UNIT-2

Architecture of 8086:

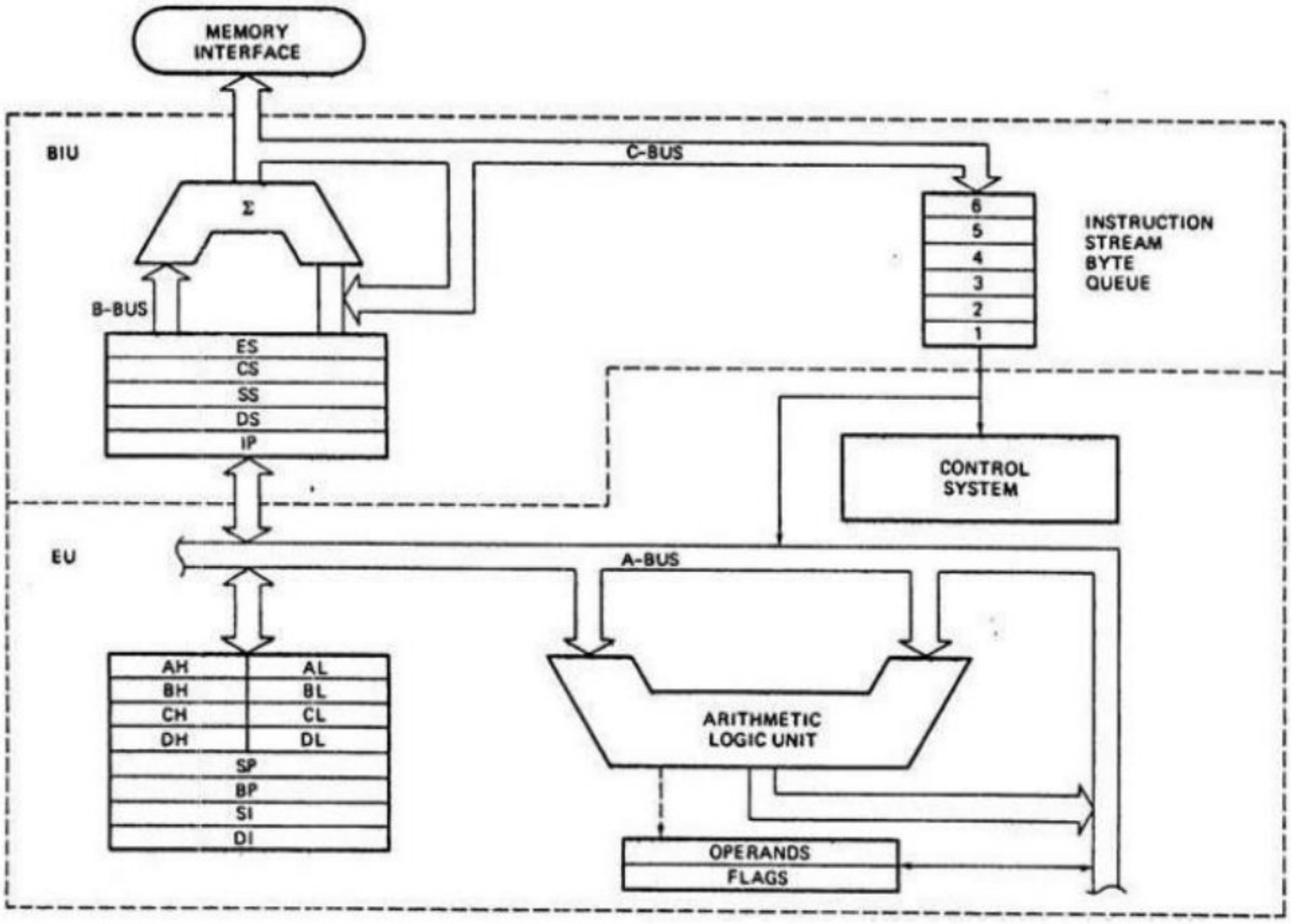


Fig: 8086 internal Architecture

The architecture of 8086 supports a 16-bit ALU , a set of 16-bit registers, and provides segmented memory addressing capability,fetched instruction queue for overlapped fetching and execution.

- Architecture of 8086 is pipeline type of architecture.
- The architecture of 8086 is divided into two functional parts i.e.,

i. **Execution unit (EU)**

ii. **Bus interface unit (BIU)**

These two units work asynchronously.

- Functional division of architecture speeds up the processing, since BIU and EU operate parallelly and independently i.e., EU executes the instructions and BIU fetches another instruction from the memory simultaneously.
- As the whole architecture is divided into two independent functional parts and both the subsystem's operations can be overlapped, hence the architecture is PIPELINING type of architecture.

EXECUTION UNIT

- The execution unit informs the BIU of the processor regarding from where to fetch the instructions from and then executes these instructions.
- The execution unit consists of the following:

- General purpose registers
- Stack pointer
- Base pointer
- Index registers
- ALU
- Flag register(FLAGS/ PSW)
- Instruction decoder
- Timing and control unit

Functions of EU

- Tells BIU regarding from where to fetch instructions or to read data.
- Receives opcode of an instruction from the queue.
- decodes the instructions.
- Executes the instruction.

Functions of various parts of EU

- Control circuitry: Directs internal operations.
- Instruction Decoder: Translates instructions fetched from memory into series of actions.
- ALU: Performs arithmetic and logical operations.
- FLAGS: Reflects the status of program.
- General purpose registers: Used to store Temporary data.
- Index and Pointer registers: Specifies/ informs about offset of operand

BUS INTERFACE UNIT

- The BIU handles transfer of data and address between the processor and memory/ I/O devices by computing address (Physical/ Effective address) and send the computed address to memory / I/O and fetches instruction codes then stores them in FIFO register set called Queue register.
- The BIU consists of the following:
  - ❖ Segment Registers
  - ❖ Instruction pointer
  - ❖ 6-Byte instruction Queue Register

Functions of BIU

- Handles transfer of data and address between processor and memory / I/O devices.
- Compute physical address and send it to memory interfaces.
- Fetches instruction codes and stores it in Queue
- Reads/Writes data from/to memory/ I/O devices

Functions of various parts of BIU

- **Segment registers :** Used to hold the starting address of the segment registers.
- **Queue register:** Used to store pre fetched instructions and inputs it to EU.
- **Instruction Pointer:** Used to point to the next instruction to be executed by EU.
- While the EU is decoding an instruction or executing an instruction which does not require use of the buses, the BIU fetches up to six instruction bytes that will be following the present instruction from memory and stores them in the queue register simultaneously.

