Control Unit

Unit: 3

Computer Organization & Architecture

B Tech 3rd Sem

Khushboo
ECE
Department

Content

Control Unit:

- Instruction types, formats
- Basic Computer Instructions
- Control Unit – Basic control unit
- Hardwire and micro programmed control unit
- Concept of Horizontal and Vertical microprogramming
- Instruction cycles and sub cycles (fetch and execute etc)
- Micro operations
- Execution of a complete instruction.
- Program Control
- Parallel Processing – Flynn's Classification
- Pipelining
- RISC & CISC architecture

Course Objective

- Implementation of control unit techniques and the concept of Pipelining.
- Study of Instruction, types of instruction, format of instuction,Cycle and sub cycle, micro operation.

Course Outcome

- Implementation of control unit techniques and the concept of Pipelining.

Instruction Types

Most computer instructions can be classified into three categories:

1. Data transfer instructions

2. Data manipulation instructions

3. Program control instructions

•Data transfer instructions cause transfer of data from one location to another without changing the binary information content.

•Data manipulation instructions are those that perform arithmetic, logic, and shift operations.

•Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.

Instruction Types

1.Data Transfer Instructions -

Data transfer instructions move data from one place in the computer to another without changing the data content.

TABLE 8-5 Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Instruction Types

- **Load** instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- **Store** instruction designates a transfer from a processor register into memory.
- **Move** instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another.
- **Exchange** instruction swaps information between two registers or a register and a memory word.
- **Input and output** instructions transfer data among processor registers and input or output terminals.
- **Push and Pop** instructions transfer data between processor registers and a memory stack.

Instruction Types

TABLE 8-6 Eight Addressing Modes for the Load Instruction

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1) +	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

ADR stands for an address, *NBR* is a number or operand, *X* is an index register, *R*1 is a processor register, and *AC* is the accumulator register

Instruction Types

2. Data Manipulation Instructions -

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types:

A.Arithmetic instructions

B. Logical and bit manipulation instructions

C.Shift instructions

Instruction Types

TABLE 8-7 Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Instruction Types

TABLE 8-8 Typical Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Instruction Types

TABLE 8-9 Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Instruction Types

3. Program control instructions –

•Program control instructions specify conditions for altering the content of the program counter, while data transfer and manipulation instructions specify conditions for data-processing operations.

•The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution.

Instruction Types

TABLE 8-10 Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Instruction Types

Status Bit Conditions

- It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions can be stored for further analysis.
- **Status bits are also called *condition-code* bits or *flag* bits.**

Instruction Types

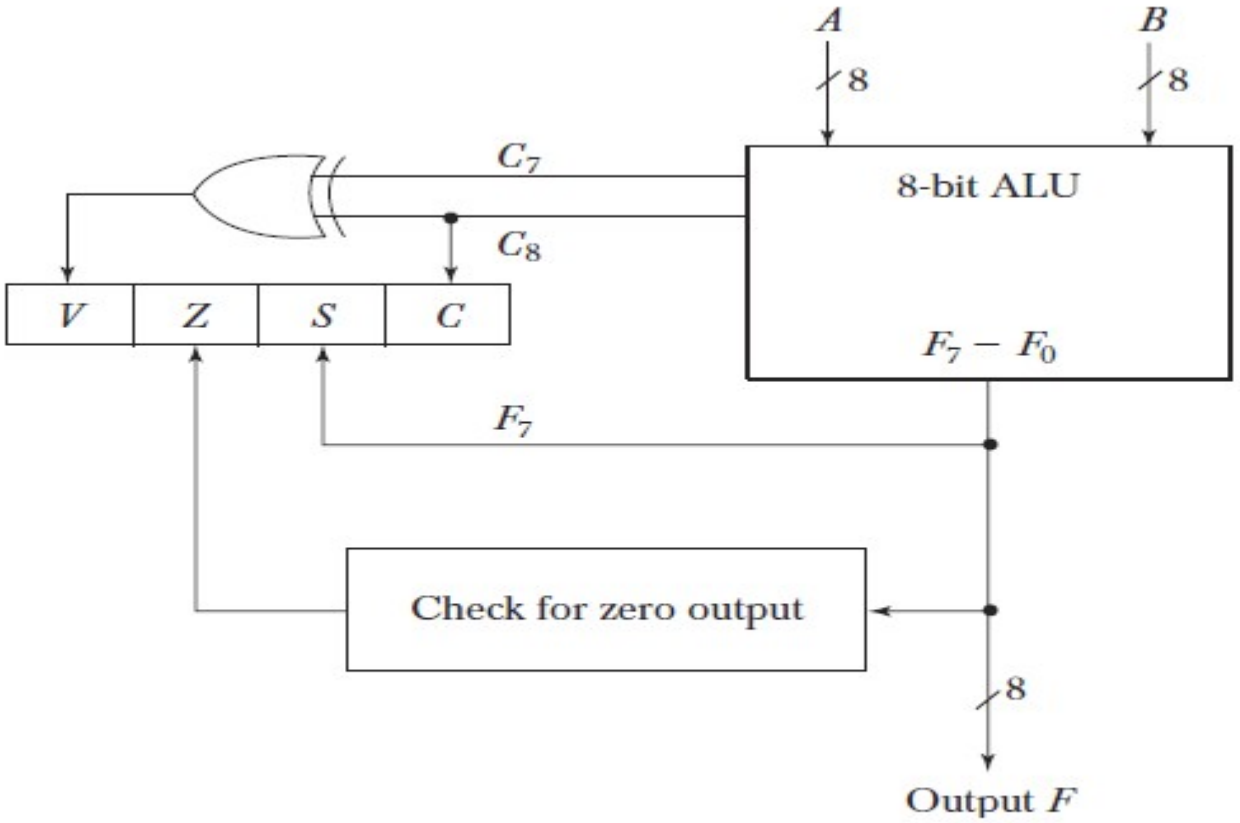


Figure 8-8 Status register bits.

Instruction Types

Status Bit Conditions

1.Bit *C* (carry) is set to 1 if the end carry *C*8 is 1. It is cleared to 0 if the carry is 0.

2.Bit *S* (sign) is set to 1 if the highest-order bit *F7* is 1. It is set to 0 if the bit is 0.

3.Bit *Z* (zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise. In other words, *Z* 1 if the output is zero and *Z* 0 if the output is not zero.

4.Bit *V* (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement

Instruction Types

Conditional Branch Instructions

TABLE 8-11 Conditional Branch Instructions

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$

Instruction

- **Program**
  - A sequence of (machine) instructions
- **(Machine) Instruction**
  - A group of bits that tell the computer to *perform a specific operation* (a sequence of micro-operation)
- The instructions of a program, along with any needed data are stored in memory
- The CPU reads the next instruction from memory
- It is placed in an *Instruction Register* (IR)
- Control circuitry in control unit then translates the instruction into the sequence of micro operations necessary to implement it

Instruction Format

- The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.
- Most computers fall into one of three types of CPU organizations:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

Instruction Format

- An instruction is of various length depending on the number of addresses it contains.
- On the basis of number of addresses, instructions can be classified –

1. Three – address instruction
2. Two – address instruction
3. One – address instruction
4. Zero – address instruction
5. RISC instructions

Instruction Format

Three-Address Instructions

- This format of instruction use three addresses to specify either a processor register or a memory operand.
- The program in assembly language that evaluates

X = (A + B) * (C + D) is

ADD R1, A, B	**R1 ← M[A]+M[B]**
ADD R2, C, D	**R2 ← M[C]+M[D]**
MUL X, R1, R2	**M[X] ← R1* R2**

- It is assumed that the computer has two processor registers, *R*1 and *R*2. The symbol *M*[*A*] denotes the operand at memory address symbolized by *A*.

Instruction Format

Two-Address Instructions

- This format of instruction uses only two addresses to specify a processor register or a memory location.
- The program in assembly language that evaluates

X = (A + B) * (C + D) is

MOV R1, A	**R1 ← M[A]**
ADD R1, B	**R1 ← R1+M[B]**
MOV R2, C	**R2 ← M[C]**
ADD R2, D	**R2 ← R2+M[D]**
MUL R1, R2	**R1 ← R1*R2**
MOV X, R1	**M[X] ← R1**

Instruction Format

One-Address Instructions

- This format of instruction uses only one address to specify a register or a memory location.

- **It uses an implied accumulator (AC) register for all data manipulation**.

- **For multiplication and division there is a need for a second register**. However, **here we will neglect the second register and assume that the *AC* contains the result of all operations.**

Instruction Format

- The program in assembly language that evaluates

X = (A + B) * (C + D) is

LOAD	**A**	**AC ← M[A]**
ADD	**B**	**AC ← AC+M[B]**
STORE	**T**	**M[T] ← AC**
LOAD	**C**	**AC ← M[C]**
ADD	**D**	**AC ← AC+M[D]**
MUL	**T**	**AC ← AC * M[T]**
STORE	**X**	**M[X] ← AC**

- All operations are done between the *AC* register and a memory operand. ***T* is the address of a temporary memory location required for storing the intermediate result.**

Instruction Format

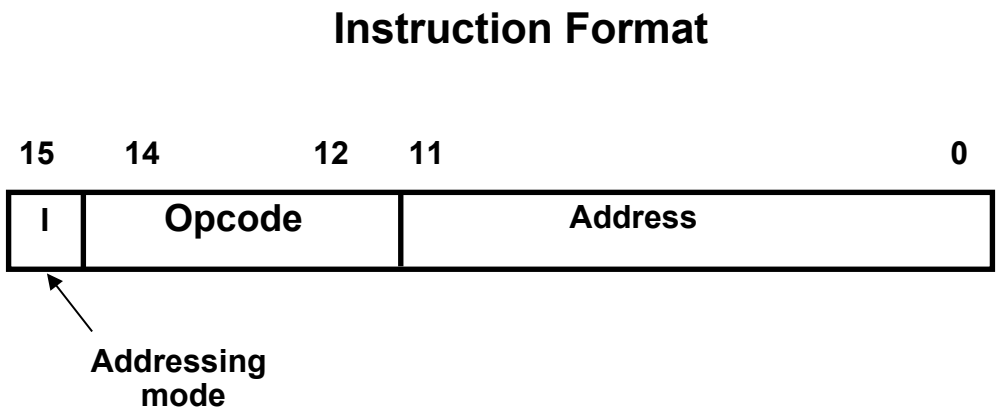
Zero-Address Instructions

- A stack-organized computer does not use an address field for the instructions ADD and MUL.
- The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
- Program shows how **X = (A + B) * (C + D)** will be written for a stack-organized computer. (*TOS* stands for top of stack).

```
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 ← (C+D)*(A+B)
POP    X      M[X] ← TOS
```

Basic Computer Instruction

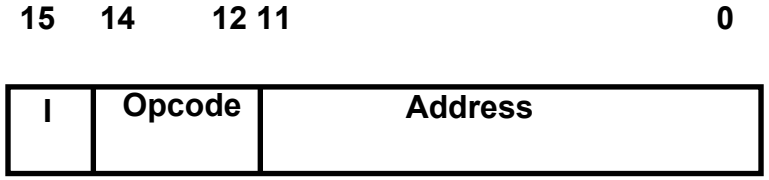
- In the Basic Computer, bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing)
- Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction's opcode.



Basic Computer Instruction

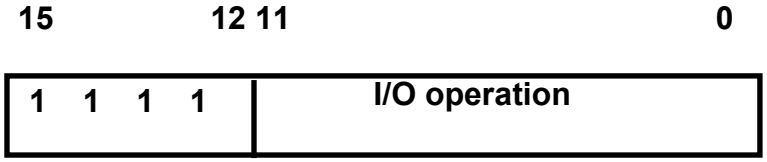
- **Basic Computer Instruction Format**

1. Memory-Reference Instructions (OP-code = 000 ~ 110)



2. Register-Reference Instructions (OP-code = 111, I = 0)

3. Input-Output Instructions (OP-code =111, I = 1)



Control Unit (CU)

- **Control Unit** is the part of the computer's central processing unit (CPU), which directs the operation of the processor.
- It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor.
- A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor.

