

Unit 3 Notes

Registers of basic computer

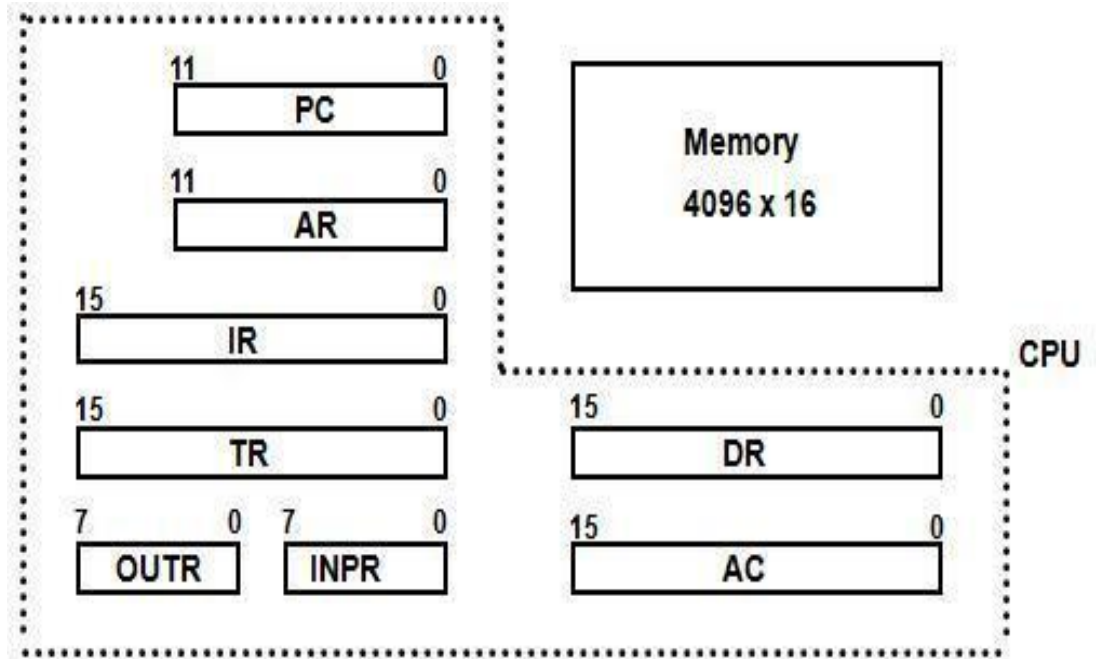


Figure 2.4: Basic Computer Register and Memory

- The data register (DR) holds the operand read from memory.
- The accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the instruction register (IR).
- The temporary register (TR) is used for holding temporary data during the processing.
- The memory address register (AR) has 12 bits.
- The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed.
- Instruction words are read and executed in sequence unless a branch instruction is encountered. A branch instruction calls for a transfer to a nonconsecutive instruction in the program.
- Two registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

Register Symbol	Bits	Register Name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Table : List of Registers for Basic Computer

Instruction Format with its types.

- The basic computer has three instruction code formats, as shown in figure 2.6.

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



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



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

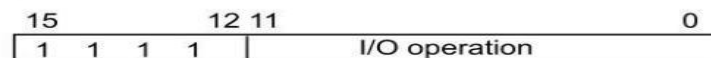


Figure 2.6: Basic computer instruction format

- Each format has 16 bits.
- The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.
- A **memory-reference instruction** uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.
- The **register reference instructions** are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
- An **input-output instruction** does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

Instruction cycle

- A program residing in the memory unit of the computer consists of a sequence of instructions. 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.
- After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction. □
- This process continues unless a HALT instruction is encountered.