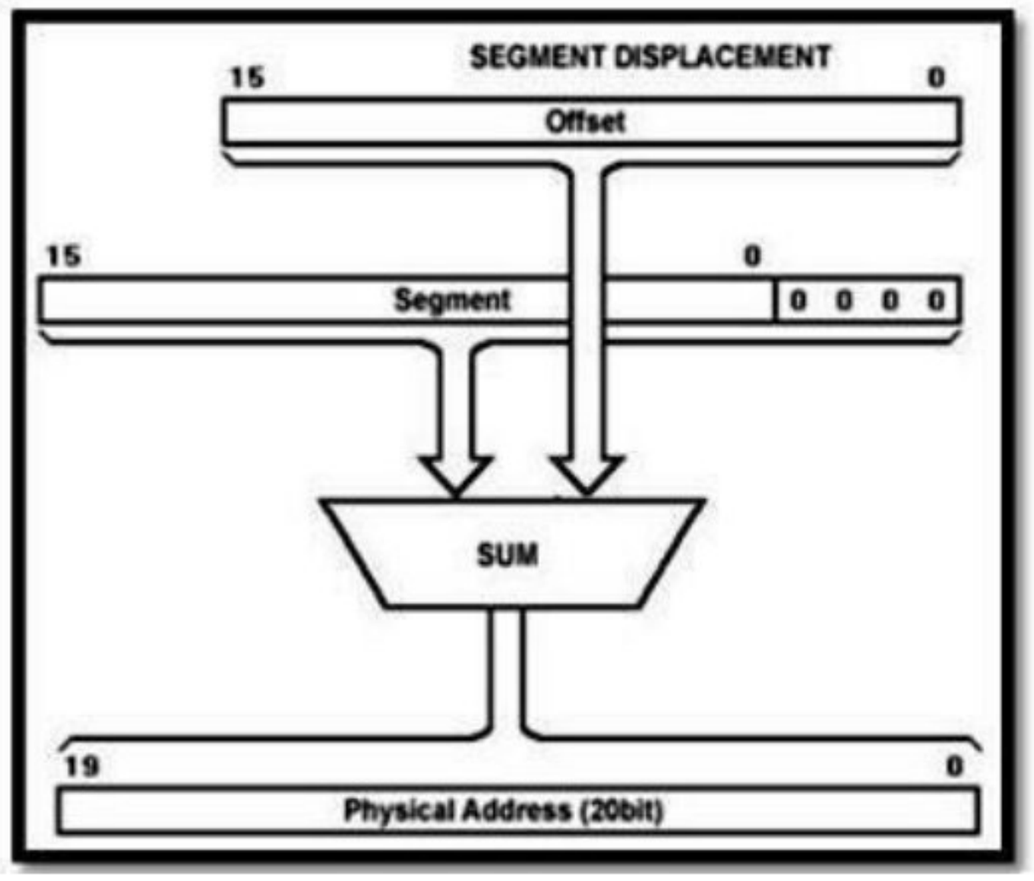
Logical and Physical Address

- Addresses within a segment can range from address 00000h to address 0FFFFh. This corresponds to the 64K-bytelength of the segment. An address within a segment is called an offset or logical address.
- A logical address gives the displacement from the base address of the segment to the desired location within it, as opposed to its "real" address, which maps directly anywhere into the 1 MByte memory space. This "real" address is called the physical address.

Difference between the physical and the logical address:

- The physical address is 20 bits long and corresponds to the actual binary code output by the BIU on the address bus lines. The logical address is an offset from location 0 of a given segment.

Segment address ⟶ 1005H
Offset address ⟶ 5555H
Segment address ⟶ 1005H ⟶ 0001 0000 0000 0101
Shifted by 4 bit positions ⟶ 0001 0000 0000 0101 0000
+
Offset address ⟶ 0101 0101 0101 0101
Physical address ⟶ 0001 0101 0101 1010 0101
1 5 5 A 5



a physical address while addressing memory. The segment address value is to be taken from an appropriate segment register depending upon whether code, data or stack are to be accessed, while the offset may be the content of IP, BX, SI, DI, SP, BP or an immediate 16-bit value, depending upon the addressing mode.

In case of 8085, once the opcode is fetched and decoded, the external bus remains free for some time, while the processor internally executes the instruction. This time slot is utilised in 8086 to achieve the overlapped fetch and execution cycles. While the fetched instruction is executed internally, the external bus is used to fetch the machine code of the next instruction and arrange it in a queue known as predecoded instruction byte queue. It is a 6 bytes long, first-in first-out structure. The instructions from the queue are taken for decoding sequentially. Once a byte is decoded, the queue is rearranged by pushing it out and the queue status is checked for the possibility of the next opcode fetch cycle. While the opcode is fetched by the Bus Interface Unit (BIU), the Execution Unit (EU) executes the previously decoded instruction concurrently. The BIU along with the Execution Unit (EU) thus forms a pipeline. The bus interface unit, thus manages the complete interface of execution unit with memory and I/O devices, of course, under the control of the timing and control unit.

The execution unit contains the register set of 8086 except segment registers and IP. It has a 16-bit ALU, able to perform arithmetic and logic operations. The 16-bit flag register reflects the results of execution by the ALU. The decoding unit decodes the opcode bytes issued from the instruction byte queue. The timing and control unit derives the necessary control signals to execute the instruction opcode received from the queue, depending upon the information made available by the decoding circuit. The execution unit may pass the results to the bus interface unit for storing them in memory.

Flag register of 8086

D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
				O	D	I	T	S	Z		AC		P		CY

There are total 9 flags in 8086 and the flag register is divided into two types:

(a) Status Flags – There are 6 flags in 8086 microprocessor which become set(1) or reset(0) depending upon condition after either 8-bit or 16-bit operation. These flags are conditional/status flags.

The 6 status flags are:

(b) Sign Flag (S): This flag is set when the result of any computation is negative.

(c) Zero Flag (Z): This flag is set when the result of any computation or comparison performed is zero.

(d) Auxiliary Cary Flag (AC): This flag is set when there is acarry from the lower nibble.

(e) Parity Flag (P): This flag is set when the lower byte of the result contains even number of 1's .

(f) Overflow Flag: This flag will be set (1) if the result of a signed operation is too large to fit in the number of bits available to represent it, otherwise reset (0).(eg:50+32= 82)

(g) Carry Flag (CY)): This flag is set when there is a carry out of the MSB in case of addition or a borrow in case of subtraction.

Control Flags – The control flags enable or disable certain operations of the microprocessor.There are 3 control flags in 8086 microprocessor and these are:

Directional Flag (D) – This flag is specifically used by string manipulation instructions string instructions. If this flag is 0,the string is processed beginning from the lowest address to the highest address. If this flag is 1,the string is processed beginning from the highest address to the lowest address.

Interrupt Flag: If interrupt flag is set (1), the microprocessor will recognize interrupt requests from the peripherals.

If interrupt flag is reset (0), the microprocessor will not recognize any interrupt requests and will ignore them.

Trap Flag (T) –Setting trap flag puts the microprocessor into single step mode for debugging.

INSTRUCTION SET ARCHITECTURE OF CPU

Register Transfer Language:

- A digital computer system exhibits an interconnection of digital modules such as registers, decoders, arithmetic elements, and Control logic. These digital modules are interconnected with some common data and control paths to form a complete digital system. Digital modules are best defined by the registers and the operations that are performed on the data stored in them.
- The **operations performed on the data stored in registers are called Micro-operations**. A microoperation is an elementary operation performed on the information stored in oneor more registers. The result of the operation may replace the previous binary information of a register or may be transferred to another register. Examplesof microoperations are shift, count, clear, and load.
- The **Register Transfer Language** is the **symbolic representation of notations used to specify the sequence of micro-operations**.

