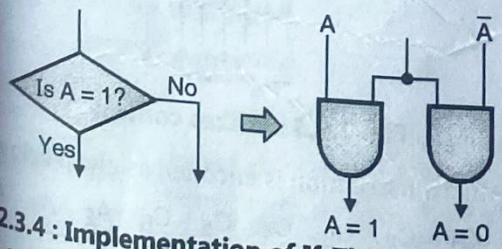

Fig. 2.3.4 : Implementation of If-Then-Else in delay element method of hardwired control unit

2.3.3 Sequence Counter Method

- In this method, multiple clock signals are derived from the master clock using a standard counter-decoder approach as shown in the Fig. 2.3.5.
- These signals are applied to the combinational portion of the circuit.
- As shown in Fig. 2.3.5, the counter keeps on incrementing and generating different counts.
- The counts are decoded using a decoder and the decoder outputs are given to various components as control signals in the CPU.

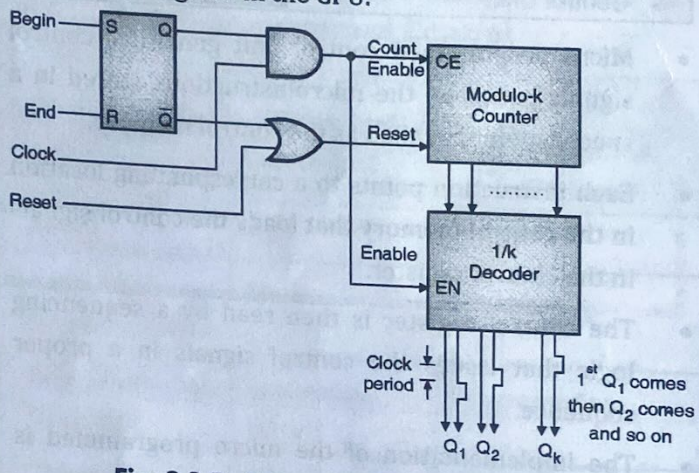

Fig. 2.3.5 : Sequential counter method of hardwired control unit implementation

2.3.4 PLA Method

- In this method a PLA (Programmable Logic Array) is used to generate the control signals. PLA is an array of AND gates at input and the OR gates at output.
- The inputs are to be given to the AND gates, which can be connected to the specific OR gates as required.
- The OR gates outputs are the outputs of the overall PLA and are used as control signals in the system i.e. the inputs to the AND array is from various control signals generated and the output of the OR array is given as control signals to various components of the processor as well as the external control signals required.
- Fig. 2.3.6 shows the implementation of the PLA method of implementation of control unit.

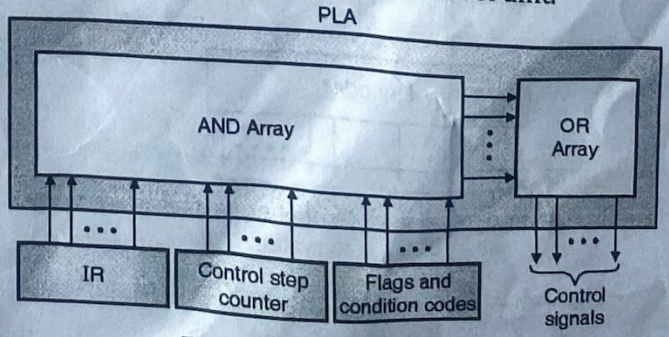

Fig. 2.3.6 : PLA technique

2.4 Control Unit : Soft Wired (Micro programmed) Control Unit Design Methods

University Question

Q. Explain with diagram functioning of Microprogrammed Control Unit. MU - May 14, 8 Marks

- Micro programmed control unit generates control signals based on the microinstructions stored in a special memory called as the control memory.
- Each instruction points to a corresponding location in the control memory that loads the control signals in the control register.
- The control register is then read by a sequencing logic that issues the control signals in a proper sequence.
- The implementation of the micro programmed is shown in the Fig. 2.4.1.
- The Instruction Register (IR), Status flag and condition codes are read by the sequencer that generates the address of the control memory location for the corresponding instruction in the IR.