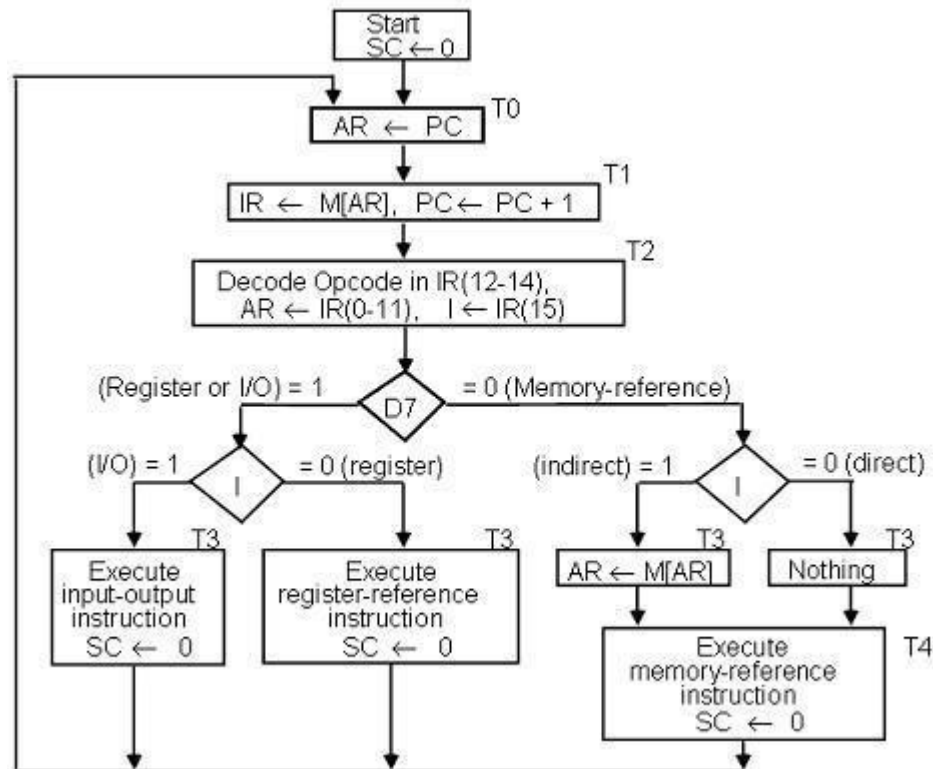


Figure : Flowchart for instruction cycle (initial configuration)

- The flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.
- If $D7 = 1$, the instruction must be register-reference or input-output type. If $D7 = 0$, the operation code must be one of the other seven values 110, specifying a memory-reference instruction. Control then inspects the value of the first bit of the instruction, which now available in flip-flop I.
- If $D7 = 0$ and $I = 1$, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory.
- The three instruction types are subdivided into four separate paths. The selected

operation is activated with the clock transition associated with timing signal T₃. This can be symbolized as follows:

$$D'7 \mid T_3: AR \xleftarrow{M[AR]} D'7 \mid T_3: \text{Nothing}$$

$$D7 \mid T_3: \text{Execute a register-reference instruction}$$

$$D7 \mid T_3: \text{Execute an input-output instruction}$$

- When a memory-reference instruction with $I = 0$ is encountered, it is not necessary to do anything since the effective address is already in AR.
- However, the sequence counter SC must be incremented when $D'7 \mid T_3 = 1$, so that the execution of the memory-reference instruction can be continued with timing variable T₄.
- A register-reference or input-output instruction can be executed with the clock associated with timing signal T₃. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with T₀ = 1. SC is either incremented or cleared to 0 with every positive clock transition.

Interrupt Cycle

The way that the interrupt is handled by the computer can be explained by means of the flowchart shown in figure below.

- An interrupt flip-flop R is included in the computer.
- When $R = 0$, the computer goes through an instruction cycle.
- During the execute phase of the instruction cycle IEN is checked by the control. □
- If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits.
- If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information.
- In this case, control continues with the next instruction cycle. If either flag is set to 1 while $IEN = 1$, flip-flop R is set to 1.
- At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

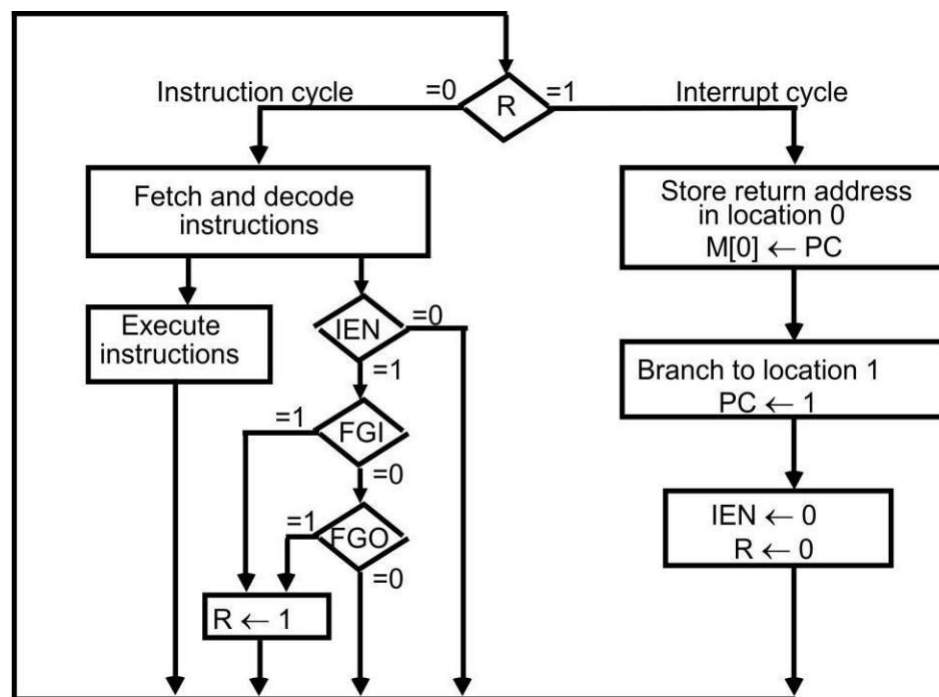


Figure : Flowchart for interrupt cycle

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register, a memory stack, or a specific memory location.
- Here we choose the memory location at address 0 as the place for storing the return

address.

- Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.
- This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted.
- The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruction is placed at the end of the I/O service program.

Register transfer statements for the interrupt cycle

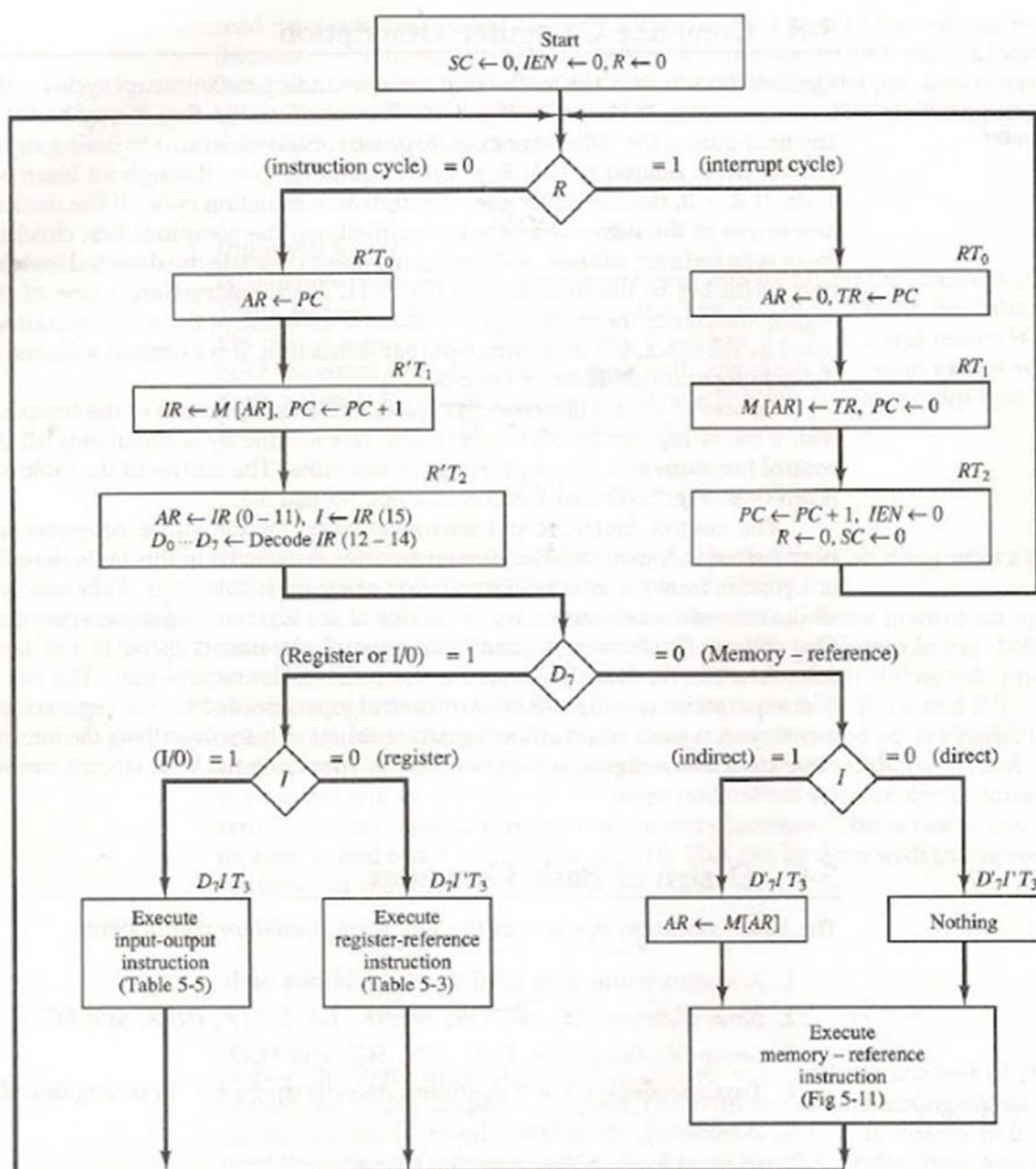
- The flip-flop is set to 1 if $IEN = 1$ and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T_0 , T_1 or T_2 are active.
- The condition for setting flip-flop $R = 1$ can be expressed with the following register transfer statement:

$$T_0'T_1'T_2' (IEN) (FGI + FGO): R \leftarrow 1$$

- The symbol + between FGI and FGO in the control function designates a logic OR operation. This is AND with IEN and $T_0'T_1'T_2'$.
- The fetch and decode phases of the instruction cycle must be modified and Replace T_0 , T_1 , T_2 with $R'T_0$, $R'T_1$, $R'T_2$
- Therefore the interrupt cycle statements are :
 - $RT_0: AR \leftarrow 0, TR \leftarrow PC$
 - $RT_1: M[AR] \leftarrow TR, PC \leftarrow 0$
 - $RT_2: PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
- During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR.
- With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0.
- The third timing signal increments PC to 1, clears IEN and R, and control goes back to T_0 by clearing SC to 0.
- The beginning of the next instruction cycle has the condition RT_0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes.

Flow chart for Complete Instruction cycle.

- The final flowchart of the complete instruction cycle, including the interrupt cycle for the basic computer, is shown in Figure below.
- The interrupt flip-flop R may be set at any time during the indirect or execute phases.
- The control returns to timing signal T_0 after SC is cleared to 0.
- If $R = 1$, the computer goes through an interrupt cycle. If $R = 0$, the computer goes through an instruction cycle.
- If the instruction is one of the memory-reference instructions, the computer first checks if there is an indirect address and then continues to execute the decoded instruction according to the flowchart.
- If the instruction is one of the register-reference instructions, it is executed with one of the microoperations register reference.
- If it is an input-output instruction, it is executed with one of the microoperation's input-output reference.



CISC	RISC
The original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
Instructions can take several clock cycles	Single-cycle instructions
Hardware-centric design – the ISA does as much as possible using hardware circuitry	Software-centric design – High-level compilers take on most of the burden of coding many software steps from the programmer
More efficient use of RAM than RISC	Heavy use of RAM (can cause bottlenecks if RAM is limited)
Complex and variable length instructions	Simple, standardized instructions
May support microcode (micro-programming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

Microprogrammed Control Unit

- **Control Memory**
- **Sequencing Microinstructions**
- **Microprogram Example**
- **Design of Control Unit**
- **Microinstruction Format**

Control Unit

- **Initiate sequences of microoperations**

- ❑ **Control signal** (*that specify microoperations*) in a bus-organized system
groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units

- **Two major types of Control Unit**

- » **Hardwired Control :**

- ❖ The control logic is implemented with gates, F/Fs, decoders, and other digital circuits
 - ❖ + Fast operation.
 - ❖ -Wiring change (if the design has to be modified)

- » **Microprogrammed Control :** *in this Chapter*

- The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
 - + Any required change can be done by updating the microprogram in control memory.
 - - Slow operation

Control Word

The control variables at any given time can be represented by a string of 1's and 0's.

Microprogrammed Control Unit

A control unit whose binary control variables are stored in memory (*control memory*).

Microinstruction

The microinstruction specifies one or more microoperations

Microprogram

A sequence of microinstruction

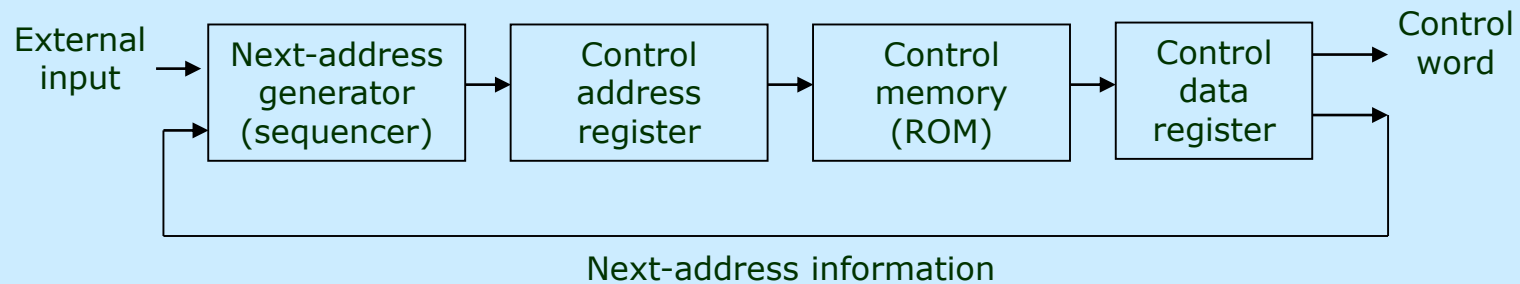
» **Dynamic microprogramming : *Control Memory* = RAM**

- RAM can be used for writing (*to change a writable control memory*)
- Microprogram is loaded initially from an auxiliary memory such as a magnetic disk

» **Static microprogramming : *Control Memory* = ROM**

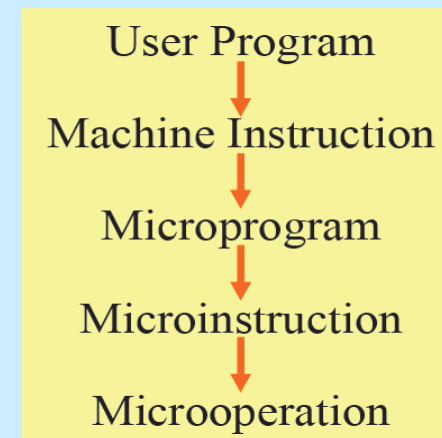
- Control words in ROM are made permanent during the hardware production.