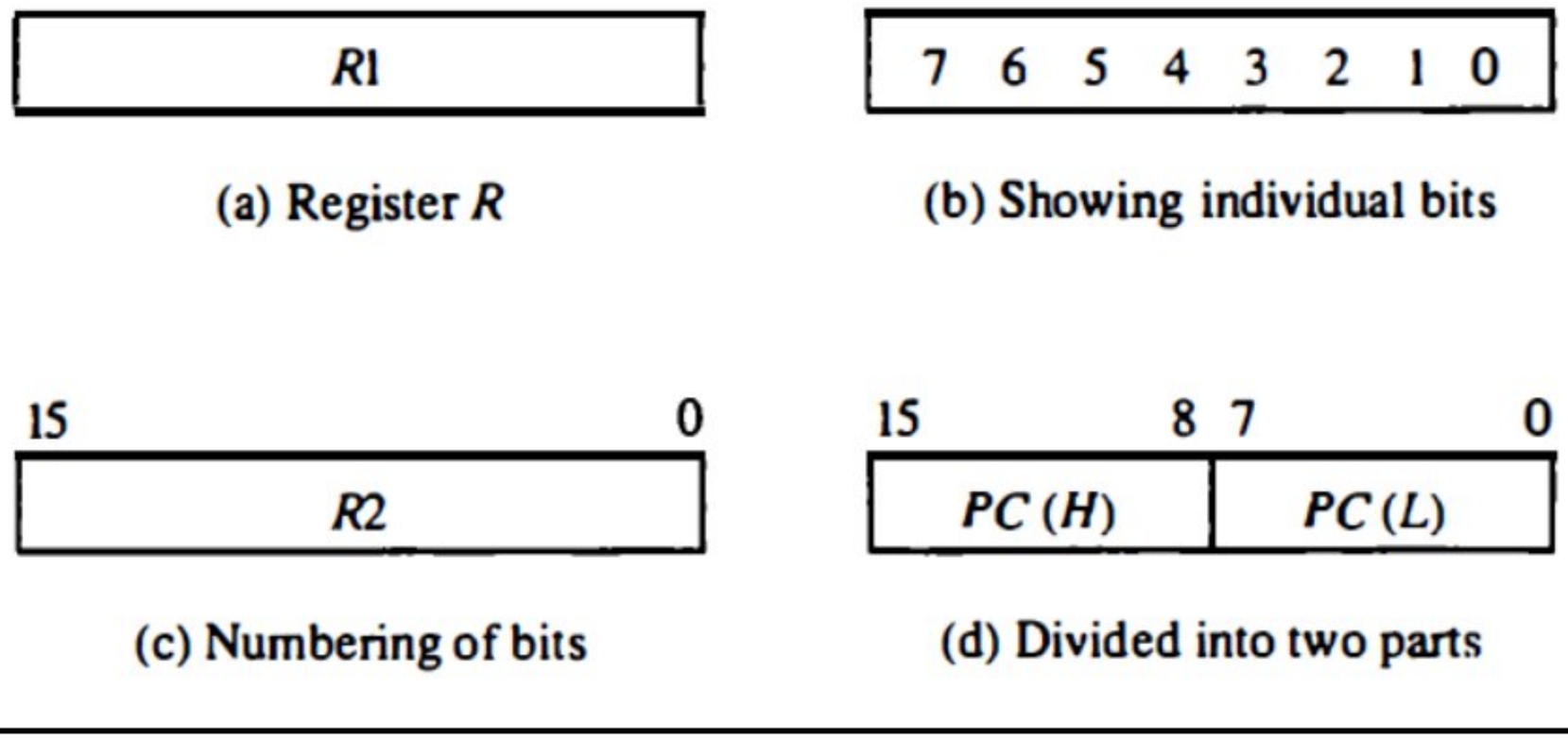
In a computer system, **data transfer takes place between processor registers and memory and between processor registers and input-output systems**. These data transfer can be represented by standard notations given below:

- Notations R0, R1, R2..., and so on represent processor registers.
- The addresses of memory locations are represented by names such as LOC, PLACE, MEM, etc.
- Input-output registers are represented by names such as DATA IN, DATA OUT and so on.
- The content of register or memory location is denoted by placing square brackets around the name of the register or memory location.

Register Transfer:

Computer registers are denoted by capital letters (sometimes followed by numerals) to denote the function of the register. The register that holds an address for the memory unit is usually called a **memory address register** and is denoted by **MAR**. Other registers are PC (for program counter), IR (for instruction register, and R1 (for processor register). An n-bit register is sequence of n-flipflops numbered from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left.

The most common way to represent a register is by a rectangular box with the name of the register inside, as shown in the figure below. The individual bits can be distinguished as shown in (b). The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c). A16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H(for high byte). The name of the 16-bit register is PC. The symbol PC (0-7) or PC (L) refers to the low-order byte and PC(8-15) or PC(H) to the high-order byte.



(a) Register *R*

(b) Showing individual bits

(c) Numbering of bits

(d) Divided into two parts

Fig: Block diagram of registers

Information transfer from one register to another is designated in symbolicform by means of a replacement operator as shown below, which denotes a transfer of the contents of register R1 into register R2.Contents of R2 are replaced by the contents of R1.By definition, thecontent of the source register R1 does not change after the transfer.register transfer implies that circuits areavailable from the outputs of the source register to the inputs of the destinationregister.

R2 <--R1

Sometimes, we may want the transfer to occur only under a predetermined controlcondition. This can be shown by means of an if-then statement

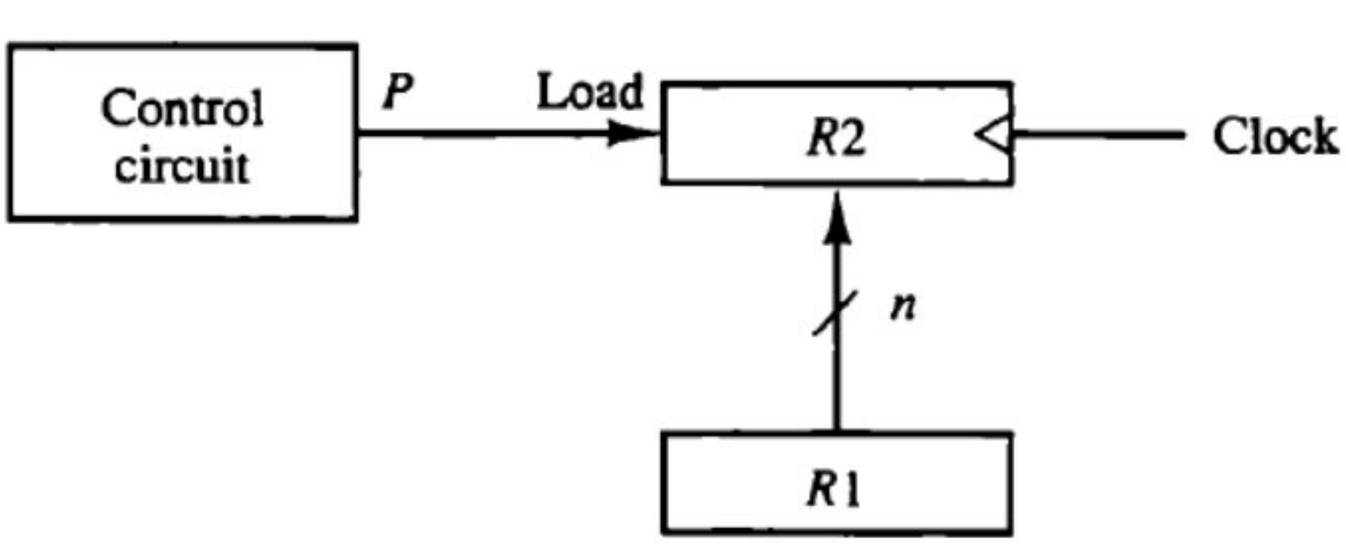
If (P = 1) then (R2 <--R1)

where P is a control signal generated in the control section.A control function is a Boolean variable that isequal to 1 or 0. The control function is included in the statement as follows

P: R2 <--R1

The control condition is terminated with a colon. It symbolizes the requirementthat the transfer operation be executed by the hardware only if P = 1.

Every statement written in a register transfer notation implies a hardwareconstruction for implementing the transfer. Figure below shows the block diagramthat depicts the transfer from R1 to R2. The n outputs of register R1 areconnected to the n inputs of register R2. The letter n will be used to indicateany number of bits for the register. It will be replaced by an actual numberwhen the length of the register is known. Register R2 has a load input that isactivated by the control variable P. It is assumed that the control variable issynchronized with the same clock as the one applied to the register.



In the timing diagram below, P is activated in the control section by the rising edgeof a clock pulse at time t . The next positive transition of the clock at time t + 1finds the load input active and the data inputs of R2 are then loaded into theregister in parallel. P may go back to 0 at time t + 1; otherwise, the transferwill occur with every clock pulse transition while P remains active.

Note:Even though the control condition such as P becomes active just after time t,the actual transfer does not occur until the register is triggered by the nextpositive transition of the clock at time t + 1 .