Functions of the Control Unit

- It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
- It interprets instructions.
- It controls data flow inside the processor.
- It receives external instructions or commands to which it converts to sequence of control signals.
- It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
- It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

Types of the Control Unit

There are two types of control units:

1. Hardwired control unit and
2. Micro programmable control unit.

Types of the Control Unit

1. **Hardwired control unit**

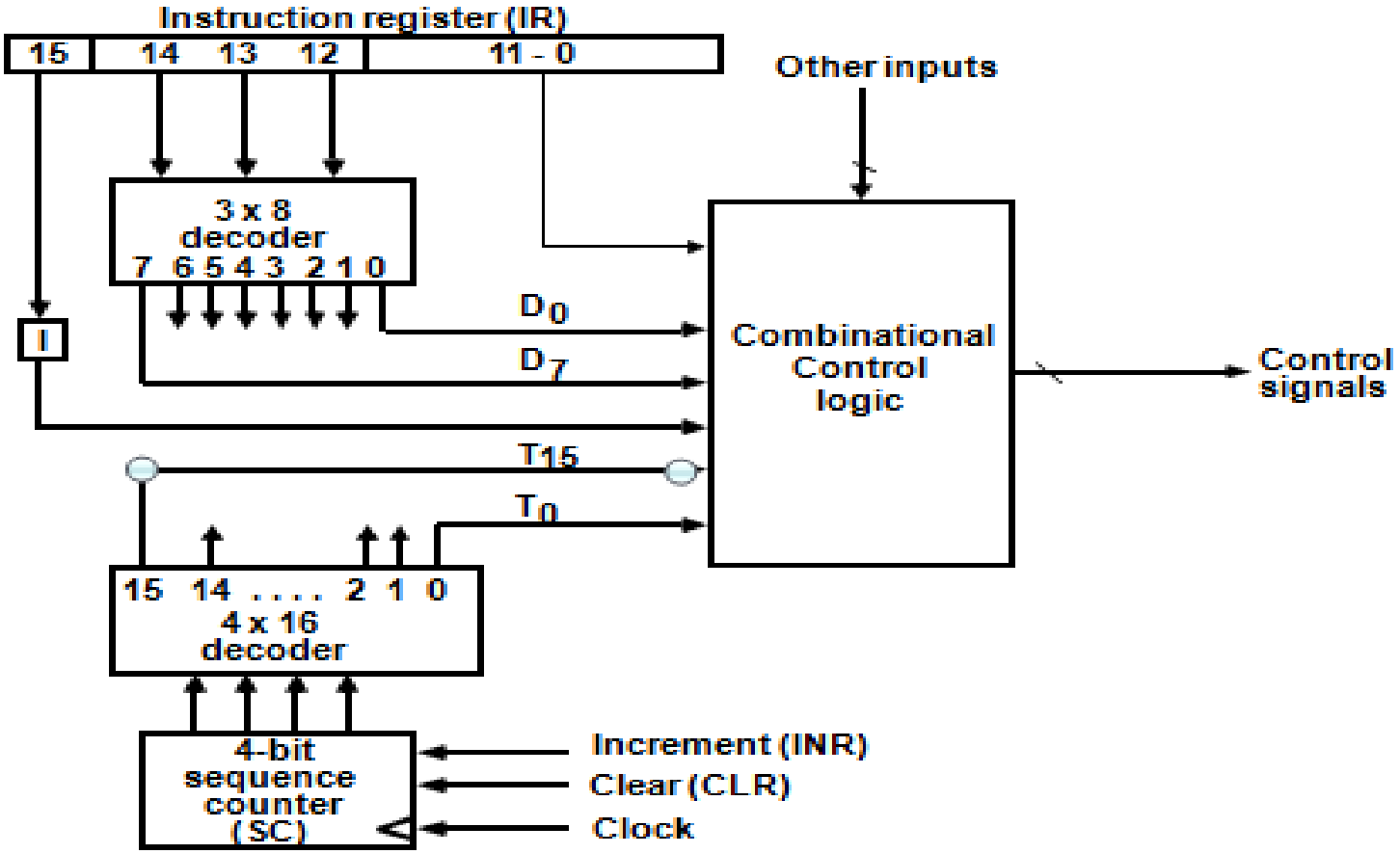
- The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- It has the advantage that it can be optimized to produce a fast mode of operation.
- A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed.

2. **Micro programmable control unit**

- In the microprogrammed organization, the control information is stored in a control memory.
- The control memory is programmed to initiate the required sequence of microoperations. In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

Control unit of Basic Computer

- **Control unit of Basic Computer**



Control Timing Signal

As an example, consider the case where *SC* is incremented to provide timing signals *T*0, *T*1, *T*2, *T*3, and *T*4 in sequence. **At time *T*4, *SC* is cleared to 0 if decoder output *D*3 is active**. This is expressed symbolically by the statement

***D*3 *T*4: *SC* ← 0**

Control Timing Signal

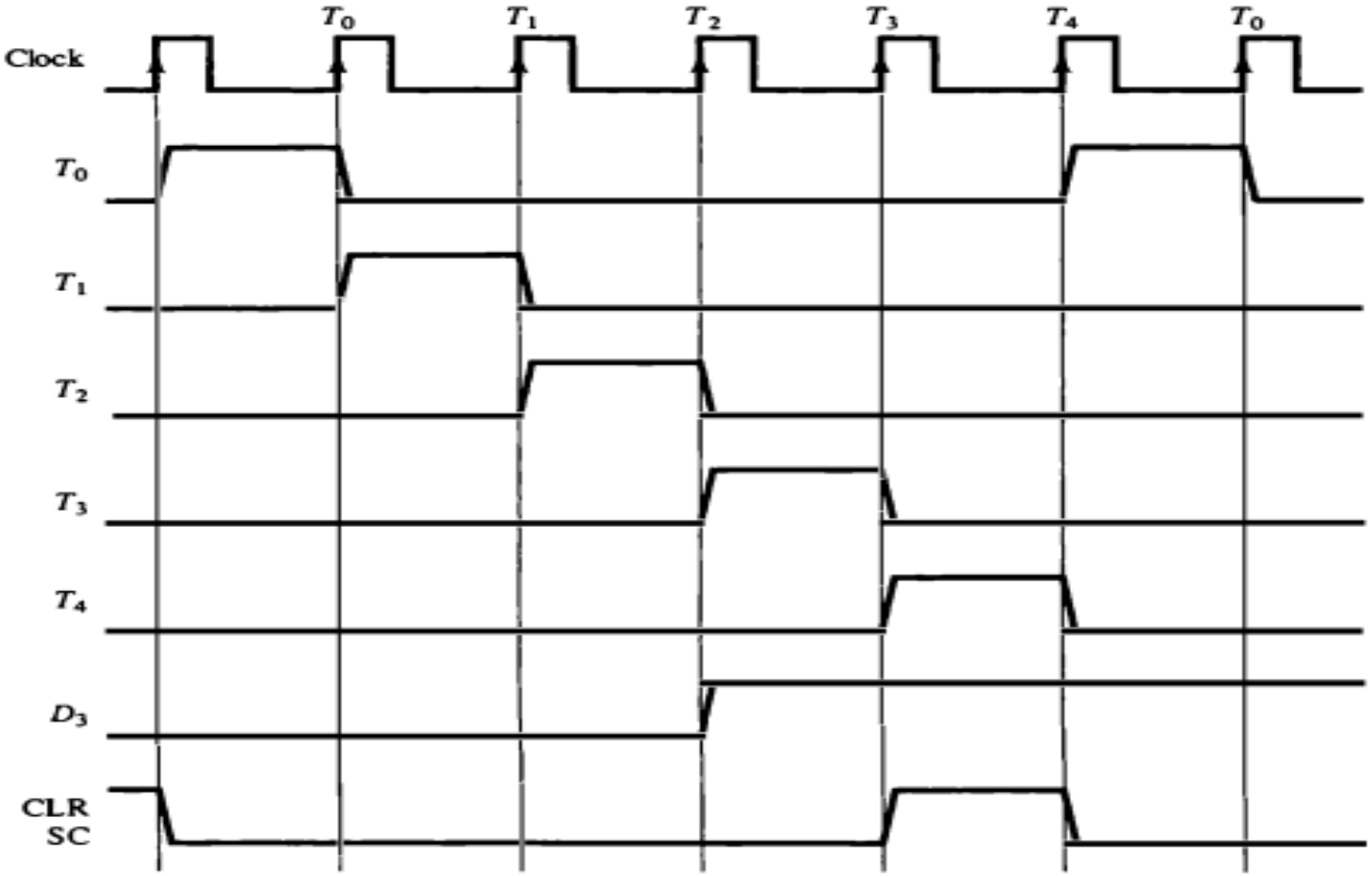


Figure 5-7 Example of control timing signals.

Micro programmable control unit

Micro programmable control unit

- In the microprogrammed organization, the control information is stored in a control memory.
- The control memory is programmed to initiate the required sequence of microoperations. In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

Micro programmable control unit

- The general configuration of a microprogrammed control unit is demonstrated in the block diagram of Fig. 7-1.
- The control memory is assumed to be a ROM, within which all control information is permanently stored.
- 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.
- For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.
- The next address may also be a function of external input conditions. 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.

Micro programmable control unit

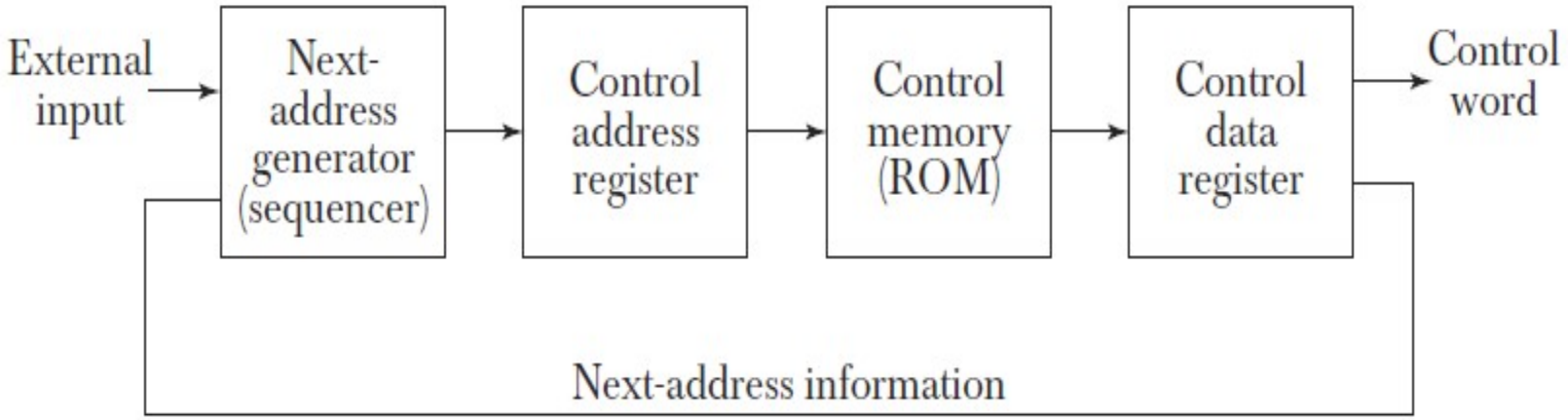


Figure 7-1 Microprogrammed control organization.

Micro programmable control unit

- The **next address generator is sometimes called a microprogram *sequencer,*** as it determines the address sequence that is read from control memory.