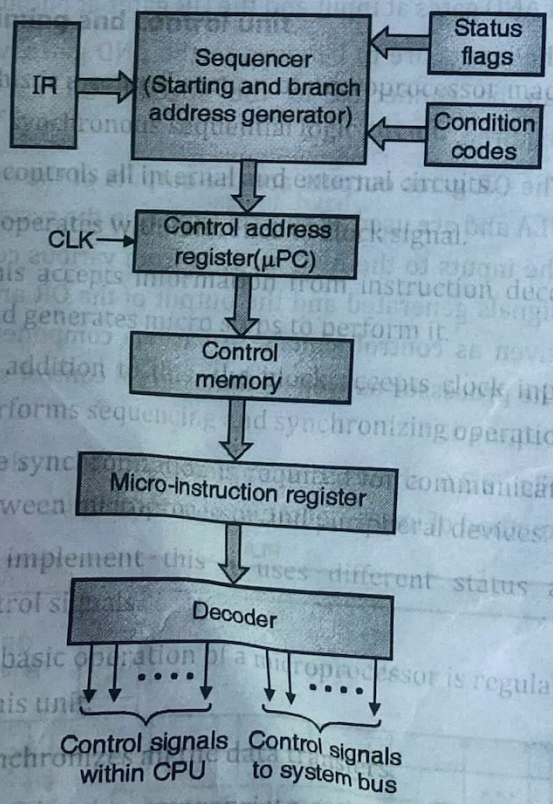

Fig. 2.4.1 : Micro programmed control unit

- This address is stored in the Control address register that selects one of the locations in the control memory having the corresponding control signals.
- These control signals are given to the microinstruction register, decoded and then given to the individual components of the processor and the external devices.

2.4.1 Wilkie's Microprogrammed Control Unit

University Question

Q. Explain Wilkie's Engine (Hardwired Control Unit) in detail. MU - Dec. 14, 10 Marks

- First working model of a micro-programmed control unit was proposed by Wilkie's in 1952.
- In the above design, a microinstruction has two major components :
  1. Control field
  2. Address field

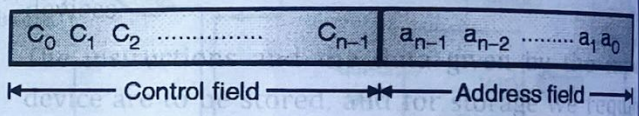

Fig. 2.4.2 : A typical microinstruction

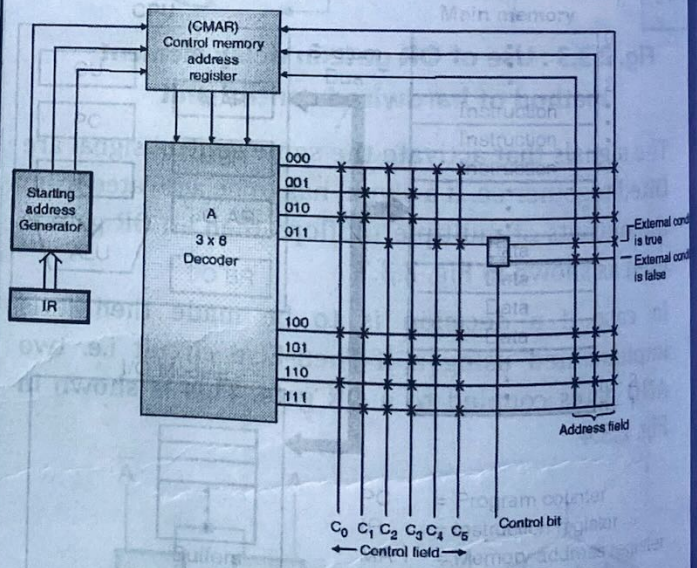

Fig. 2.4.3 : Wilkes control

- If a microinstruction is encoded as given below :

C_0	C_1	C_2	C_3	C_4	C_5	C_6	A_2	A_1	A_0
0	1	0	0	1	1	0	0	1	0

- Then the control information 0100110 indicates that on execution of above microinstruction, control signals C_1, C_4 and C_5 will be activated.