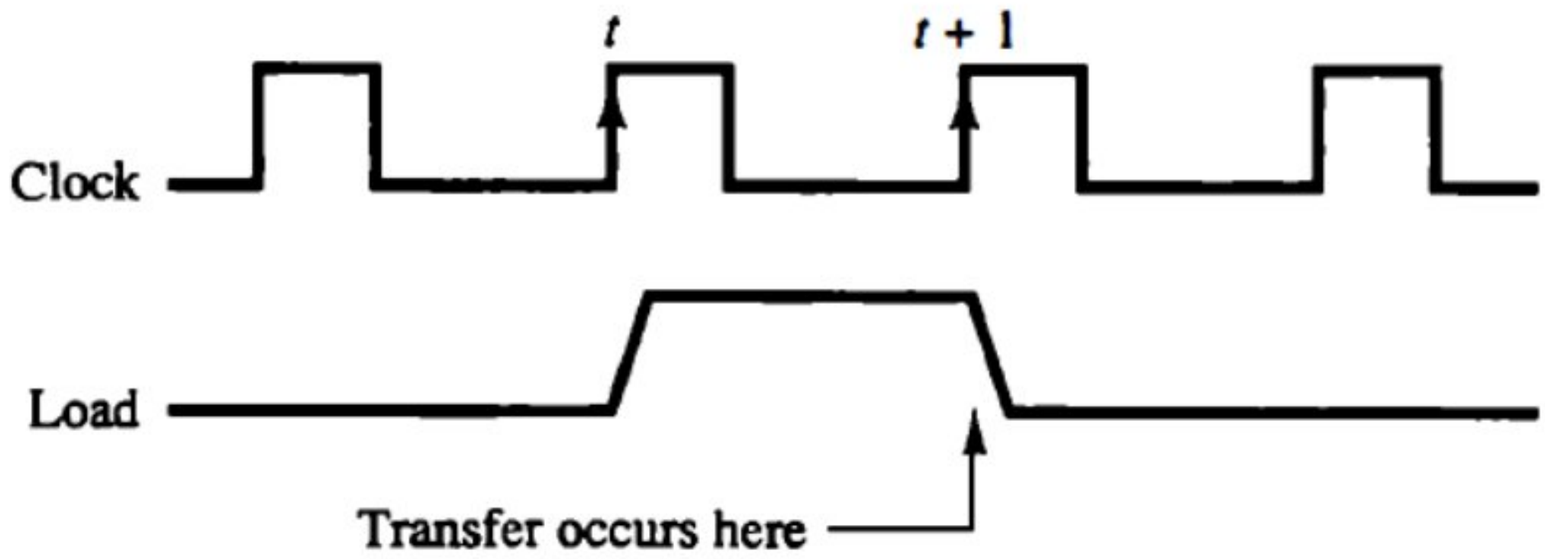


Fig: Timing Diagram

Registers are denoted by capital letters, and numerals may follow the letters. Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register. The arrow denotes a transfer of information and the direction of transfer. A comma is used to separate two or more operations that are executed at the same time.

The statement

T: R2 <- R1, R1 <- R2

It denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1.

The basic symbols of the register transfer notation are given below:

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R2*
Parentheses ( )	Denotes a part of a register	*R2*(0–7), *R2*(*L*)
Arrow ←	Denotes transfer of information	*R2* ← *R1*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *R2*

Fig: Basic symbols of register Transfer

Memory Transfer:

The transfer of information from a memory word to the outside environment is called a read operation. The transfer of new information to be stored into the memory is called a write operation. A memory word will be symbolized by the letter M.

The particular memory word among the many available is selected by thememory address during the transfer. It is necessary to specify the address ofM when writing memory transfer operations. This will be done by enclosingthe address in square brackets following the letter M.

Memory Read: Consider a memory unit that receives the address from a register, called the address register, symbolized by AR. The data are transferred to another register, called the data register, symbolized by DR. The read operation can be stated as follows:

Read: DR <- M [AR]

This causes a transfer of information into DR from the memory word M selected by the address in AR.

Memory Write: The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data is in register R1 and the address in AR. The write operation can be stated symbolically as follows:

Write: M[AR] ← R1

This causes the transfer of information from R1 into the memory word M selected by the address in AR.

Instruction cycle:

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.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

FETCH AND DECODE: Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal To. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on.

The Micro-operations for the fetch and decode phases can be specified by the following register transfer statements:

T_0: ***AR ← PC***

The address from PC to AR during the clock transition associated with timing signal T0.

T_1: ***IR ← M[AR], PC ← PC + 1***

The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program

T_2: ***D0, …, D7 ← Decode IR(12-14),AR ← IR(0-11), I ← IR(l5)***

At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR.

Decoding: The timing signal that is active after the decoding is T3. During time T3, the control unit determines the type of instruction that was just read from memory. Decoder output D7, 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 D7 = 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 D7 = 0 and I = 1, we have a

memoryreferenceinstruction with an indirect address.The microoperation for the indirect addresscondition can be symbolized by the register transfer statement:

AR ← M [AR]

$D'_7\, IT_3$: $AR \leftarrow M[AR]$
$D'_7\, I'T_3$: Nothing
$D_7\, I'T_3$: Execute a register-reference instruction
$D_7 IT_3$: Execute an input–output instruction

When a **memory-reference instruction** with I = 0 is encountered, it is notnecessary to do anything since the effective address is already in AR. However,the sequence counter SC must be incrementedso that theexecution of the memory-reference instruction can be continued with timingvariable T4.After the instruction is executed,SC is cleared to 0 and control returns to the fetch phase with T0 = 1 .

Register-reference instructions are recognized by the control when 07 = 1 andI = 0.The 12 bitsavailable in IR(0-11) are transferred to AR during time T2.These instructions are executed with the clocktransition associated with timing variable T3.The execution of a register-reference instruction is completedat time T3.The sequence counter SC is cleared to 0 and the control goesback to fetch the next instruction with timing signal T0.

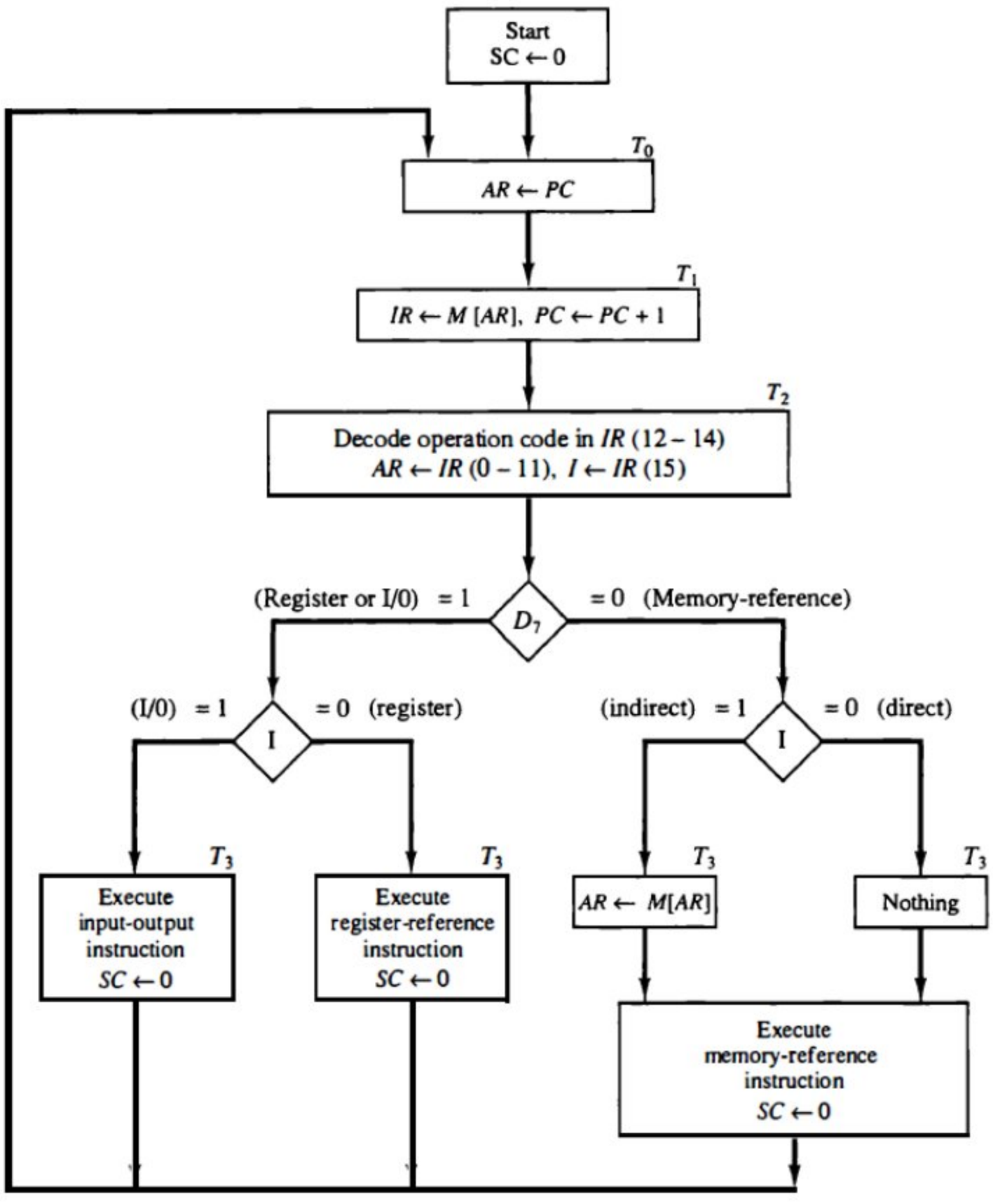


Fig:Flowchart for instruction cycle

Addressing Modes

ADDRESSING MODES OF 8086:

Addressing modes is the manner in which operands are given in an instruction.The addressing modes of 8086 are as follows:

1) **IMMEDIATE ADDRESSING MODE:**In this mode the operand is specified in theinstruction itself. Instructions are longer but the operands are easily identified.
   Eg: MOV CL, 12H ; **Moves 12 immediately into CL register**
   MOV BX, 1234H ; **Moves 1234 immediately into BX register**
2) **REGISTER ADDRESSING MODE**:In this mode operands are specified using registers. Instructions are shorter but operands cant be identified by looking at the instruction.
   Eg: MOV AX,BX
   ADD BX,CX
3) **DIRECT ADDRESSING MODE:**In this mode address of the operand is directly specified in the instruction.
   Eg:MOV CL, [2000H] ; CL Register gets data from memory location 2000H
   CL ←[2000H]
   MOV [3000H], DL ; Memory location 3000H gets data from DL Register
   [3000H] ← DL
4) **INDIRECT ADDRESSING MODE:** In Indirect Addressing modes, address is given by a register. The register can be incremented in a loop to access a series of locations. There are various sub-types of Indirect addressing mode.
   REGISTER INDIRECT ADDRESSING MODE
   This is the most basic form of indirect addressing mode.Here address is simply given by a register.
   Eg: MOV CL, [BX] ; CL gets data from a memory location pointed by BX
   CL ← [BX]. If BX = 2000H, CL ← [2000H]
   Eg: MOV [BX], CL ; CL is stored at a memory location pointed by BX
   [BX] ←CL. If BX = 2000H, [2000H] ← CL.

REGISTER RELATIVE ADDRESSING MODE : Here address is given by a register plus a numeric displacement.

Eg: MOV CL, [BX + 03H] ; CL gets data from a location BX + 03H

CL ← [BX+03H]. If BX = 2000H, then CL ← [2003H]

Eg: MOV [BX + 03H], CL ; CL is stored at location BX + 03H

[BX+03H] ←CL. If BX = 2000H, then [2003H] ← CL.

BASE INDEXED ADDRESSING MODE Here address is given by a sum of two registers. This is typically useful in accessing an array or a look up table. One register acts as the base of the array holding its starting address and the other acts as an index indicating theelement to be accessed.

Eg: MOV CL, [BX + SI] ; CL gets data from a location BX + SI ; CL ← [BX+SI]. ;

If BX = 2000H, SI = 1000H, then CL ç [3000H] Eg: MOV [BX + SI], CL ; CL is stored at location BX + SI ; [BX+SI] ← CL. ; If BX = 2000H, SI = 1000H, then [3000H] ← CL.

BASE RELATIVE PLUS INDEX ADDRESSING MODE Here address is given by a sum of base register plus index register plus a numeric displacement.

Eg: MOV CL, [BX+SI+03H] ; CL gets data from a location BX + SI + 03H ;

CL ←[BX+SI+03H]. ;

If BX = 2000H, SI = 1000H, then CL ← [3003H]

Eg: MOV [BX+SI+03H], CL ; CL is stored at location BX + SI + 03H ;

[BX+SI+03H] ←CL. ;

If BX = 2000H, SI = 1000H, then [3003H] ← CL.

IMPLIED ADDRESSING MODE: In this addressing mode, the operand is not specified at all, as it is an implied operand. Someinstructions operate only on a particular register. In such cases, specifying the register becomesunnecessary as it becomes implied.

Eg: STC ; Sets the Carry flag.; This instruction can only operate on the Carry Flag.

Eg: CMC ; Complements the Carry flag.; This instruction can only operate on the Carry Flag

Instruction Set:

Most computer instructions can be classified into three categories:

1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

1) Data Transfer Instructions:Data transfer instructions move data from one place in the computer to another without changing the data content. The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.Table below gives a list of eight data transfer instructions used in many computers.

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

Accompanying each instruction is a mnemonic symbol. Different computers use different mnemonics for the same instruction name.

The **load instruction** has been used mostly to designate a transfer from memory to a processor register, usually an accumulator. The **store instruction** designates a transfer from a processor register into memory. The **move instruction** has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words. The **exchange instruction** swaps information between two registers or a register and a memory word. The **input and output instructions** transfer data among processor registers and input or output terminals. The **push and pop** instructions transfer data between processor registers and a memory stack.

2) Data Manipulation Instructions: The data manipulation instructions in a typical computer are usually divided into three basic types:

- ✓ Arithmetic instructions
- ✓ Logical and bit manipulation instructions

- ✓ Shift instructions

Arithmetic instructions :The four basic arithmetic operations are addition, subtraction, multiplication,and division. Most computers provide instructions for all four operations.Some small computers have only addition and possibly subtraction instructions.

A list of typical arithmetic instructions is given in Table given below:

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

The **increment instruction** adds 1 to the value stored in a register or memory word.The **decrement instruction** subtracts 1 from a value stored in a registeror memory word.The add, subtract, multiply, and divide instructions may be available fordifferent types of data. The data type assumed ·to be in processor registersduring the execution of these arithmetic operations is included in the definitionof the operation code. An arithmetic instruction may specify fixed-point orfloating-point data, binary or decimal data, single-precision or double-precision data.

The mnemonics for three add instructions that specifydifferent data types are shown below:

ADDI Add two binary integer numbers

ADDF Add two floating-point numbers

ADDD Add two decimal numbers in BCD

The instruction "**add with carry**" performs the addition on two operands plus the value of the carry fromthe previous computation. Similarly, the "**subtract with borrow**" instructionsubtracts two words and a borrow which may have resulted from a previoussubtract operation. The **negate instruction** forms the 2' s complement of anumber, effectively reversing the sign of an integer when represented in thesigned-2's complement form.

Logical and Bit Manipulation Instructions: Logical instructions **perform binary operations on strings of bits** stored in registers. They are useful for manipulating individual bits or a group of bits that represent binary-coded information. The **logical instructions consider each bit of the operand separately** and treat it as a Boolean variable. By proper application of the logical instructions, it is possible to change bit values, to clear a group of bits, or to insert new bit values into operands stored in registers or memory words.

Some logical and bit manipulation instructions are shown in the figure below:

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

The clear instruction causes the specified operand to be replaced by D's.The complement instruction produces the 1's complement by inverting all thebits of the operand. The AND, OR, and XOR instructions produce the corresponding logical operations on individual bits of the operands. Although theyperform Boolean operations, when used in computer instructions, the logicalinstructions should be considered as performing bit manipulation operations.There are three-bit manipulation operations possible: a selected bit can becleared to 0, or can be set to 1, or can be complemented. The three logicalinstructions are usually applied to do just that.

Shift Instructions: Shifts are operations in which the bits of a word are moved to the left or right. Shift instructions may specify either logicalshifts, arithmetic shifts, or rotate-type operations. In either case the shift maybe to the right or to the left.Table below lists four types of 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

The **logical shift** inserts 0to the end bit position. The end position is the leftmost bit for shift right and the rightmost bit position for the shift left.