- 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.

Micro programmable control unit

- Control address register (CAR)
  - Specifies address of next microinstruction
- Next address generator (microprogram sequencer)
  - Determines address sequence for control memory
- Microprogram sequencer functions
  - Increment CAR by one
  - Transfer external address into CAR
  - Load initial address into CAR to start control operations
- Control data register (CDR)- or pipeline register
  - Holds microinstruction read from control memory
  - Allows execution of microoperations specified by control word simultaneously with generation of next microinstruction

Address Sequencing

Address sequencing/ next address generating capabilities required in a control unit are-

- Incrementing CAR
- Unconditional or conditional branch, depending on status bit conditions
- Mapping from bits of instruction to address for control memory
- Facility for subroutine call and return

Address Sequencing

- Figure 7-2 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.
- 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 microprogram wishes to return from the subroutine.

Address Sequencing

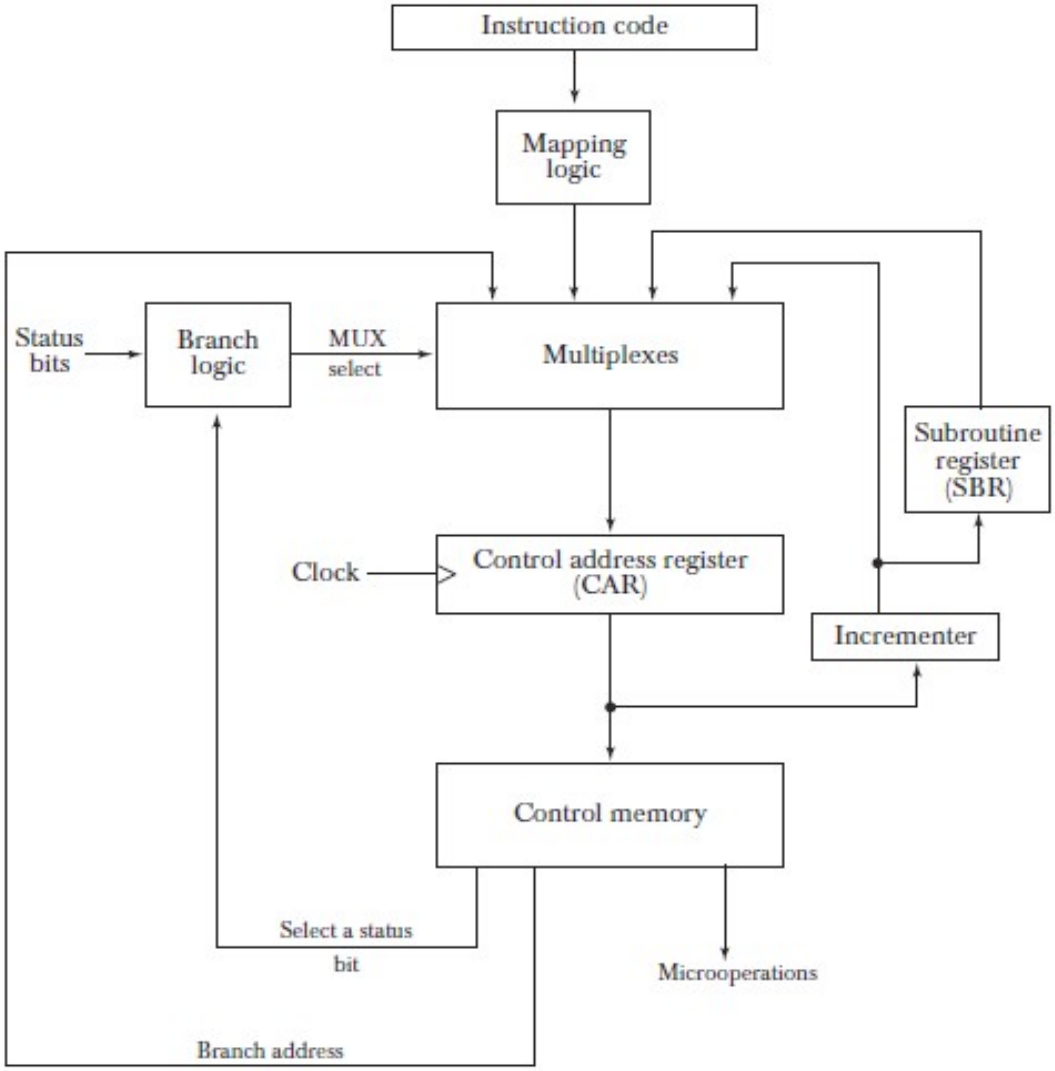


Figure 7-2 Selection of address for control memory.

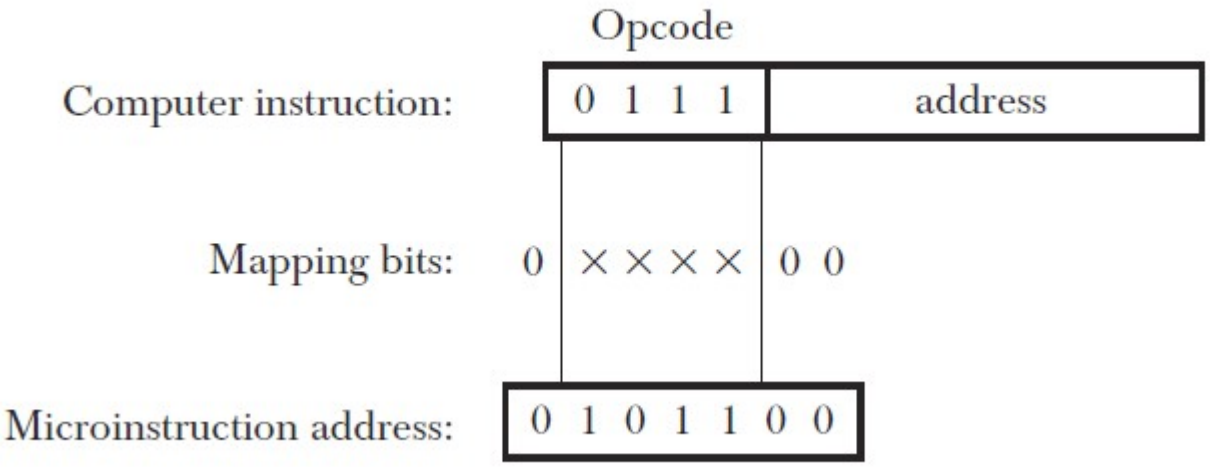
Address Sequencing

- Each computer instruction has its own microprogram routine stored in a given location of the control memory.

- **Mapping of instruction**

Transformation from instruction code bits to address in control memory where routine is located

Figure 7-3 Mapping from instruction code to microinstruction address.



Address Sequencing

- For example, a computer with a simple instruction format as shown in Fig. 7-3 has an operation code of four bits which can specify up to 16 distinct instructions.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in Fig. 7-3.
- 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.

Microprogram Example

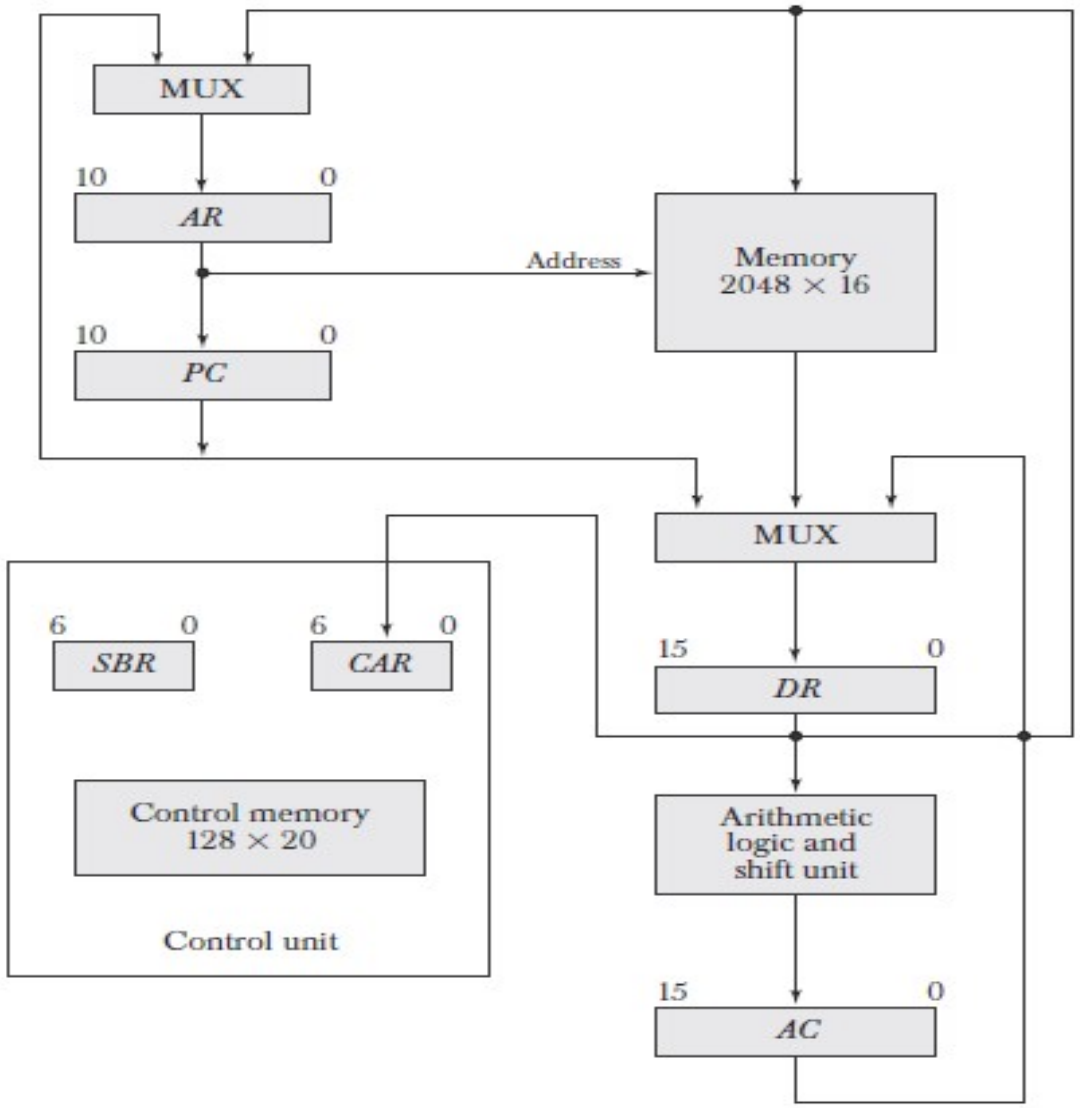


Figure 7-4 Computer hardware fonfiguration.

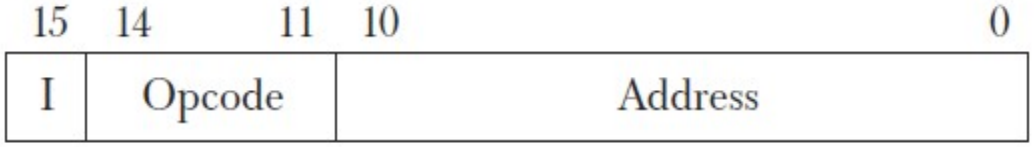
Microprogram Example

Computer Configuration

- The block diagram of the computer is shown in Fig.
- It consists of two memory units: a main memory for storing instructions and data, and a control memory for storing the microprogram.
- Four registers are associated with the processor unit and two with the control unit. The processor registers are program counter *PC*, address register *AR*, data register *DR*, and accumulator register *AC.*
- The control unit has a control address register *CAR* and a subroutine register *SBR.*

Microprogram Example

Figure 7-5 Computer instructions.