- Address field contains the address of the next microinstruction.
- Thus, after execution of the above instruction, the next instruction to be executed is one which is at the address 010.
- The control field tells the control signals which are to be activated and the address field provide the address of the next microinstruction to be executed.
- In Wilkie's control, control memory is organized as a program logic array.
- The Control Memory Access Register (CMAR) can be loaded from an external source (instruction register) as well as from the address field of a microinstruction.
- A machine instruction typically provides the starting address of a micro-program in control memory.
- On the basis of starting address from instruction register, decoder activates one of the eight output lines.
- This activated line, in turn, generates control signals and the address of the next microinstruction to be executed.
- This address is once again fed to the CMAR resulting in activation of another control line and address field.
- This cycle is repeated till the execution of the instruction is achieved.
- For example, as shown below, if the machine instruction under execution causes the decoder to have an entry address for a machine instruction in control memory at line 000.
- The decoder activates the lines in the sequence given as follows :

Line activated	Control signal generated	Address of next microinstruction
000	C_0, C_2, C_4, C_5	001
001	C_1, C_3	010
010	C_0, C_1, C_3	011
011	C_2, C_4, C_5	2

- On execution of microinstruction at address 011, address of the next microinstruction depends on the external condition.
- If the condition is true then the address 101 will be selected else the address 110 will be selected.

2.4.2 Comparison between Hardwired and Micro-programmed Control

Attribute	Hardwired Control	Micro-programmed Control
Speed	Fast	Slow
Cost of implementation	More	Cheaper
Implementation approach	Sequential circuit	Programming
Flexibility	Not flexible, difficult to modify for new instruction.	Flexible, new machine instructions can easily be added.
Ability to handle Complex instructions	Difficult	Easier
Design process	Complicated	Systematic
Decoding and sequencing logic	Complex	Easy
Applications	RISC µp	CISC µp
Instruction set size	Small	Large
Control memory	Absent	Present
Chip area required	Less	More

2.5 Concepts of Nano Programming :

University Question

Q. Explain concepts of nanoprogramming.

MU : May 14, Dec. 14, May 15, Dec. 16, 6 Marks

- The following block diagram of Fig. 2.5.1 explains the flow in a nanoprogram concept.
- The microprogram memory (control ROM) is much narrower because it contains pointers to the actual microinstructions.
- The microinstruction register contains a short pointer that points to the nanoinstruction memory.

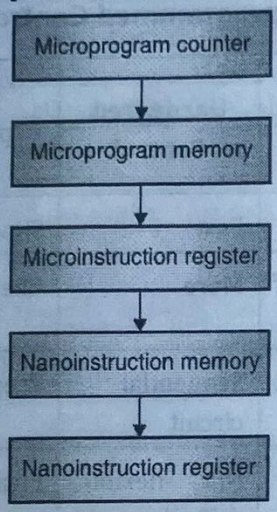

Fig. 2.5.1 : Nano programming

- The nanoinstruction memory contains the actual micro-instructions which is very wide.

2.6 Introduction to RISC and CISC Architecture

2.6.1 RISC versus CISC

Sr. No.	Properties	RISC	CISC
1.	**Number of Instructions**	Less	More
2.	**Addressing Modes**	Less	More
3.	**Instruction Formats**	Less	More
4.	**Instruction Size**	Fixed	Variable
5.	**Control Unit**	Hardwired	Micro-programmed
6.	**Number of Bus Cycles to execute an instruction**	Single CPU cycle (for 80% Instructions)	Multiple CPU cycles
7.	**Control Logic And Decoding Subsystem**	Simple	Complex
8.	**Pipelining**	Huge no. of stages of Pipelining	Difficulty in efficient implementation
9.	**Design time and Probability of Design Errors**	Smaller time and less probable	Long time and Significant probability
10.	**Complexity of Compiler**	Simpler	More complex and the results of "optimization" may not be most efficient and the fastest machine language code
11.	**HLL instructions**	Supported	Not Supported