Microprogrammed Control Organization



Control Memory

- »A memory is part of a control unit :
- »Computer Memory (*employs a microprogrammed control unit*)
 - Main Memory : for storing user program (*Machine instruction/data*)
 - Control Memory : for storing microprogram (*Microinstruction*)



Control Address Register

»Specify the address of the microinstruction

Micro program Sequencer

- »Determine the address sequence that is read from control memory
- »Next address of the next microinstruction can be specified several way depending on the sequencer input.

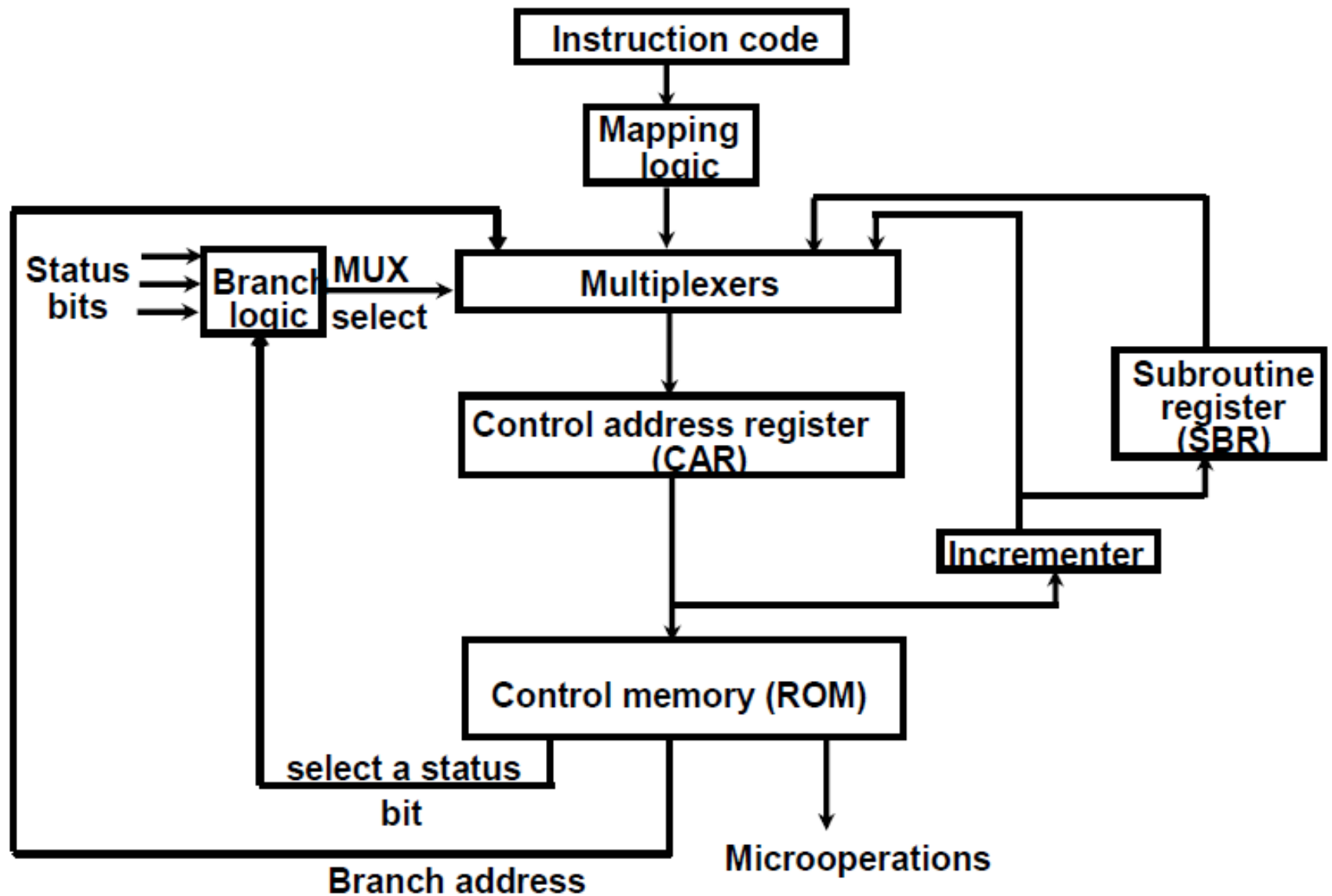
Sequencing Capabilities Required in a Control Storage