(a) Instruction format

Symbol	Opcode	Description
ADD	0000	$AC \rightarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

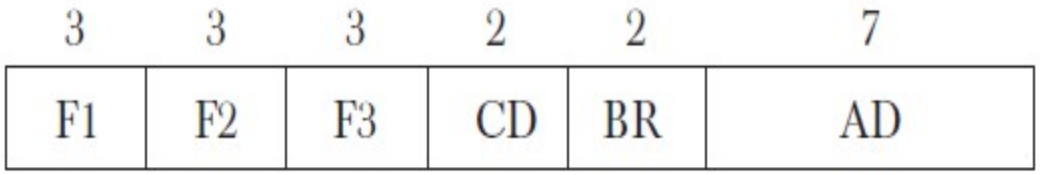
EA is the effective address

(b) Four computer instructions

Microprogram Example

Microinstruction Format

The microinstruction format for the control memory is shown in Fig. The 20 bits of the microinstruction are divided into four functional parts.



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

Microprogram Example

TABLE 7-1 Symbols and Binary Code for Microinstruction Fields

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

Microprogram Example

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow AC$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Microprogram Example

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

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2-5) \leftarrow DR(11-14)$, $CAR(0,1,6) \leftarrow 0$

Hardwired/Microprogrammed

Sr. No	Attribute	Hardwired	Microprogrammed
1	Speed	Fast	Slow
2	Cost of implementation	More	Cheaper
3	Implementation approach	Sequential circuit	Programming
4	Flexibility	Not flexible	Flexible
5	Ability to handle complex instruction	Difficult	Easier
6	Design process	Complicated	Systematic
7	Decoding and sequencing logic	Complex	Easy
8	Application	RISC µp	CISC µp
9	Control memory	Absent	Present
10	Chip area	Less	more

Concept of horizontal and vertical microprogramming.

Micro-programmed control unit

Micro-programmed control unit can be classified into two types based on the type of Control Word stored in the Control Memory.

1. Horizontal micro-programmed control unit
2. Vertical micro-programmed control unit.

Concept of horizontal and vertical microprogramming.

Horizontal micro-programmed control unit

•The **control signals are represented in the decoded** binary format, i.e., 1 bit/CS.

Here 'n' control signals require n bit encoding.

•In horizontal organization, as mentioned above, you can assume that every bit in the control word corresponds to a control signal.

•Horizontal organization has more control over the potential parallelism of operations in the data path; however, it uses up lots of control store.

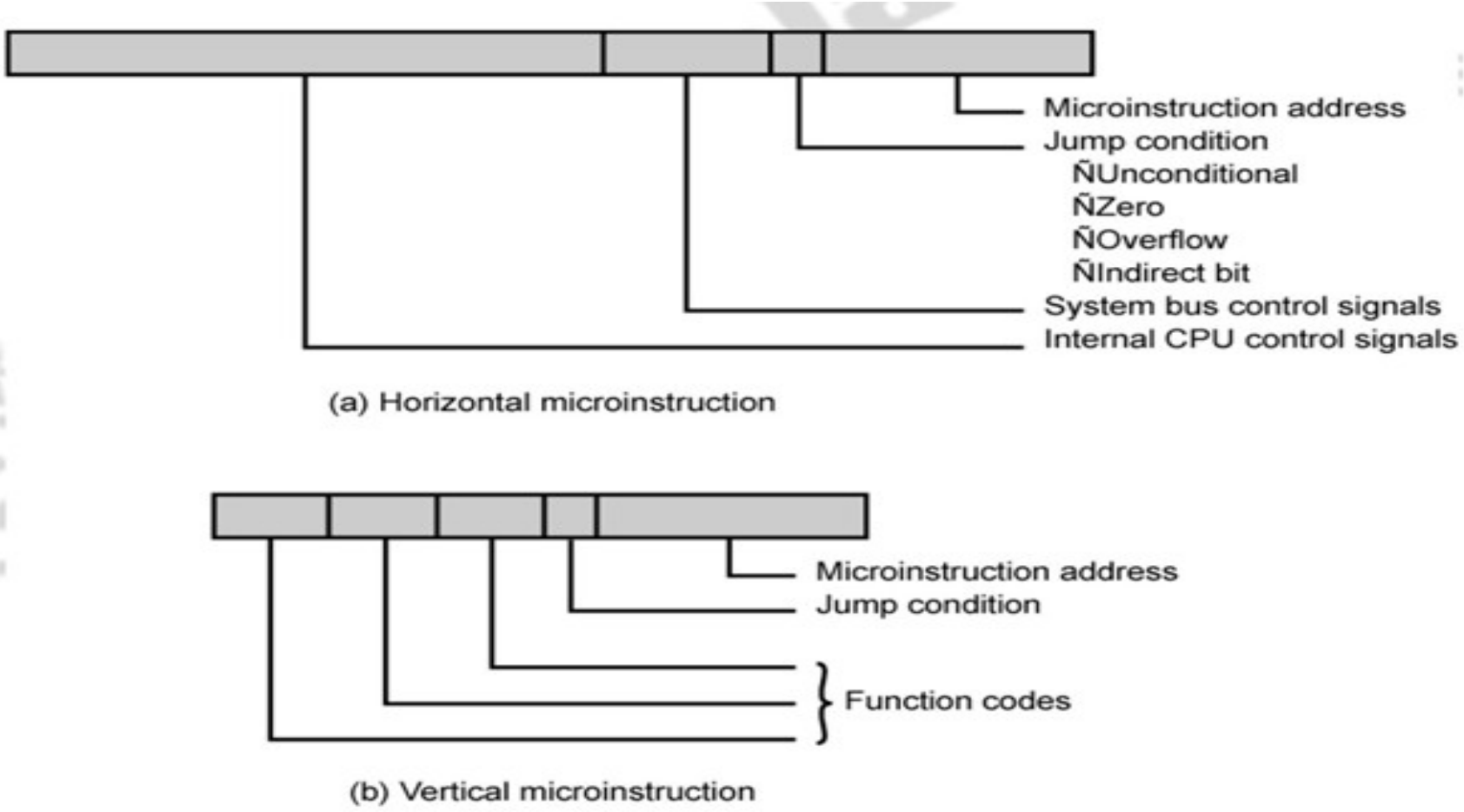
Concept of horizontal and vertical microprogramming.

Vertical micro-programmed control unit

- **The control signals are represented in the encoded** binary format. **Here 'n' control signals require $\log_2 n$ bit encoding.**

- In the case of a vertical organization, the signals are grouped and encoded in order to reduce the size of the control word.

Concept of horizontal and vertical microprogramming.



Concept of horizontal and vertical microprogramming.

Horizontal µ-programmed CU	Vertical µ-programmed CU
It supports longer control word.	It supports shorter control word.
It allows higher degree of parallelism. If degree is n, then n Control Signals are enabled at a time.	It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
No additional hardware is required.	Additional hardware in the form of decoders are required to generate control signals.
It is faster than Vertical micro-programmed control unit.	it is slower than Horizontal micro-programmed control unit.
It is less flexible than Vertical micro-programmed control unit.	It is more flexible than Horizontal micro-programmed control unit.

Concept of horizontal and vertical microprogramming.

Horizontal μ-programmed CU	Vertical μ-programmed CU
Horizontal micro-programmed control unit uses horizontal microinstruction, where every bit in the control field attaches to a control line.	Vertical micro-programmed control unit uses vertical microinstruction, where a code is used for each action to be performed and the decoder translates this code into individual control signals
Horizontal micro-programmed control unit makes less use of ROM encoding than vertical micro-programmed control unit.	Vertical micro-programmed control unit makes more use of ROM encoding to reduce the length of the control word.

Instruction Cycle

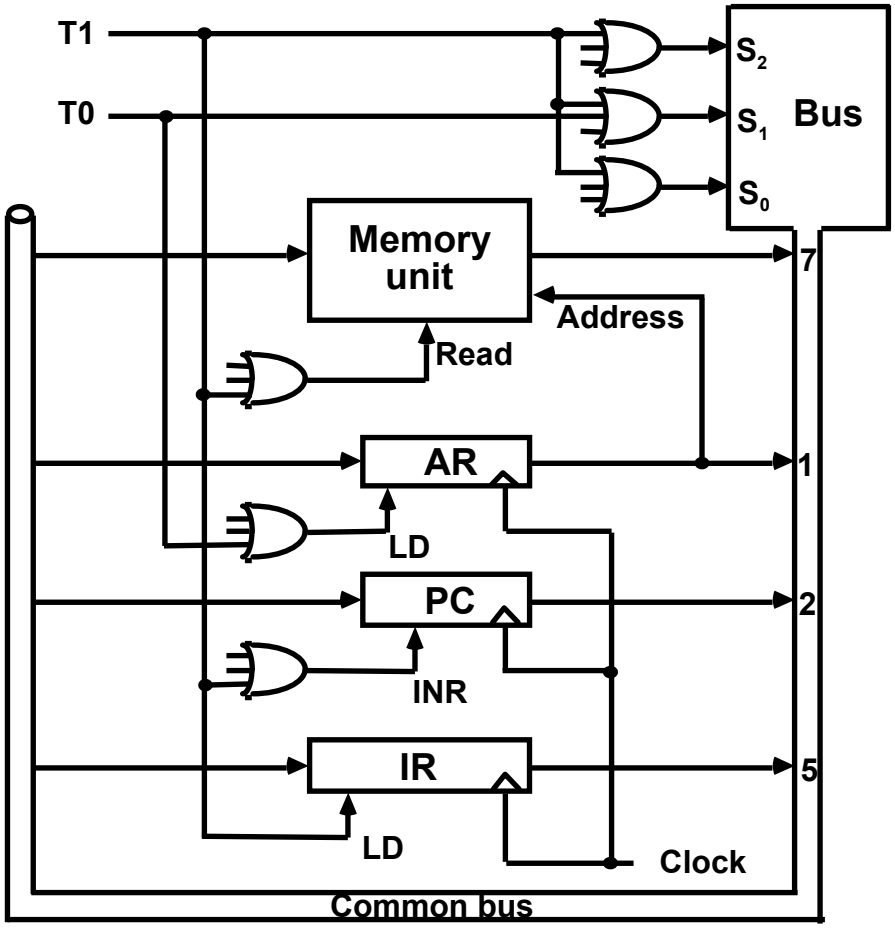
- A program residing in the memory unit of the computer consists of a sequence of instructions.
- The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases.
- In the basic computer each instruction cycle consists of the following phases:

1. **Fetch an instruction from memory.**
2. **Decode the instruction.**
3. **Read the effective address from memory if the instruction has an indirect address.**
4. **Execute the instruction.**

Instruction Cycle

- After an instruction is executed, the cycle starts again at step 1, for the next instruction
- *Note*: Every different processor has its own (different) instruction cycle.