2.6.2 RISC Properties

A RISC system must satisfy the following properties :

1. Single-cycle execution of all (or at least 80 percent) instructions.
2. Single-word standard length of all instructions.
3. Small number of instructions (<= 128).
4. Small number of instruction formats (<= 4).
5. Small number of addressing modes (<= 4).
6. Memory access possible by load and store instructions only.
7. All operations, except load and store, are register to register i.e. within the CPU.
8. It must have a hardwired control unit.
9. It must also have a relatively large (at least 32) general-purpose CPU register file.

2.6.3 Register Window

1. Since there are a huge number of registers in a RISC processor, it can be useful in saving the latency period during procedure call. Parameter passing in is possible by the **register window**. This policy also allows reasonable HLL support in RISC designs.
2. The register file is subdivided into groups of registers, called register **windows**.
3. A certain group of 'i' registers, suppose R_0 to $R_{(i-1)}$, are designated as **global registers**. The global registers are accessible to all procedures running on the system at all the times.
4. On the other hand, each procedure is assigned a separate window within the register file, which is not accessible to other procedures.
5. The window base (first register within the window) is pointed to by a field called **current window pointer (CWP)** located in the CPU's **status register (SR)**.

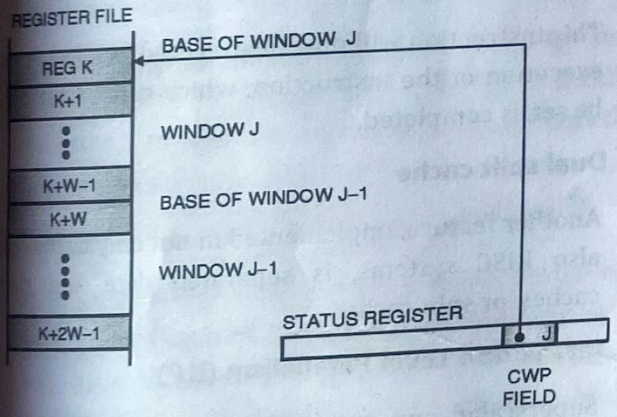

Fig. 2.6.1 : Simple Non-overlapping Register Window

6. If the currently running procedure is assigned the register window J, hence taking up registers K, K+1, . . ., K+W-1 (where W is the number of registers per window), the CWP contains the value J, hence pointing to the base of window J. If the next procedure to execute takes up window J+1, the value in the CWP field will be incremented accordingly to point to J+1.
7. Register windowing can work more efficiently for parameter passing between calling and called procedures by partial overlapping of the windows. The last N registers of window J will be the first N registers of window J+1.
8. If the procedure taking up window J calls a procedure, which will be assigned the next window J+1, it can pass N parameters to the called procedure by placing their values into registers (K+W-N) to (K+W-1). The same registers will be automatically be available to the called procedure without any further movement of data.