1. Incrementing of the control address register
2. Unconditional and conditional branches
3. A mapping process from the bits of the machine instruction to an address for control memory
4. A facility for subroutine call and return

Control data register

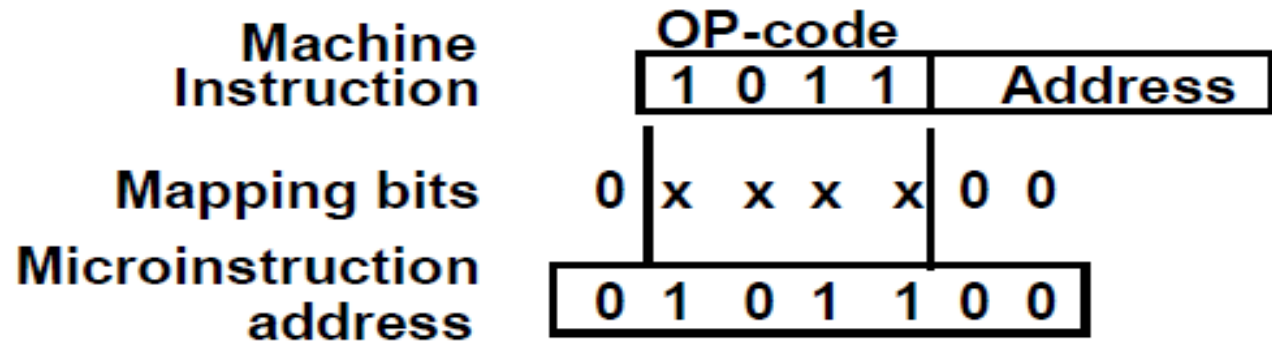
- »Hold the microinstruction read from control memory
- »Allows the execution of the microoperations specified by the control word ***simultaneously*** with the generation of the next microinstruction

Microprogram sequencer

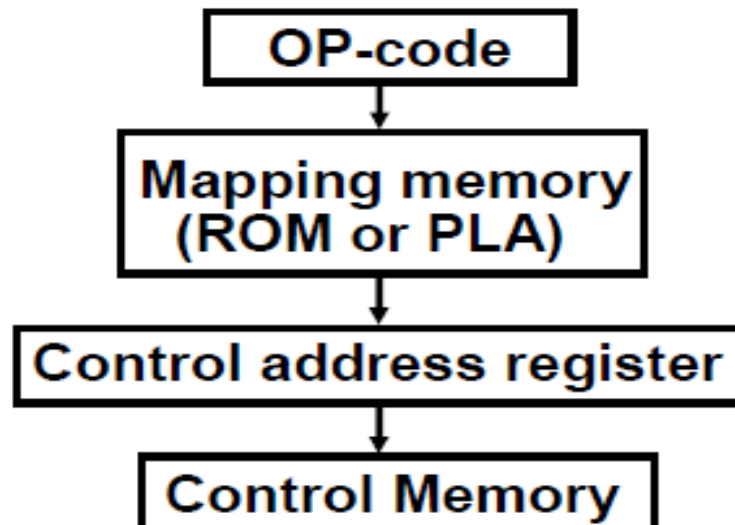


Mapping of Instructions to Microroutines

Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram



Mapping function implemented by ROM or PLA



Machine Instruction Format

Machine instruction format

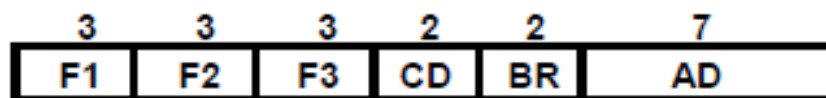


Sample machine instructions

Symbol	OP-code	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
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