Instruction Cycle-Fetch and Decode



T0: AR ← PC
($S_0S_1S_2$=010, T0=1)

T1: IR ← M [AR],
PC ← PC + 1
(S0S1S2=111, T1=1)

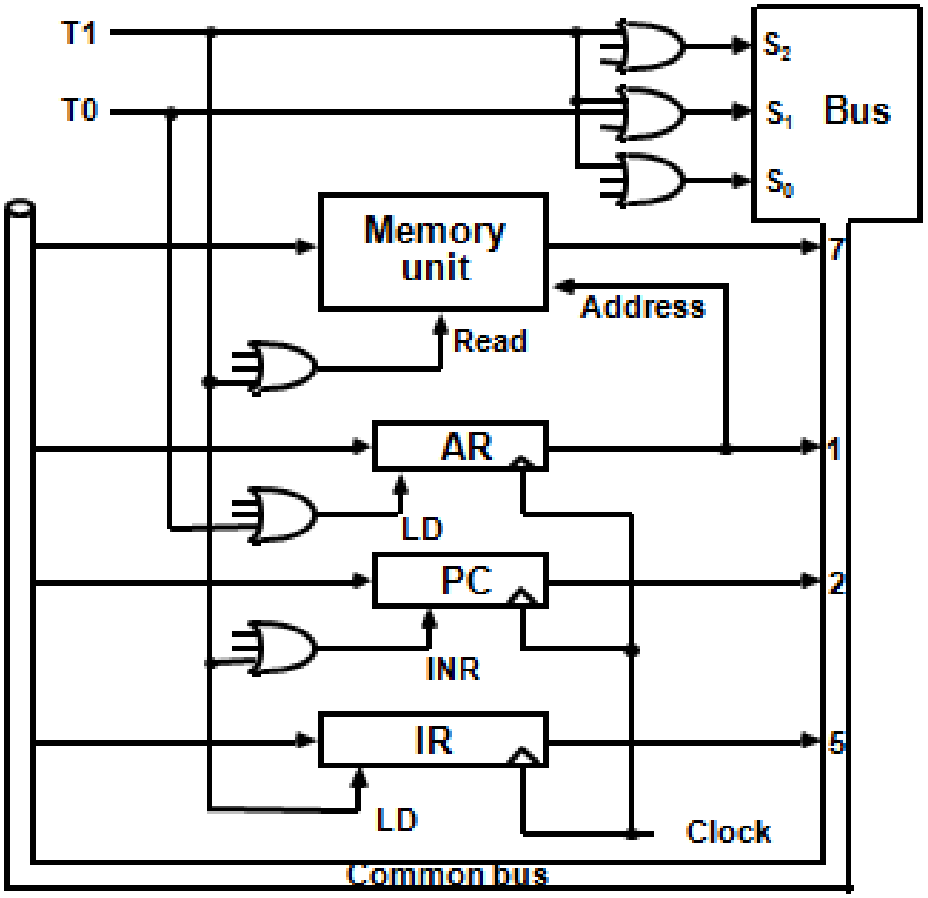
T2: D0, . . . , D7
Decode IR(12-14),
AR ← IR(0-11),
I ← IR(15)

Fig: Register Transfer for the fetch phase

Instruction Cycle -Fetch and Decode

- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)



Instruction Cycle-Fetch and Decode

T0 Cycle:

1.Place the content of PC onto the bus by making the bus selection inputs S2S1S0 equal to 010.

2.Transfer the content of bus to AR by enabling the LD input of AR.

T1 Cycle:

1.Enable the read input of memory.

2.Place the content of memory onto the bus by making S2S1S0= 111.

3.Transfer the content of bus to IR by enabling the LD input of IR.

4.Increment PC by enabling the INR input of PC.

Instruction Cycle- Fetch and Decode

Determine the Type of Instruction

- The timing signal that is active after the decoding is *T*3.
- During time *T*3, the control unit determines the type of instruction that was just read from memory.
- The flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.

Instruction Cycle & Sub Cycle

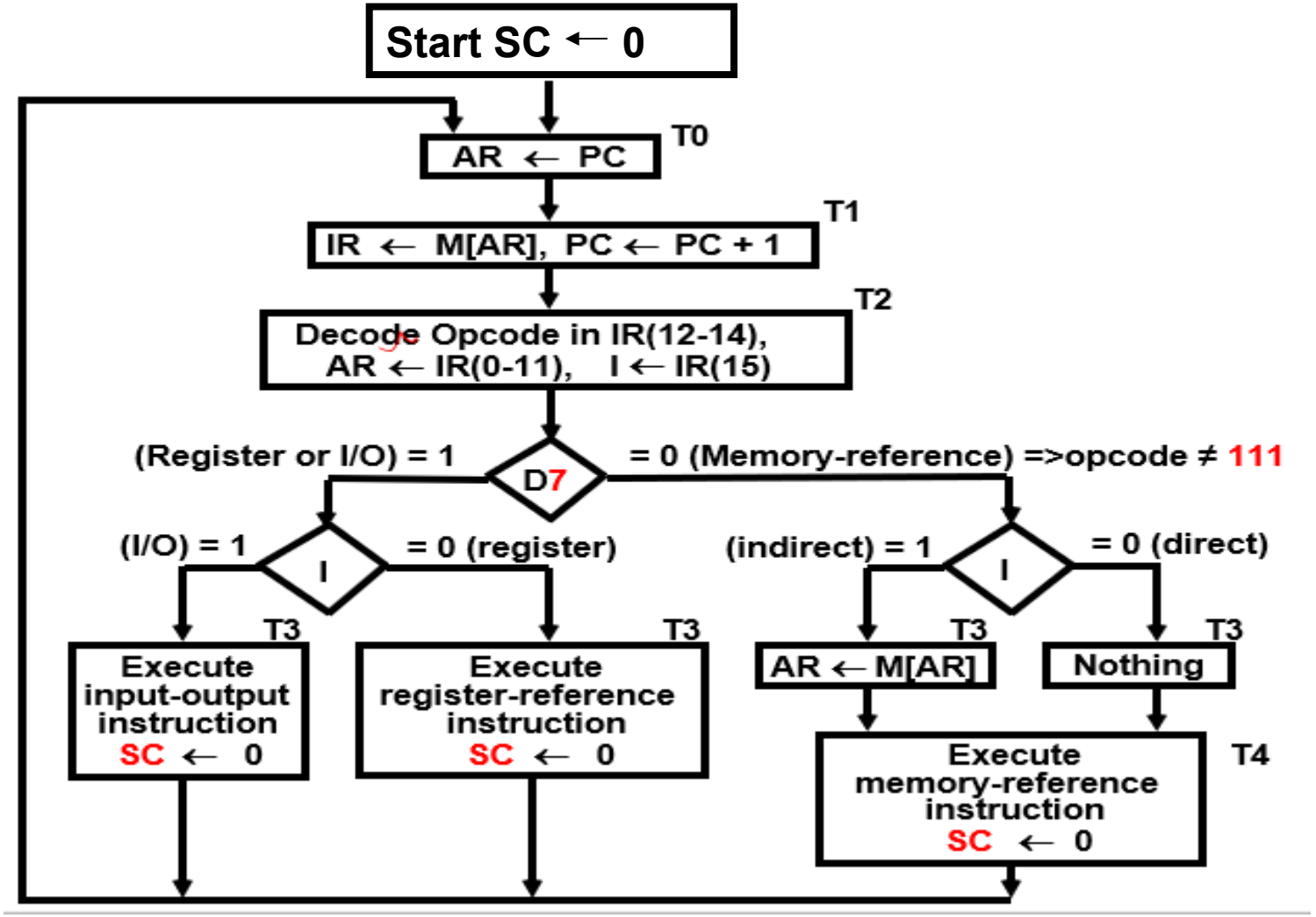


Fig: Flowchart for Instruction Cycle

Instruction Cycle- Fetch and Decode

- The timing signal that is active after the decoding is *T*3. During time *T*3, the control unit determines the type of instruction that was just read from memory.
- The flowchart of Fig. 5-9 presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.
- Decoder output *D*7 is equal to 1 if the operation code is equal to binary 111.
- If *D7* = 1, the instruction must be a register-reference or input–output type.
- If *D*7 = 0, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.
- Control then inspects the value of the first bit of the instruction, which is now available in flip-flop *I*. If *D*7 = 0 and *I* 1, we have a memory reference instruction with an indirect address. It is then necessary to read the effective address from memory.

Execution of Complete Instruction

As for example, consider the instruction : "Add contents of memory location NUM to the contents of register R1 and store the result in register R1." For simplicity, assume that the address NUM is given explicitly in the address field of the instruction .That is, in this instruction, direct addressing mode is used.

Execution of Complete Instruction

Execution of this instruction requires the following action :

1. Fetch instruction
2. Fetch first operand (Contents of memory location pointed at by the address field of the instruction)
3. Perform addition
4. Load the result into R1.

Following sequence of control steps are required to implement the above operation for the single-bus architecture that we have discussed in earlier section.

Microoperations

The operations performed on the data stored in registers known as micro-operations. Example: Shift, Count Clear and Load. The micro-operations are classified as follows-

1.Register transfer micro-operations:

These type of microoperations are used to transfer binary information from one register to another.

2.Arithmetic micro-operations :

These micro-operations are used to perform arithmetic operations on numeric data stored in the registers.

3.Logic micro-operations:

These microoperations perform bit manipulations on nonnumeric data stored in registers.

4.Shift micro operations:

They are used to perform shift operations on data stored in registers.

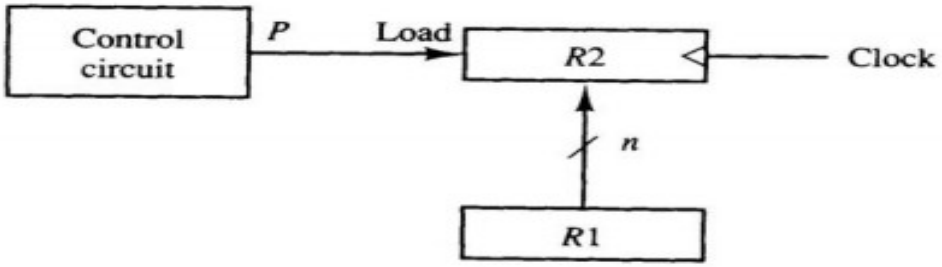
Microoperations

1. **Register transfer micro-operations**: These type of microoperations are used to transfer binary information from one register to another.

- Designate information transfer from one register to another by

R2 ← R1

- If the transfer is to occur only under a predetermined control condition, designate it by If (P = 1) then (R2 ← R1) or, P: R2 ← R1, where P



(a) Block diagram

Microoperations

2. Arithmetic micro-operations :

These micro-operations are used to perform arithmetic operations on numeric data stored in the registers.

Example	Description
R3 ← R1 + R2	Addition
R3 ← R1 - R2 (R1 + R2' + 1)	Subtraction
R2 ← R2'	Complement (really a logic operation)
R2 ← -R2 (R2' + 1)	Negation
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

Microoperations

3. Logic micro-operations:

- These microoperations perform bit manipulations on data stored in registers.
- Individual bits of registers are operated with other corresponding register bits. Example: the XOR of R2 and R1 is symbolized by

P: R1 ←R1 ⊕R2

Example: R1 = 1010 and R2 = 1100

1010	Content of R1
1100	Content of R2
0110	Content of R1 after P = 1

Microoperations

4. **Shift Micro-Operations:** –

These operations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operation.

The content of register can be shifted to the left or to the right. At the same time the bits are shifted, the flip flop receives the binary information from the serial input.

There are three types of shift micro operation-

1. **Logic shift** 2. **Circular shift** 3. **Arithmetic shift**

Logical Shift:- The symbol **"shl"** is used for **logical shift left** and **"shr"** is used for **logical shift right**.