2.6.4 Miscellaneous Features or Advantages of RISC Systems

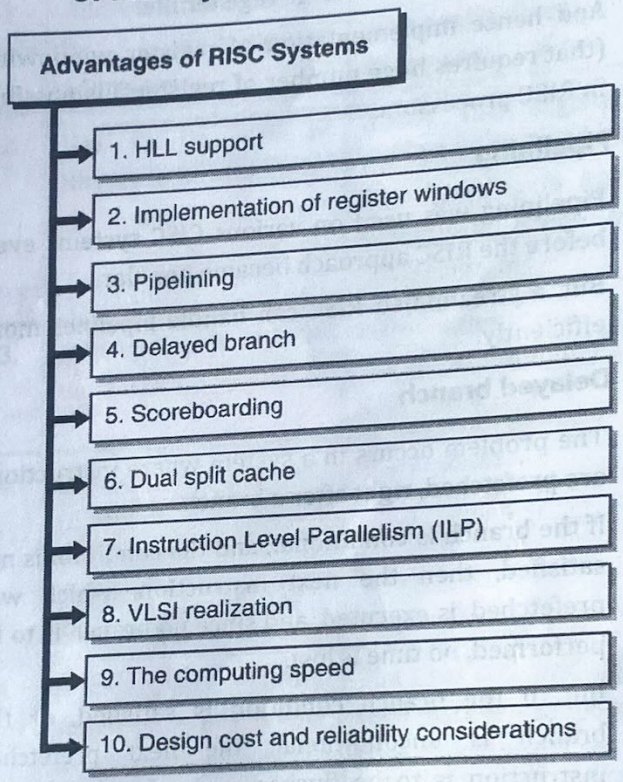

Fig. 2.6.2 : Advantages of RISC Systems

1. HLL support

(a) The support for High Level Language (HLL) features is mandatory in the design of any computing system.

(b) The procedure call-return and parameters passing is the most time-consuming operation in typical HLL programs.

(c) HLL support is provided in RISC machines by supporting efficiently the handling of local variables, constants, and procedure calls, while leaving less frequent HLL operations to instructions sequences and subroutines.

(d) One of the mechanisms supporting the handling of procedures, and their parameter passing in particular, is the feature of the **register window**.

2. Implementation of register windows

(a) A CISC processor's control unit takes up a large percentage of the chip area, leaving very little space for other subsystems and basically not permitting a large register file, needed for an efficient implementation of windowing.

(b) A RISC processor's control unit makes up a must smaller percentage of the chip area, yielding the necessary space for a large register file.

(c) And hence implementation of register windowing (that requires huge number of registers) is possible in RISC processors.

3. Pipelining

(a) Pipelining was used on various CISC systems even before the RISC approach became popular.

(b) But, a streamlined RISC can handle pipelines more efficiently.

4. Delayed branch

(a) The problem occurs in a system where instructions are prefetched, right after a branch.

(b) If the branch is conditional, and the condition is not satisfied, then the next instruction, which was prefetched, is executed, and since no branch is to be performed, no time is lost.

(c) But, if the branch condition is satisfied, or the branch is unconditional, the next prefetched instruction is to be flushed and other instruction pointed to by the branch address is to be fetched in its place. The time required to prefetch the flushed instruction is wasted.

(d) Such waste of time is solved by using the **delayed branch approach.**

(e) In this approach, the instructions are reshuffled such that the operation does not change the result.

(f) A successful branch is assumed and the execution of the branch is delayed until the already prefetched instructions are executed. Hence, no time is lost and there is no change in the intended program operation.

(g) But the compiler has to take care that the instructions followed by the branch instructions are to be executed irrespective to the branch to be taken or not.

5. Scoreboarding

(a) Another problem in instruction pipelines is that data dependency.

(b) The data in some register put by instruction 1 may be required by instruction 2; and before the value the register is available, the instruction 2 may ready for execution yielding a possibly incorrect result.

(c) A method used to solve this problem is called **scoreboarding**.

(d) A special CPU control register i.e. the **scoreboard register,** is required for this purpose.

(e) If there are 32 registers, a scoreboard register 32 bit long will be required; each of its bit represents one of the 32 CPU registers.

(f) If register 'i' is involved as a destination in the execution of instruction 1, bit 'i' in the scoreboard register is set; and as long as bit 'i' is set, any subsequent instruction in the pipeline will be prevented from using Ri in any way until bit 'i' is cleared.

(g) This instruction will now be executed as soon as the execution of the instruction, which caused bit 'i' to be set, is completed.

6. Dual split cache

Another feature, implemented in not only a CISC but also RISC systems, is separated data and code caches, or split cache.

7. Instruction Level Parallelism (ILP)

Superscalar and superpipelined designs are also mostly implemented in a RISC design.