Microinstruction Format



F1, F2, F3: Microoperation fields

CD: Condition for branching

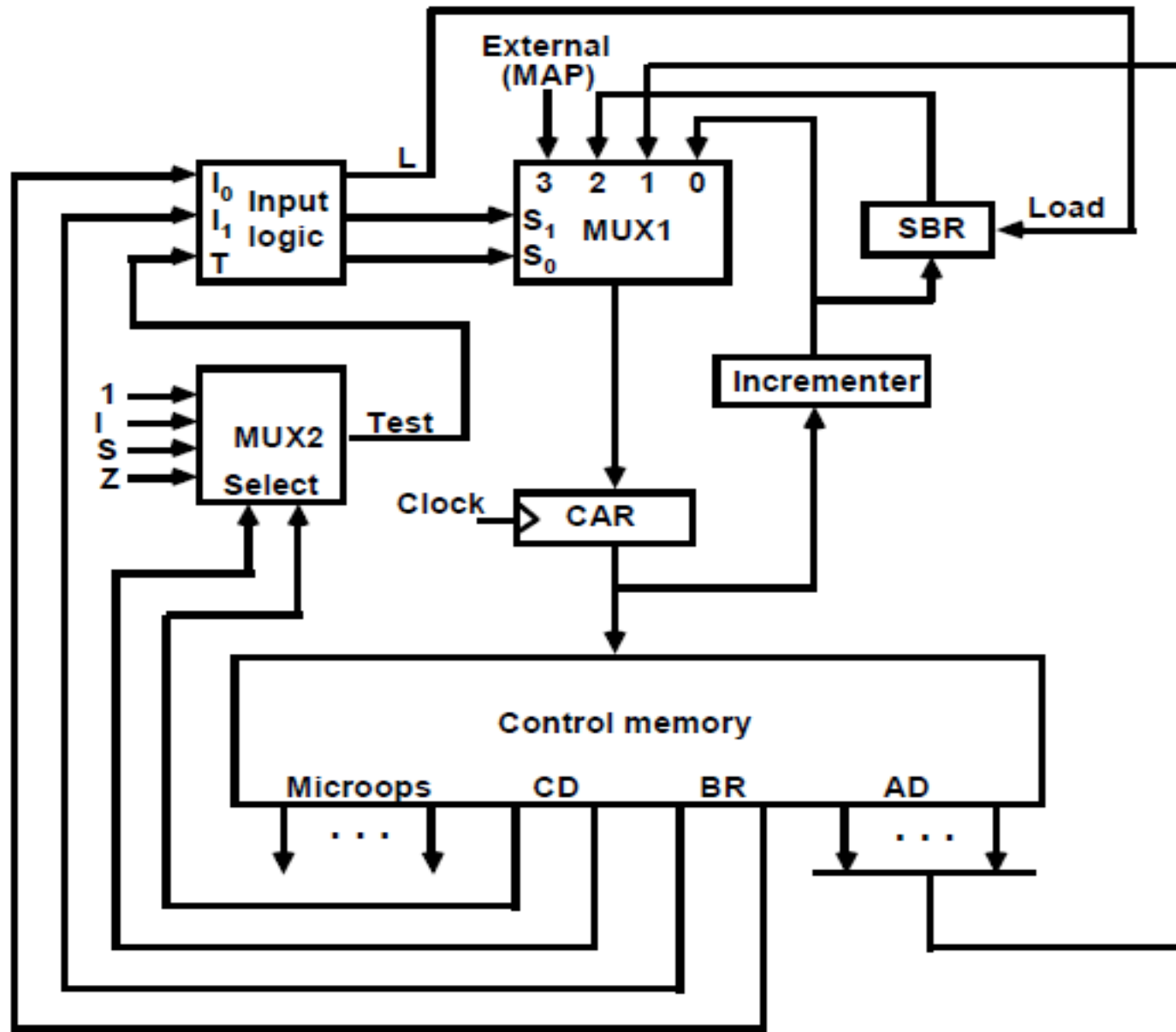
BR: Branch field

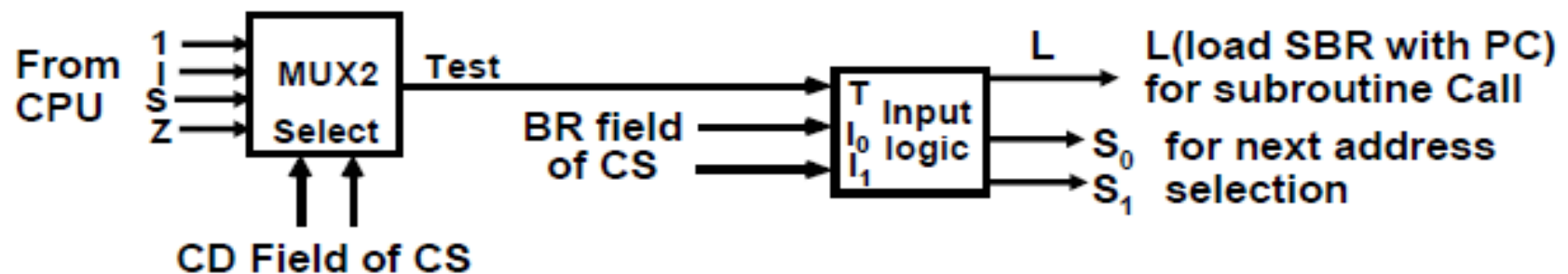
AD: Address field

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	$CAR(2-5) \leftarrow DR(11-14), CAR(0,1,6) \leftarrow 0$

Microprogram Sequencer





Input Logic

I ₀ I ₁ T	Meaning	Source of Address	S ₁ S ₀	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$$\begin{aligned}
 S_0 &= I_0 \\
 S_1 &= I_0 I_1 + I_0' T \\
 L &= I_0' I_1 T
 \end{aligned}$$

HORIZONTAL AND VERTICAL MICROINSTRUCTION FORMAT

Horizontal Microinstructions

Each bit directly controls each micro-operation or each control point

Horizontal implies a long microinstruction word

Advantages: Can control a variety of components operating in parallel.