Microoperations

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr} R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

Microoperations

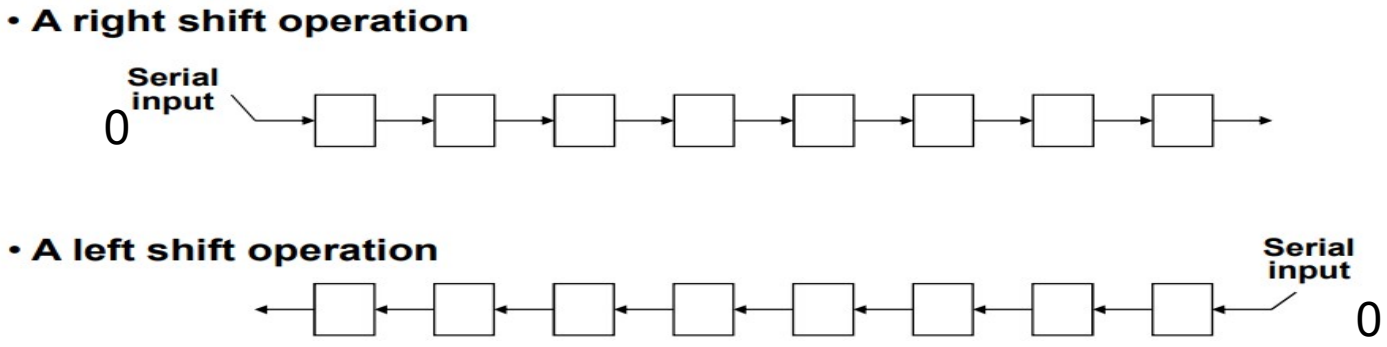
Logic shift –

- A *logical* shift is one that transfers 0 through the serial input.
- Symbols *shl* and *shr* for logical shift-left and shift-right microoperations.

For example:

$$R1 \leftarrow \text{shl } R1$$
$$R2 \leftarrow \text{shr } R2$$

are two microoperations that specify a 1-bit shift to the left of the content of register *R*1 and a 1-bit shift to the right of the content of register *R*2.



Microoperations

Circular shift –

•The *circular* shift (also known as a *rotate* operation) circulates the bits of the register around the two ends without loss of information.

•This is accomplished by connecting the serial output of the shift register to its serial input.

NIET Greater Noida
GET FUTURE READY

Program Control

- Program control is how a program makes decisions or organizes its activities. Program control typically involves executing particular code based on the outcome of a prior operation or a user input.

- **Program control** is how a **program** makes decisions or organizes its activities. **Program control** typically involves executing particular code based on the outcome of a prior operation or a user input.

- A **program control** instruction changes address value in the **PC** and hence the normal flow of execution.

- Change in **PC** causes a break in the execution of instructions. capability to branch to different **program** segments. Branch (BR) and Jump (JMP) instructions are used sometimes interchangeably but, they are different.

Program Control

CONDITIONAL BRANCH INSTRUCTIONS

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
	Unsigned compare conditions (A - B)	
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
	Signed compare conditions (A - B)	
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Program Control

Types of Program Control Instructions:

There are different types of Program Control Instructions:

1.Compare Instruction:

Compare instruction is specifically provided, which is similar to a subtract instruction except the result is not stored anywhere, but flags are set according to the result.

Example: CMP R1, R2 ;

2.Unconditional Branch Instruction:

It causes an unconditional change of execution sequence to a new location.

Example: JUMP L2

3.Conditional Branch Instruction:

A conditional branch instruction is used to examine the values stored in the condition code register to determine whether the specific condition exists and to branch if it does.

Program Control

4. **Subroutines:**
   A subroutine is a program fragment that lives in user space, performs a well-defined task. It is invoked by another user program and returns control to the calling program when finished.

Example: CALL and RET

5. **Halting Instructions:**

NOP Instruction – NOP is no operation. It cause no change in the processor state other than an advancement of the program counter. It can be used to synchronize timing.

HALT – It brings the processor to an orderly halt, remaining in an idle state until restarted by interrupt, trace, reset or external action.

Program Control

6. Interrupt Instructions:

Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced.

RESET – It reset the processor. This may include any or all setting registers to an initial value or setting program counter to standard starting location.

TRAP – It is non-maskable edge and level triggered interrupt. TRAP has the highest priority and vectored interrupt.

INTR – It is level triggered and maskable interrupt. It has the lowest priority. It can be disabled by resetting the processor.

Parallel Processing

- Parallel processing is a class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.

- Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time.

- The purpose of parallel processing is to speed up the computer processing capability and increase its **throughput, that is, the amount of processing that can be accomplished during a given interval of time.**

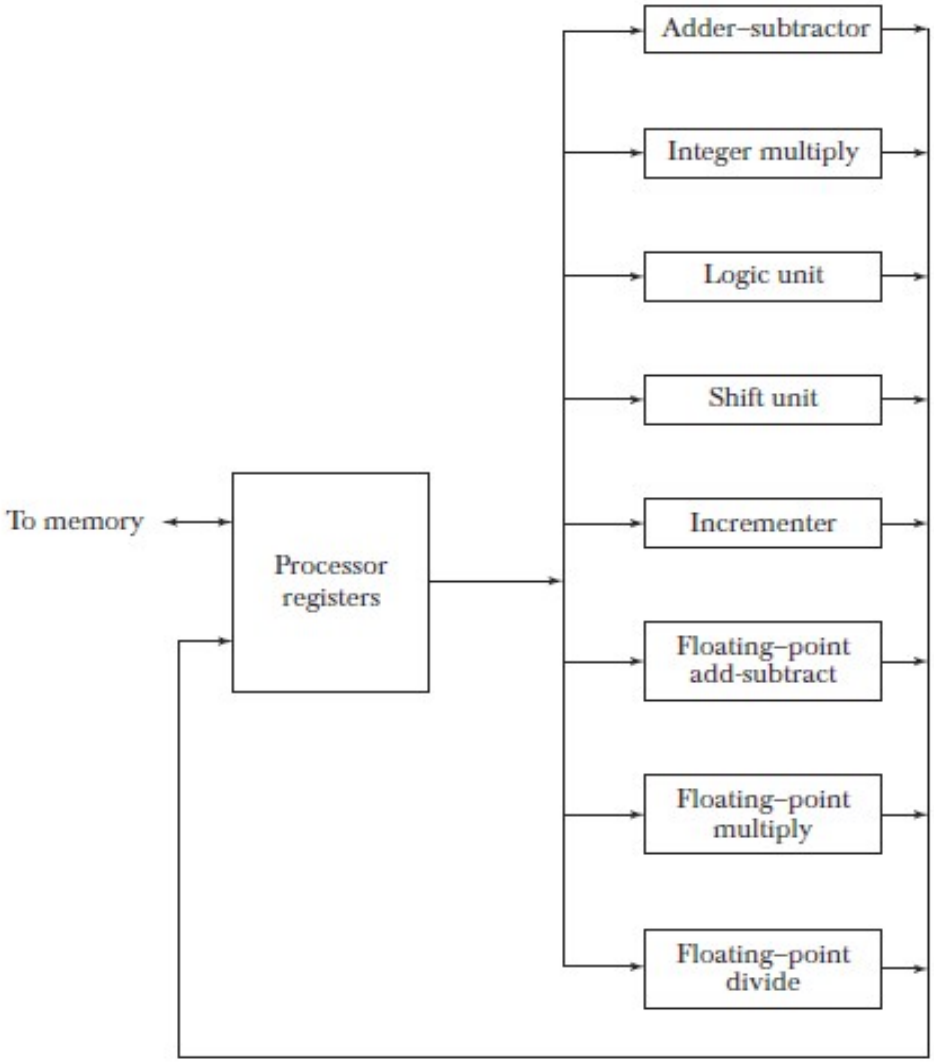
Parallel Processing

- Figure 9-1 shows one possible way of separating the execution unit into eight functional units operating in parallel. The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands.

- All units are independent of each other, so one number can be shifted while another number is being incremented. A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.

Parallel Processing

Figure 9-1 Processor with multiple functional units.



Parallel Processing- Flynn Classification

- There are a variety of ways that parallel processing can be classified.
- One classification introduced by M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.
- The normal operation of a computer is to fetch instructions from memory and execute them in the processor.
- **The sequence of instructions read from memory constitutes an *instruction stream.***
- **The operations performed on the data in the processor constitutes a *data stream.***

Parallel Processing

- Parallel processing may occur in the instruction stream, in the data stream, or in both. Flynn's classification divides computers into four major groups as follows:

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

Parallel Processing

- **SISD** represents the organization of a **single computer containing a control unit, a processor unit, and a memory unit**. Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.
- **SIMD** represents an organization **that *includes many processing units under the supervision of a common control unit***. All processors receive the same instruction from the control unit but operate on different items of data.
- **MISD** structure is only of theoretical interest since no practical system has been constructed using this organization.
- **MIMD** organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multicomputer systems can be classified in this category.

Parallel Processing

- One type of parallel processing that does not fit Flynn's classification is pipelining.
- The following are some representative applications of parallel processing computers:

1. Numerical weather forecasting,
2. computational aerodynamics,
3. finite-element analysis,
4. remote-sensing applications,
5. genetic engineering,
6. computer-asseted tomography, and
7. Weapon research and defence.

Pipelining

- Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess 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.

Pipelining

- The simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit.
- The register holds the data and the combinational circuit performs the suboperation in the particular segment.
- The output of the combinational circuit in a given segment is applied to the input register of the next segment.
- A clock is applied to all registers after enough time has elapsed to perform all segment activity. In this way the information flows through the pipeline one step at a time.

Pipelining

- Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

$$Ai * Bi + Ci \qquad \text{for } i = 1, 2, 3, \ldots, 7$$

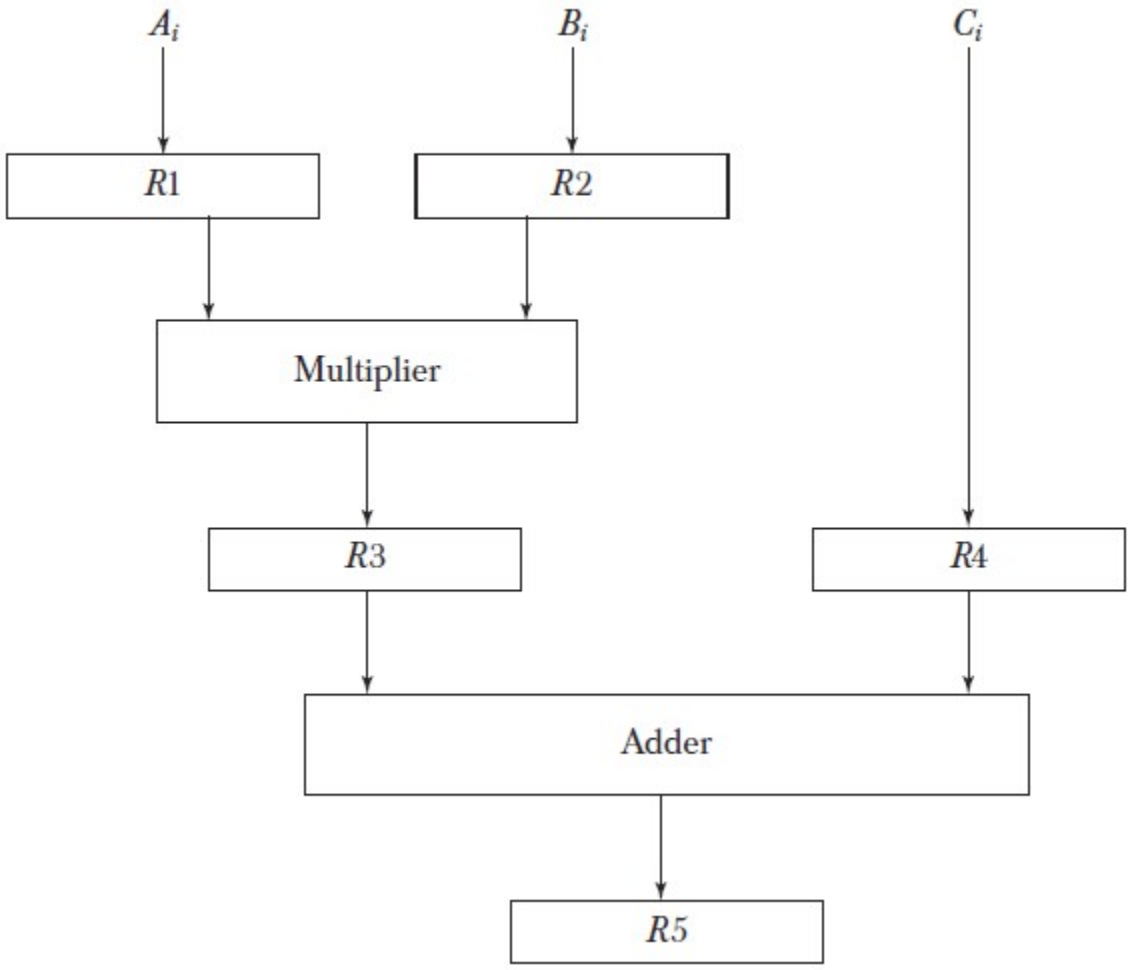
- Each suboperation is to be implemented in a segment within a pipeline.
- The suboperations performed in each segment of the pipeline are as follows:

$R1 \leftarrow Ai, R2 \leftarrow Bi$	Input *Ai* and B*i*
$R3 \leftarrow R1 * R2, R4 \leftarrow Ci$	Multiply and input *Ci*
$R5 \leftarrow R3 + R4$	Add *Ci* to product

- The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table.