The **arithmetic shift-right** instruction must preserve the sign bit in the leftmost position. The sign bit is shifted to the right together with the rest of the number, but the sign bit itself remains unchanged. This is a shift-right operation with the end bit remaining the same. The arithmetic shift-left instruction inserts 0 to the end position and is identical to the logical shift-left instruction.

The rotate instructions produce a circular shift. Bits shifted out at one end of the word are not lost as in a logical shift but are circulated back into the other end. The rotate through carry instruction treats a carry bit as an extension of the register whose word is being rotated. Thus, a rotate-left through carry instruction transfers the carry bit into the rightmost bit position of the register, transfers the leftmost bit position into the carry, and at the same time, shifts the entire register to the left.

Program Control Instructions: Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer a program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered. In other words, program control instructions specify conditions for altering the content of the program counter.

Some program control instructions are listed in Table below:

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

Branch and jump instructions are used interchangeably to mean the same thing, but sometimes they are used to denote different addressing modes. Branch instruction is written as **BR ADR**, where ADR is a symbolic name for an address. Branch and jump instructions may be conditional or unconditional. Anunconditional branch instruction causes a branch to the specified address without any conditions. The conditional branch instruction specifies a conditionsuch as branch if positive or branch if zero. If the condition is met, the programcounter is loaded with the branch address and the next instruction is taken.from this address. If the condition is not met, the program counter is not changed and the next instruction is taken from the next location in sequence.

The **skip instruction** does not need an address field and is therefore azero-address instruction. A conditional skip instruction will skip the nextinstruction if the condition is met. If the condition is not met, control proceeds with thenext instruction in sequence.

The **call and return** instructions are used in conjunction with subroutines.

The **compare instruction** performs a subtraction between two operands, but the result of the operation isnot retained. However, certain status bit conditions are set as a result of theoperation. Similarly, the **test instruction** performs the logical AND of twooperands and updates certain status bits without retaining the result or changing the operands. (***Note:***The compare and test instructions do not change the program sequence directly. They are listed in Table because of their application in setting conditions for subsequent conditional branch instructions)

RISC vs CISC Architecture

RISC stands for Reduced Instruction set Computer and **CISC** stands for Complex Instruction Set Computer.RISC and CISC are the two ideologies behind making the processor.

	RISC	CISC
1	**Instructions of a fixed size**	Instructions of **variable size**
2	Most instructions take **same time to fetch**.	Instructions have **different fetching times**.
3	Instruction set **simple and small**.	Instruction set **large and complex**.
4	**Less addressing modes** as most operations are register based.	**Complex addressing modes** as most operations are memory based.
5	**Compiler design is simple**	**Compiler design is complex**
6	**Total size of program is large** as many instructions are required to perform a task as instructions are simple.	**Total size of program is small** as few instructions are required to perform a task as instructions are complex & more powerful.
7	Instructions use a **fixed number of operands.**	Instructions have **variable** number of operands.
8	Ideal for processors performing a **dedicated operation**.	Ideal for processors performing a **verity of operations**.
9	Since instructions are simple, they can be decoded by a **hardwired control unit**.	Since instructions are complex, they require a **Micro-programmed Control Unit**.
10	Execution speed is **faster** as most operations are register based.	Execution speed is **slower** as most operations are memory based.
11	As No. of cycles per instruction is fixed, it gives a **better degree of pipelining**	Since number of cycles per instruction varies, **pipelining has more bubbles** or stalls.
12	E.g.:: **ARM7**, **PIC 18** Microcontrollers.	E.g.: Intel **8085, 8086** Microprocessors.
13	They have **register based operations**.	13. **Memory based** operations.

CPU CONTROL UNIT DESIGN

- **Hardwired CU :**
  In Hardwired CU, control signals are produced by hardware. There are three types of Hardwired Control Units
  1) **STATE TABLE METHOD**
  2) **DELAY ELEMENT METHOD**
  3) **SEQUENCE COUNTER METHOD**

STATE TABLE METHOD:

1) It is the most basic type of hardwired control unit.
2) Here the behaviour of the control unit is represented in the form of a table called the state table.
3) The rows represent the T-states and the columns indicate the instructions.
4) Each intersection indicates the control signal to be produced, in the corresponding T-state of everyinstruction.
4) A circuit is then constructed based on every column of this table, for each instruction.

T-STATES	INSTRUCTIONS			
	I_1	I_2	...	I_N
T_1	$Z_{1,1}$	$Z_{1,2}$	...	$Z_{1,N}$
T_2	$Z_{2,1}$	$Z_{2,2}$	...	$Z_{2,N}$
...	...	...	...	...
T_M	$Z_{M,1}$	$Z_{M,2}$	...	$Z_{M,N}$

$Z_{1,1}$: Control Signal to be produced in T-state (T_1) of Instruction (I_1)

ADVANTAGE: It is the simplest method and is ideally suited for very small instruction sets.

DRAWBACK: As the number of instructions increase, the circuit becomes bigger and hence more complicated. As a tabular approach is used, instead of a logical approach (flowchart), there are duplications of manycircuit elements in various instructions.

Delay Element Method:

1) Here the behaviour of the control unit is represented in the form of a flowchart.

2) Each step in the flowchart represents a control signal to be produced.

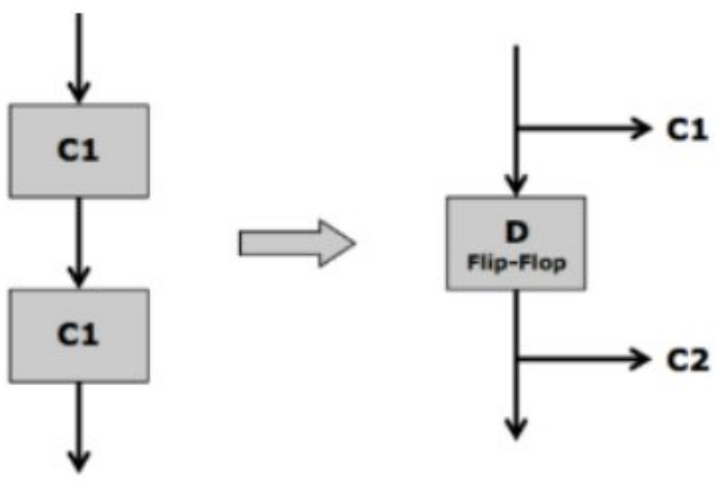
3) Once all steps of a particular instruction, are performed, the complete instruction gets executed.

4) Control signals perform Micro-Operations, which require one T-states each.

5) Hence between every two steps of the flowchart, there must be a delay element.

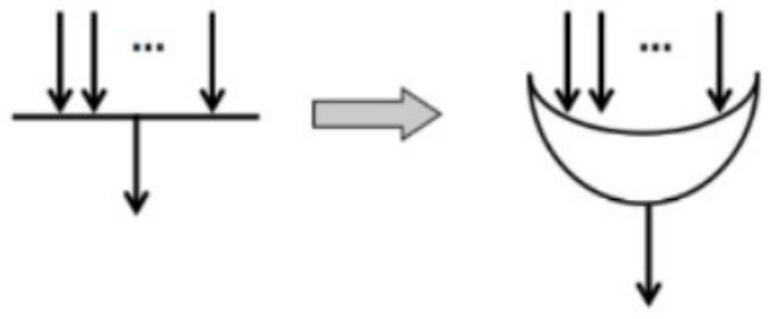
6) The delay must be exactly of one T-state. This delay is achieved by D Flip-Flops.

7) These D Flip-Flops are inserted between every two consecutive control signals.