8. VLSI realization

(a) The chip area, dedicated to the realization of the control unit, is considerably less. Therefore, on a RISC VLSI chip, there is more area available for other features (cache, FPU, part of the main memory, memory management unit, I/O ports, etc).

(b) As a result of the considerable reduction of the control area, a large number of CPU registers can fit on-chip.

(c) By reducing the control area on the VLSI chip and filling the area by numerous identical registers, the **regularization factor** (which is defined as, ratio of

chip area utilized by other features to the chip space required by control unit) of the chip increases. The higher the regularization factor, the lower is the VLSI design cost.

9. The computing speed

(a) A simpler and smaller control unit in RISC requires fewer gates. This results in shorter propagation paths for control unit signals, decreasing the delay time for control signals and hence yielding a faster operation.

(b) A significantly reduced number of instructions, formats, and addressing modes results in a simpler and smaller decoding system, resulting in faster decoding operation.

(c) A hardwire-controlled system can hence be implemented, with a reduced control unit that will in general be faster than a microprogramed control unit.

(d) A relatively large CPU register file also reduces CPU-memory traffic to fetch and store data operands.

(e) A large register set can also be used to store parameters to be passed from a calling to a called procedure, to store the information of a process that was interrupted by another.

10. Design cost and reliability considerations

(a) It takes a shorter time to complete the design of a RISC control unit, because of the smaller instruction set, fixed instruction length and less instruction formats. Thus contributing to the reduction in the overall design cost.

(b) A simpler and smaller control unit will obviously have a reduced number of design errors and, therefore, a higher reliability.

2.6.5 RISC Shortcomings

1. Since a RISC has a small number of instructions, a number of functions, performed on CISC's by a single instruction, will need more instructions on RISC. Hence, RISC's code will be longer.
2. More memory will have to be allocated for RISC programs, and the instruction accesses between the memory and the CPU will be increased.

2.6.6 ON-Chip Register File versus Cache Evaluation

Modern CISC have on-chip cache to compensate the registers of RISC processors. But, let us evaluate the advantages of register file over on-chip cache.

Sr. No.	CACHE	CPU Register file
1.	Addressed as locations in memory-long addresses.	Separate register addressing-short addresses.
2.	Has to be tens of Kbytes to be effective.	About 128 registers (of 4-bytes each, i.e. 512 bytes) will have significant effect on performance.
3.	Information loaded in units of lines (blocks).	Information can be loaded individually to each register.
4.	Slower access (Effective address calculation, virtual to physical address translation).	Faster access.
5.	Information loaded based on prefetch and replacement policies.	The user can load any information at any time.
6.	Inaccessible by the user.	Fully accessible by the user.
7.	Possibility of a miss.	No miss is possible.

Review Questions

Q. 1 Explain fetch cycle with the corresponding micro-program.

Q. 2 Describe the register organization within the CPU.

Q. 3 Explain microinstruction sequencing and execution.

Q. 4 Write short note on microinstructions to execute an instruction MOV [R1], R2.

Q. 5 Explain microinstruction sequencing and execution.

Q. 6 What are applications of microprogramming ?

Q. 7 Write short note on state table method of control unit design.

Q. 8 Explain the sequence counter method of hardwired control unit.

Q. 9 Explain with block diagram the micro-programmed control unit.

Q. 10 Describe hardwired control unit and specify its advantages.

Q. 11 Explain with diagram functioning of hardwired Control unit.

Q. 12 Describe hardwire control unit and specify its advantages.

Q. 13 Explain with diagram functioning of Microprogrammed Control Unit.

Q. 14 Explain Wilkie's microprogrammed control unit.

Q. 15 Explain Wilkie's Engine (Hardwired Control Unit) detail.

Q. 16 Explain concepts of nanoprogramming.

Q. 17 Write short note on nano-programming.

Q. 18 Write a short note on instruction formats.

Q. 19 Explain basic instruction cycle.

Q. 20 Explain the instruction cycle with interrupts.

□□