Pipelining

Figure 9-2 Example or pipeline processing.



Pipelining

TABLE 9-1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	–	–	–
2	A_2	B_2	$A_1 * B_1$	C_1	–
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	–	–	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	–	–	–	–	$A_7 * B_7 + C_7$

Pipelining

General Considerations

- Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor.
- The general structure of a four-segment pipeline is illustrated in Fig. 9-3. Each segment consists of a combinational circuit *Si* that performs a suboperation over the data stream flowing through the pipe.
- The segments are separated by registers *Ri* that hold the intermediate results between the stages.
- We define a *task* as the total operation performed going through all the segments in the pipeline.
- The behaviour of a pipeline can be illustrated with a *space-time* diagram.
- This is a diagram that shows the segment utilization as a function of time.
- The space-time diagram of a four-segment pipeline is demonstrated in Fig. 9-4.

Pipelining

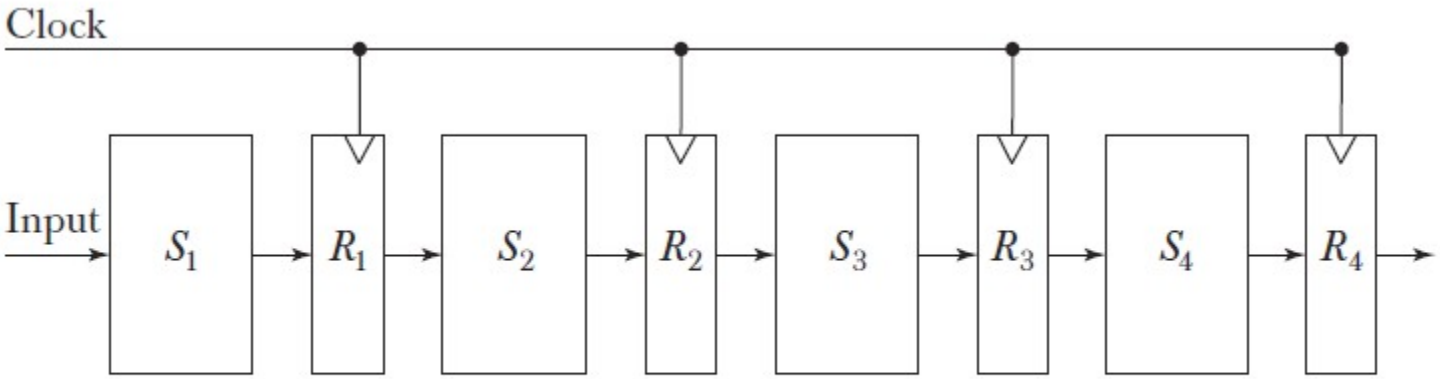


Figure 9-3 Four-segment pipeline.

Pipelining

Figure 9-4 Space-time diagram for pipeline.

	1	2	3	4	5	6	7	8	9	→ Clock cycles
Segment: 1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

Pipelining

- Consider the case where a ***k*-segment pipeline** with a **clock cycle time *tp*** is used to **execute *n* tasks**.
- The **first task *T*1 requires a time equal to *ktp*** to complete its operation since there are *k* segments in the pipe.
- The **remaining (*n* – 1) tasks** emerge from the pipe **at the rate of one task per clock cycle** and they will be completed after a time equal to **(*n* - 1)*tp***.
- Therefore, to complete *n* tasks using a *k*-segment pipeline requires

$k + (n - 1)$ clock cycles

$[k + (n - 1)]\ tp$

- Example, the diagram of Fig. 9-4 shows four segments and six tasks.
- The time required to complete all the operations is

$4 + (6 - 1) = 9$ clock cycles

Pipelining

- Consider a nonpipeline unit that performs the same operation and takes a time equal to t_n to complete each task.
- The total time required for *n* tasks is nt_n.
- The speedup of a pipeline processing over an equivalent nonpipelined processing is defined by the ratio

$$S = \frac{nt_n}{(k + n - 1)t_p}$$

- As the number of tasks increases, *n* becomes much larger than *k* - 1, and *k* + *n* - 1 approaches the value of *n*.
- Under this condition, the speedup becomes

$$S = \frac{t_n}{t_p}$$

Pipelining

- If we assume that the time it takes to process a task is the same in the pipeline and nonpipeline circuits, we will have $t_n = kt_p$. Including this assumption, the speedup reduces to

$$S = \frac{kt_p}{t_p} = k$$

Pipelining

- Let the time it takes to process a suboperation in each segment be equal to *tp* = 20 ns. Assume that the pipeline has *k* = 4 segments and executes *n* =100 tasks in sequence.

- The pipeline system will take

= (*k* + *n* - 1) *tp*

= (4 + 99) X 20

= 2060 ns to complete.

- Assuming that *tn* = *ktp* = 4 X 20 = 80 ns, a nonpipeline system requires *nktp* = 100 X 80 = 8000 ns to complete the 100 tasks.
- The speedup ratio is equal to 8000/2060 = 3.88.
- As the number of tasks increases, the speedup will approach 4, which is equal to the number of segments in the pipeline.

Pipelining

There are two areas of computer design where the pipeline organization is applicable.

An *arithmetic pipeline* divides an arithmetic operation into suboperations for execution in the pipeline segments.

An *instruction pipeline* operates on a stream of instructions by overlapping the fetch, decode, and execute phases of the instruction cycle.

Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- **An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments**.
- This causes the instruction fetch and execute phases to overlap and perform simultaneous operations.
- One possible digression associated with such a scheme is that an instruction may cause a branch out of sequence.
- In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.

Instruction Pipeline

- Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline.
- **The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer.**
- **This is a type of unit that forms a queue** rather than a stack.
- Whenever the execution unit is not using memory, the control increments the program counter and uses its address value to read consecutive instructions from memory.
- The instructions are inserted into the FIFO buffer so that they can be executed on a first-in, first-out basis.

Instruction Pipeline

Instruction Cycle-

•Computers with complex instructions require other phases in addition to fetch and execute to process an instruction completely.

•In the most general case, the computer needs to process each instruction with the following sequence of steps

1. Fetch the instruction from memory.

2. Decode the instruction.

3. Calculate the effective address.

4. Fetch the operands from memory.

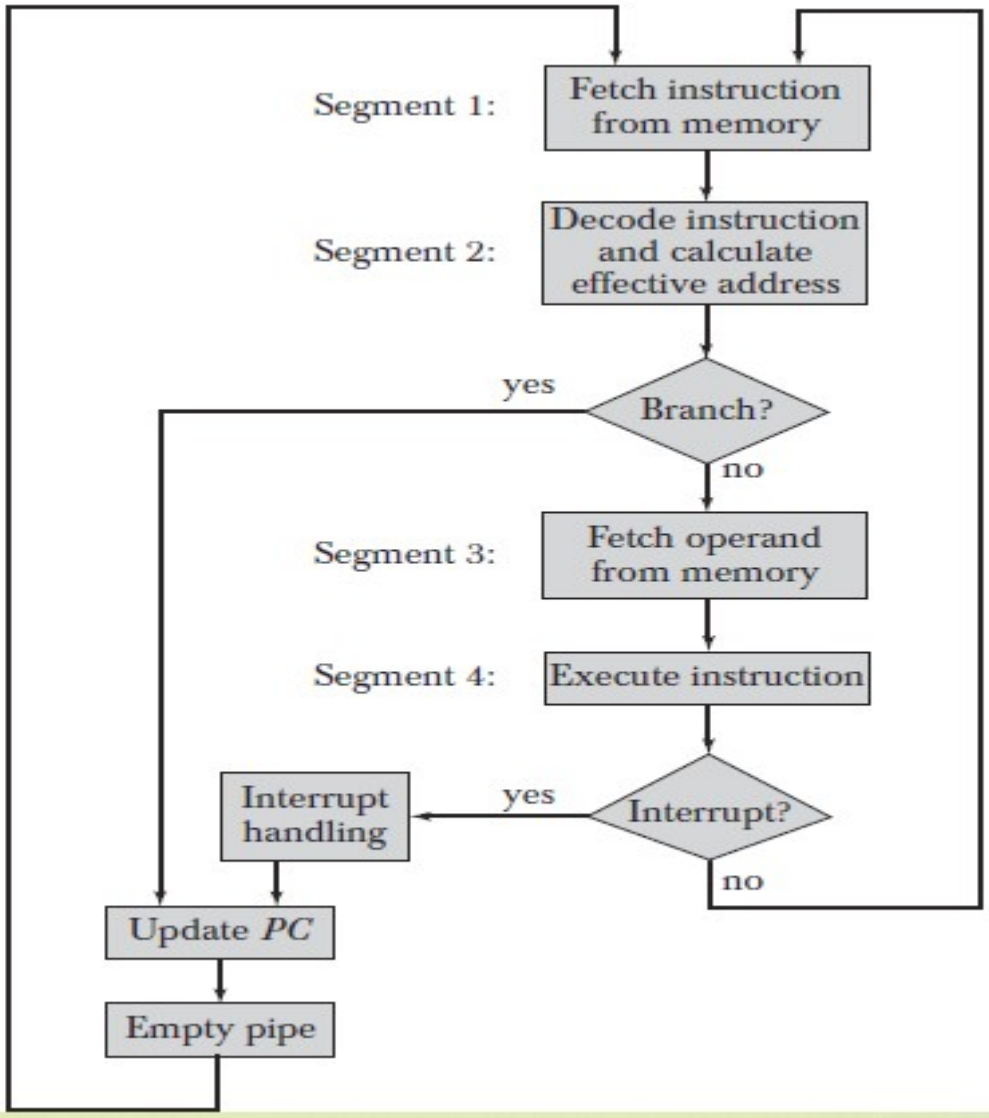
5. Execute the instruction.

6. Store the result in the proper place.

Example: Four-Segment Instruction Pipeline

- Figure 9-7 shows how the instruction cycle in the CPU can be processed with a four-segment pipeline.
- While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3.
- **Once in a while, an instruction in the sequence may be a program control type that causes a branch out of normal sequence**.
- In that case the pending operations in the last two segments are completed and **all information stored in the instruction buffer is deleted**. The pipeline then restarts from the new address stored in the program counter.
- **Similarly, an interrupt request, when acknowledged, will cause the pipeline to empty and start again from a new address value.**

Example: Four-Segment Instruction Pipeline



Example: Four-Segment Instruction Pipeline

- Figure 9-8 shows the operation of the instruction pipeline.
- The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.

1. FI is the segment that fetches an instruction.
2. DA is the segment that decodes the instruction and calculates the effective address.
3. FO is the segment that fetches the operand.
4. EX is the segment that executes the instruction.

- It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.

Example: Four-Segment Instruction Pipeline

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Figure 9-8 Timing of instruction pipeline.