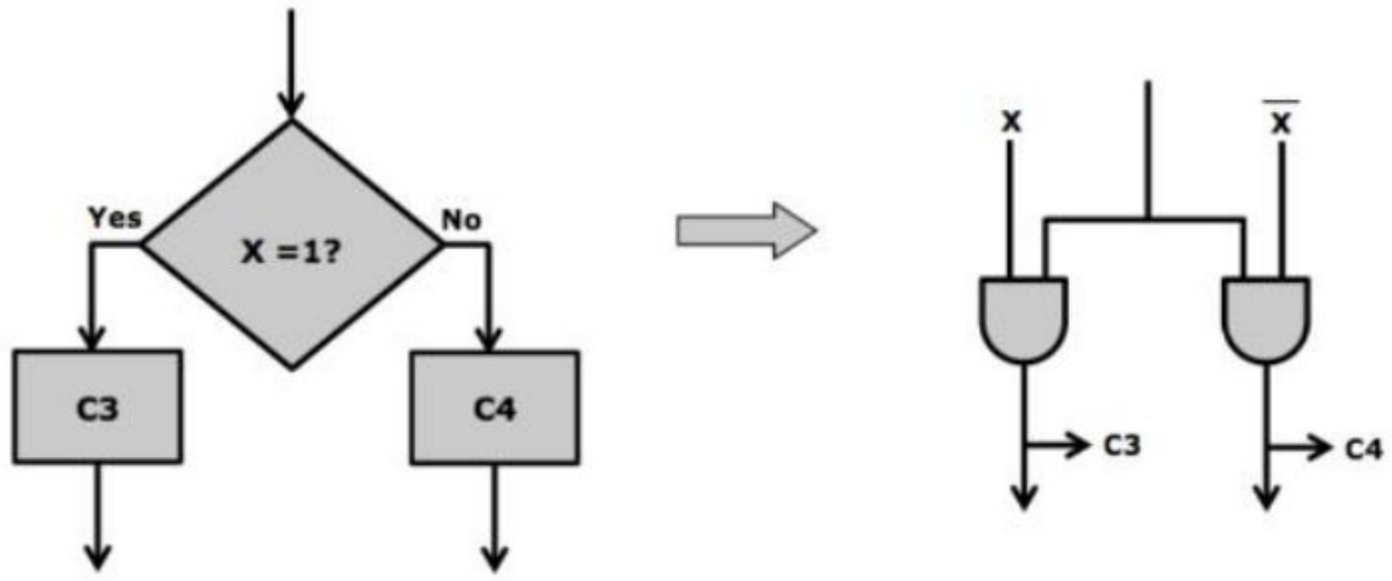


8) Of all D Flip-Flops only one will be active at a time. So the method is also called "One Hot Method".

9) In a multiple entry point, to combine two or more paths, we use an OR gate.



10)A decision box is replaced by a set of two complementing AND gates



11) A multiple entry point is substituted by an OR gate.

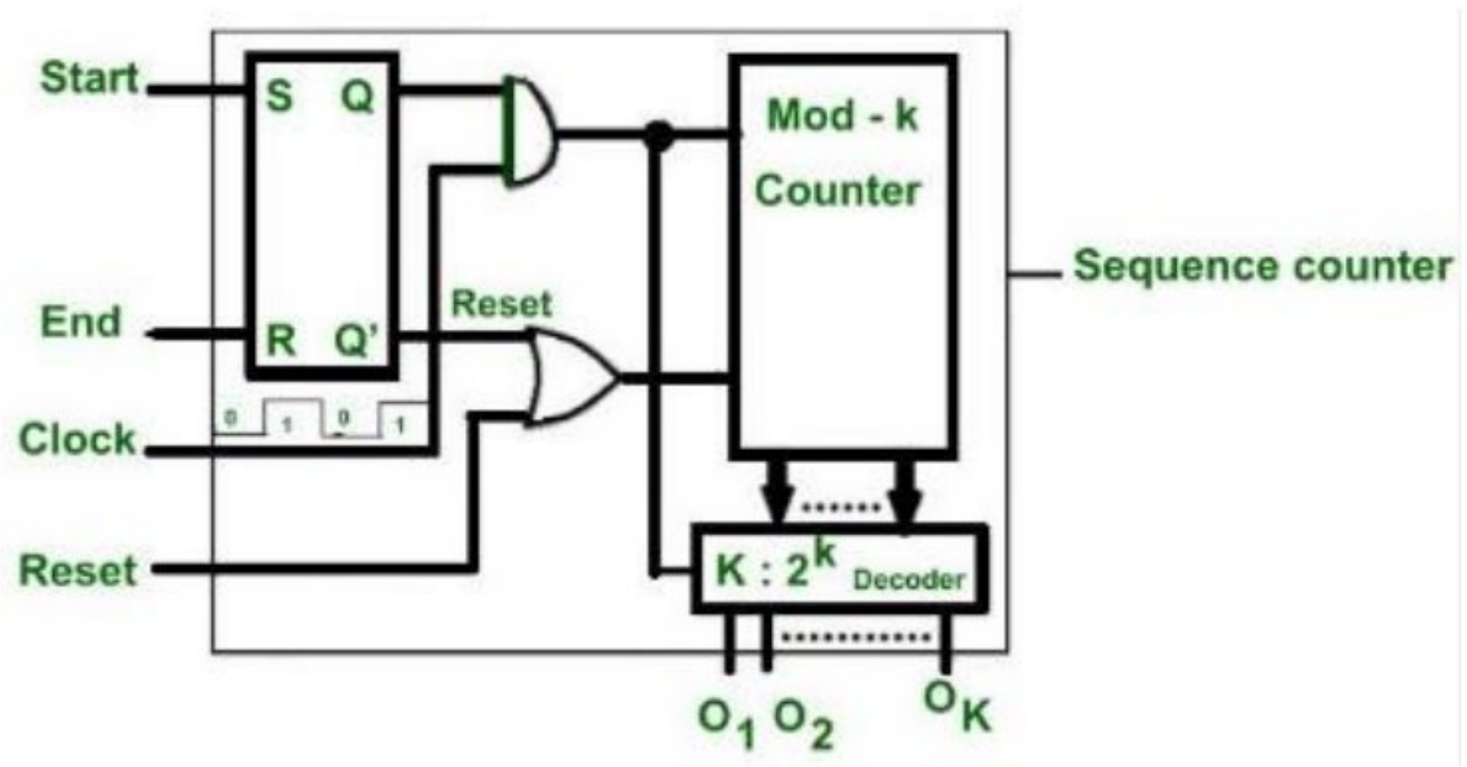
ADVANTAGE:

As the method has a logical approach, it can reduce the circuit complexity.This is done by re-utilizing common elements between various instructions.

DRAWBACK:

As the no of instructions increase, the number of D Flip-Flops increase, so the cost increases.Moreover, only one of those D Flip-Flops are actually active at a time.

SEQUENCE COUNTER METHOD:



1) This is the most popular form of hardwired control unit. The goal of this circuit is to provide triggers to different parts of the circuit after gaps of 1-Tstate.

2) It follows the same logical approach of a flowchart, like the Delay element method, but does not use all those unnecessary D Flip-Flops because at any point of time only one delay element is active and a complex circuitry would involve many delay elements which is very inefficient. The D-Flip-flops are replaced by trigger points which are activated after gaps of one T-state.

Following are the steps involved in designing a CU using Sequence Counter Method.

1) First a flowchart is made representing the behaviour of a control unit.

2) It is then converted into a circuit using the same principle of AND & OR gates.

3) We need a delay of 1 T-state (one clock cycle) between every two consecutive control signals.

4) That is achieved by the above circuit.

5) If there are “k” number of distinct steps producing control signals, we employ a “mod k” and “k”output decoder.

6) The counter will start counting at the beginning of the instruction.

7) The “clock” input via an AND gate ensures each count will be generated after 1 T-state.

8)The count is given to the decoder which triggers the generation of “k” control signals, each aftera delay of 1 T-state.

9)When the instruction ends, the counter is reset so that next time, it begins from the first count.

ADVANTAGE:

Avoids the use of too many D Flip-Flops.

GENERAL DRAWBACKS OF A HARDWIRED CONTROL UNIT

1) Since they are based on hardware, as the instruction set increases, the circuit becomes more and more complex. For modern processors having hundreds of instructions, it is virtually impossible to create Hardwired Control Units.

2) Such large circuits are very difficult to debug.