Pipeline Conflicts

In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1.Resource conflicts caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2.Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

3.Branch difficulties arise from branch and other instructions that change the value of *PC.*

Reduced Instruction Set Computer (RISC)

- An important aspect of computer architecture is the design of the instruction set for the processor.

- These are categorised into

1. RISC (**Reduced Instruction Set Computer**)
2. CISC (**Complex Instruction Set Computer**)

- **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.
- **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of increase in number of cycles per instruction.

Reduced Instruction Set Computer (RISC)

Example – Suppose we have to add two 8-bit number:

- **CISC approach:** There will be a single command or instruction for this like ADD which will perform the task.
- **RISC approach:** Here programmer will write the first load command to load data in registers then it will use a suitable operator and then it will store the result in the desired location.

So, add operation is divided into parts i.e. load, operate, store due to which RISC programs are longer and require more memory to get stored but require fewer transistors due to less complex command.

Reduced Instruction Set Computer (RISC)

1. **RISC**

- Stands for **Reduced Instruction Set Computer Processor**, a microprocessor architecture with a simple collection and highly customized set of instructions.

- It is built to minimize the instruction execution time by optimizing and limiting the number of instructions.

- It means each instruction cycle requires only one clock cycle, and each cycle contains three parameters: fetch, decode and execute.

- The RISC processor is also used to perform various complex instructions by combining them into simpler ones. RISC chips require several transistors, making it cheaper to design and reduce the execution time for instruction.

Reduced Instruction Set Computer (RISC)

Advantages of RISC Processor

1.The RISC processor's performance is better due to **the simple and limited number of the instruction set.**

2.It requires several transistors that make **it cheaper to design.**

3.RISC allows the instruction to use free space on a microprocessor because of its simplicity.

4.RISC processor is simpler than a CISC processor because of its simple and quick design, **and it can complete its work in one clock cycle**.

Reduced Instruction Set Computer (RISC)

Disadvantages of RISC Processor

1.The RISC processor's performance may vary according to the code executed because subsequent instructions may depend on the previous instruction for their execution in a cycle.

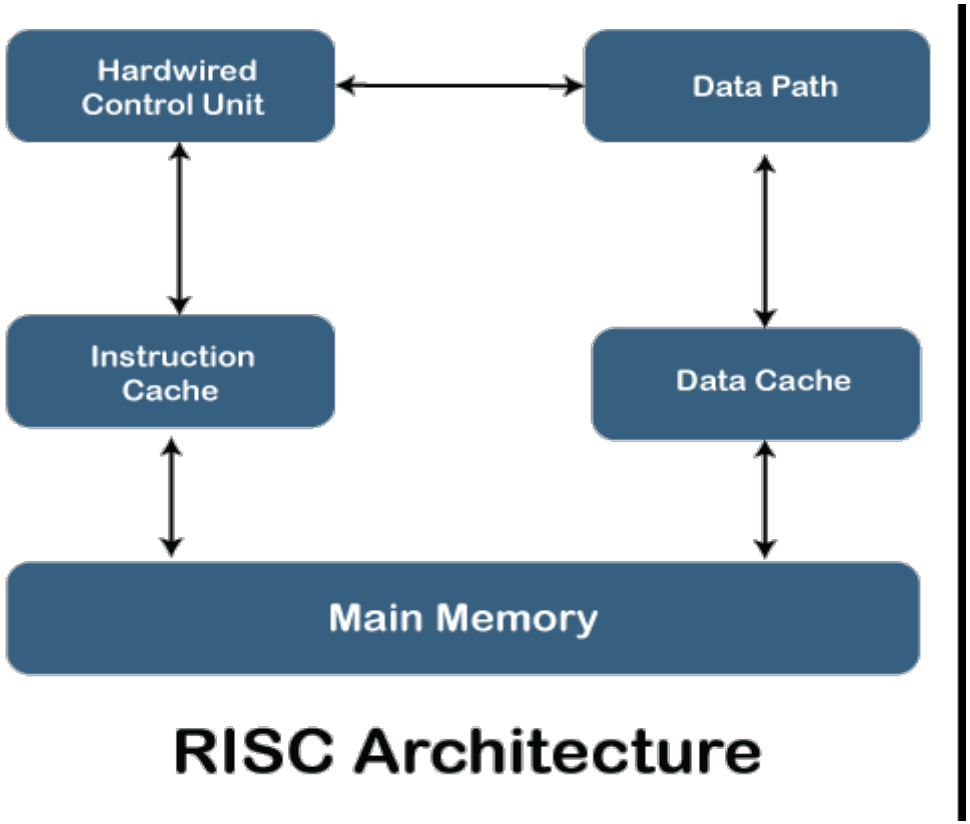
2.Programmers and compilers often use complex instructions.

3.RISC processors require very fast memory to save various instructions that **require a large collection of cache memory** to respond to the instruction in a short time.

Reduced Instruction Set Computer (RISC)

RISC Architecture

It is a highly customized set of instructions used in portable devices due to system reliability



RISC Architecture

Reduced Instruction Set Computer (RISC)

Features of RISC Processor

1.One cycle execution time: For executing each instruction in a computer, the RISC processors require one CPI (Clock per cycle). And each CPI includes the fetch, decode and execute method applied in computer instruction.

2.Pipelining technique: The pipelining technique is used in the RISC processors to execute multiple parts or stages of instructions to perform more efficiently.

3.A large number of registers: RISC processors are optimized with multiple registers that can be used to store instruction and quickly respond to the computer and minimize interaction with computer memory.

Reduced Instruction Set Computer (RISC)

4. It supports a simple addressing mode and fixed length of instruction for executing the pipeline.
5. It uses LOAD and STORE instruction to access the memory location.
6. Simple and limited instruction reduces the execution time of a process in a RISC.

Complex Instruction Set Computer (CISC)

- It has **a large collection of complex instructions that range from simple to very complex and specialized in the assembly language level, which takes a long time to execute the instructions.**

- So, **CISC approaches reducing the number of instruction on each program and ignoring the number of cycles per instruction.**

- It emphasizes to build complex instructions directly in the hardware because the hardware is always faster than software. However, **CISC chips are relatively slower as compared to RISC chips but use little instruction than RISC.**

Complex Instruction Set Computer (CISC)

Advantages of CISC Processors

1.The compiler requires **little effort to translate high-level programs or statement languages into assembly or machine language** in CISC processors.

2.The **code length is quite short, which minimizes the memory requirement.**

3.To store the instruction on each CISC, it requires very less RAM.

4.Execution of a single instruction requires several **low-level tasks**.

5.CISC creates a process to manage power usage that adjusts clock speed and voltage.

6.It **uses fewer instructions set to perform the same instruction as the RISC**.

Complex Instruction Set Computer (CISC)

Disadvantages of CISC Processors

1.CISC chips are slower than RSIC chips to execute per instruction cycle on each program.

2.The performance of the machine decreases due to the slowness of the clock speed.

3.Executing the pipeline in the CISC processor makes it complicated to use.

4.The CISC chips require more transistors as compared to RISC design.

5.In CISC it uses only 20% of existing instructions in a programming event.

Complex Instruction Set Computer (CISC)

Characteristics of CISC Processor

Following are the main characteristics of the RISC processor:

1.The length of the code is shorts, so it requires very little RAM.

2.CISC or complex instructions may take longer than a single clock cycle to execute the code.

3.Less instruction is needed to write an application.

4.It provides easier programming in assembly language.

5.Support for complex data structure and easy compilation of high-level languages.

6.It is composed of fewer registers and more addressing nodes, typically 5 to 20.

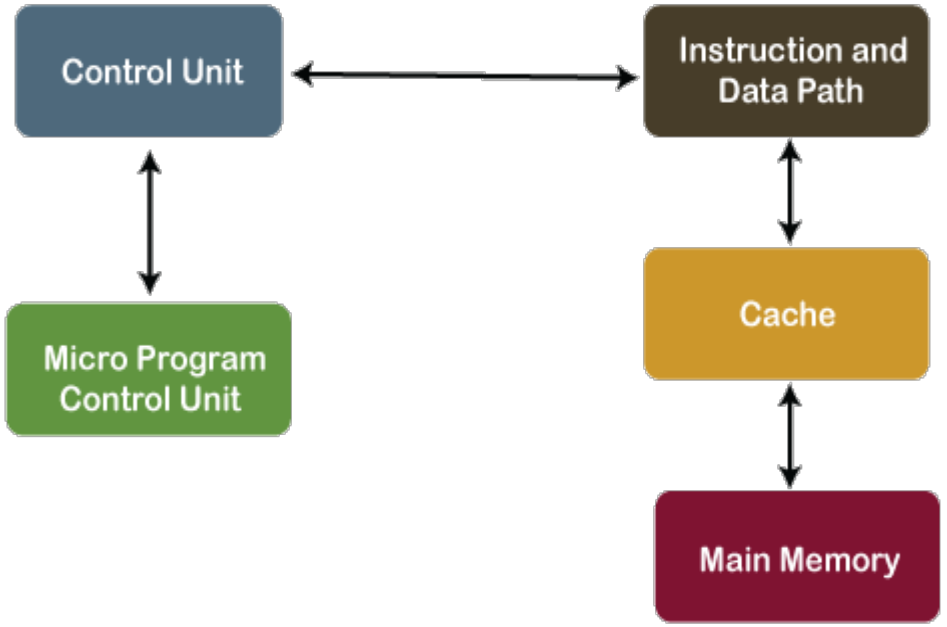
7.Instructions can be larger than a single word.

8.It emphasizes the building of instruction on hardware because it is faster to create than the software.

Complex Instruction Set Computer (CISC)

CISC Processors Architecture

The CISC architecture helps reduce program code by embedding multiple operations on each program instruction, which makes the CISC processor more complex.



CISC Architecture

Comparison

RISC	CISC
It is a Reduced Instruction Set Computer.	It is a Complex Instruction Set Computer.
It is a hard wired unit of programming in the RISC Processor.	Microprogramming unit in CISC Processor.
It requires multiple register sets to store the instruction.	It requires a single register set to store the instruction.
RISC has simple decoding of instruction.	CISC has complex decoding of instruction.
Uses of the pipeline are simple in RISC.	Uses of the pipeline are difficult in CISC.

Comparison

RISC	CISC
It uses a limited number of instruction that requires less time to execute the instructions.	It uses a large number of instruction that requires more time to execute the instructions.
RISC has more transistors on memory registers.	CISC has transistors to store complex instructions.
The execution time of RISC is very short.	The execution time of CISC is longer.
RISC architecture can be used with high-end applications like telecommunication, image processing, video processing, etc.	CISC architecture can be used with low-end applications like home automation, security system, etc
It has fixed format instruction.	**It has variable format instruction.**
The program written for RISC architecture needs to take more space in memory.	Program written for CISC architecture tends to take less space in memory.

Faculty Video Links, You tube Courses Details

You tube/other Video Links

- https://www.youtube.com/watch?v=vcvgvqnH7GA
- https://www.youtube.com/watch?v=U62iP8RkZIk
- https://www.youtube.com/watch?v=8b1Cs1Uf6hI
- https://www.youtube.com/watch?v=MSac_s-W0pc
- https://www.youtube.com/watch?v=sJdCD_APVq8
- https://www.youtube.com/watch?v=_EKgwOAAWZA

Expected Questions for University Exam

- Write a program to evaluate the arithmetic expression by using Three, Two, One and Zero address instruction. **X = (A+B*C) / (D+E*F/G+H).**
- Differentiate between RISC & CISC based microprocessor.
- What is micro programmed control unit? Give the basic structure of micro programmed control unit.
- How pipeline performance can be measured? Discuss. Give a space time diagram for visualizing the pipeline behavior for a four stage pipeline.
- Perform Shift micro operation for given data 11001010.