3) As the processor gets upgraded, the entire Control Unit has to be redesigned, due to the rigid nature of hardware design.

Microprogrammed CU

WILKES’ DESIGN FOR A MICROPROGRAMMED CONTROL UNIT:

1) Microprogrammed Control Unit produces control signals by software, using micro-instructions

2) A program is a set of instructions.

3) An instruction requires a set of Micro-Operations.

4) Micro-Operations are performed by control signals.

5) Instead of generating these control signals by hardware, we use micro-instructions. This means every instruction requires a set of micro-instructions This is called its micro-program.

6) Microprograms for all instructions are stored in a small memory called “Control Memory”. The Control memory is present inside the processor.

7) Consider an Instruction that is fetched from the main memory into the Instruction Register (IR).

8) The processor uses its unique “opcode” to identify the address of the first micro-instruction. That address is loaded into CMAR (Control Memory Address Register). CMAR passes the address to the decoder.

9) The decoder identifies the corresponding micro-instruction from the Control Memory.

10) A micro-instruction has two fields: a control filed and an address field.

Control field: Indicates the control signals to be generated.

Address field: Indicates the address of the next micro-instruction.

11) This address is further loaded into CMAR to fetch the next micro-instruction.

12) For a conditional micro-instruction, there are two address fields.This is because, the address of the next micro-instruction depends on the condition.The condition (true or false) is decided by the appropriate control flag.

13) The control memory is usually implemented using FLASH ROM as it is writable yet non-volatile.

ADVANTAGES

1) The biggest advantage is flexibility.

2) Any change in the control unit can be performed by simply changing the micro-instruction.

3) This makes modifications and up gradation of the Control Unit very easy.

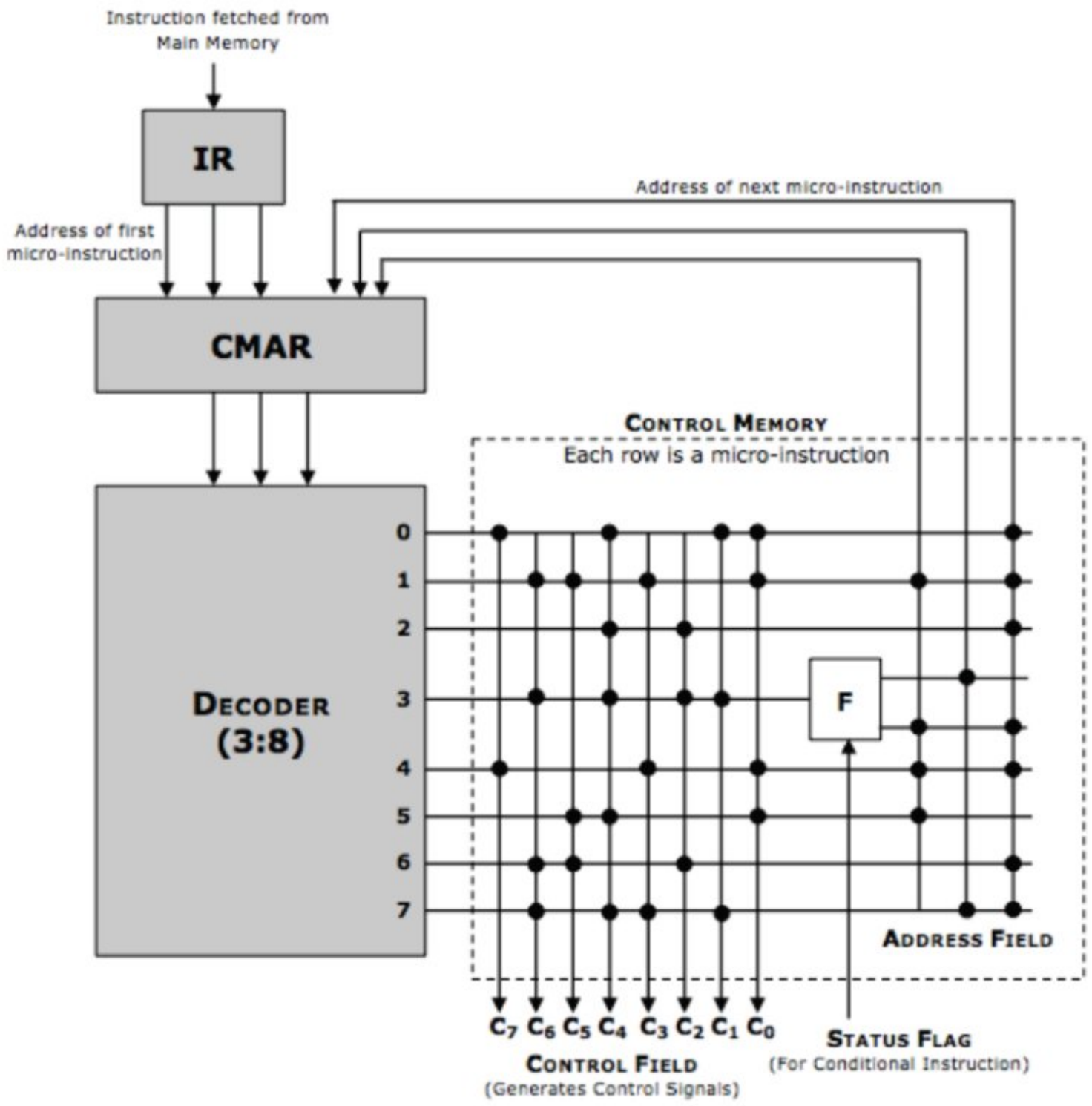
4) Moreover, software can be much easily debugged as compared to a large Hardwired Control Unit.

DRAWBACKS

1) Control memory has to be present inside the processor, increasing its size.

2) This also increases the cost of the processor.

3) The address field in every micro-instruction adds more space to the control memory. This can beeasily avoided by proper micro-instruction sequencing.



TYPICAL MICROPROGRAMMED CONTROL UNIT

1) Microprogrammed Control Unit produces control signals by software, using micro-instructions.

2) A program is a set of instructions.

3) An instruction requires a set of Micro-Operations.

4) Micro-Operations are performed by control signals.

5) Here, these control signals are generated using micro-instructions.

6) This means every instruction requires a set of micro-instructions

7) This is called its micro-program.

8) Microprograms for all instructions are stored in a small memory called "Control Memory".

9) The Control memory is present inside the processor.

10) Consider an Instruction that is fetched from the main memory into the Instruction Register (IR).

11) The processor uses its unique "opcode" to identify the address of the first micro-instruction.

12) That address is loaded into CMAR (Control Memory Address Register) also called μIR.

13) This address is decoded to identify the corresponding μ-instruction from the Control Memory.

14) There is a big improvement over Wilkes' design, to reduce the size of micro-instructions.

15) Most micro-instructions will only have a Control field.

16) The Control field Indicates the control signals to be generated.

17) Most micro-instructions will not have an address field.

18) Instead, μPC will simply get incremented after every micro-instruction.

19) This is as long as the μ-program is executed sequentially.

20) If there is a branch μ-instruction only then there will be an address filed.

21) If the branch is unconditional, the branch address will be directly loaded into CMAR.

22) For Conditional branches, the Branch condition will check the appropriate flag.

23) This is done using a MUX which has all flag inputs.

24) If the condition is true, then the MUX will inform CMAR to load the branch address.

25) If the condition is false CMAR will simply get incremented.

26) The control memory is usually implemented using FLASH ROM as it is writable yet non-volatile.

ADVANTAGES

1) The biggest advantage is flexibility.

2) Any change in the control unit can be performed by simply changing the micro-instruction.

3) This makes modifications and up gradation of the Control Unit very easy.

4) Moreover, software can be much easily debugged as compared to a large Hardwired Control Unit.

5) Since most micro-instructions are executed sequentially, they don't need for an address field.

6) This significantly reduces the size of micro-instructions, and hence the Control Memory.

DRAWBACKS

1) Control memory has to be present inside the processor, increasing its size.

2) This also increases the cost of the processor.