--> Advantage of efficient hardware utilization

Disadvantages: Control word bits are not fully utilized

--> CS becomes large --> Costly

Vertical Microinstructions

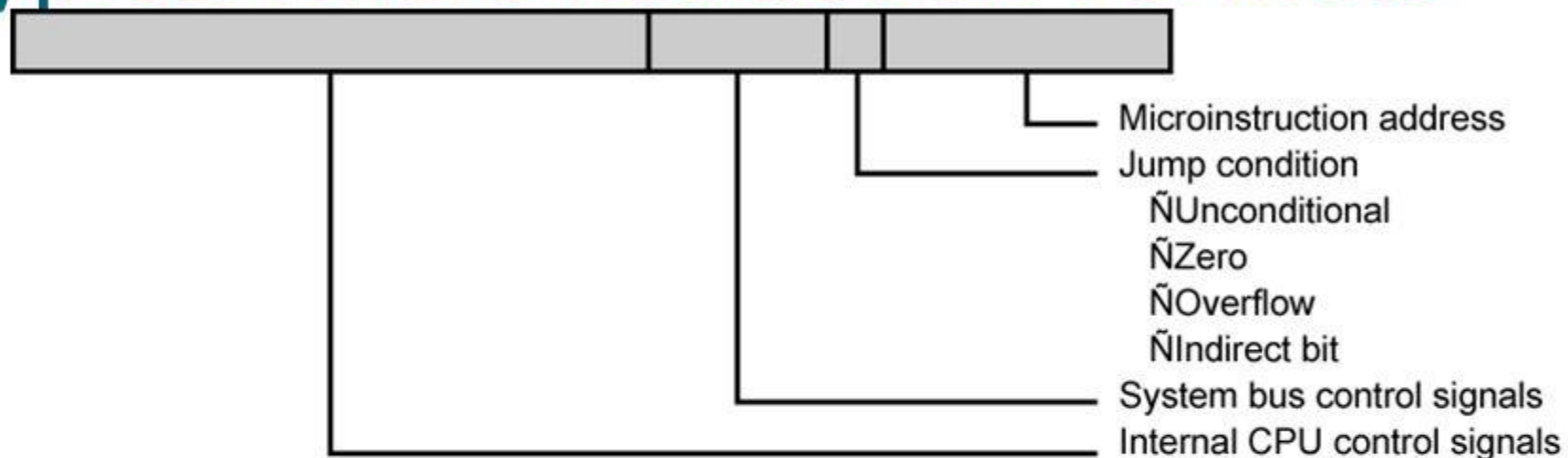
A microinstruction format that is not horizontal

Vertical implies a short microinstruction word

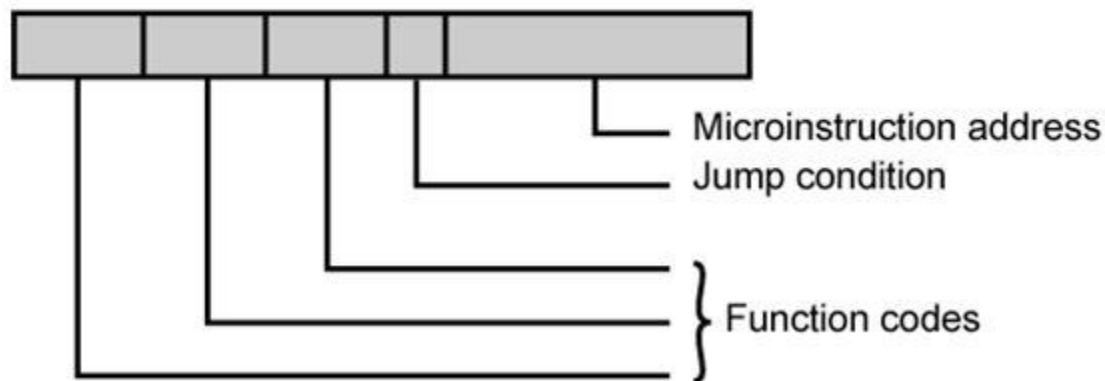
Encoded Microinstruction fields

--> Needs decoding circuits for one or two levels of decoding

Typical Microinstruction Formats



(a) Horizontal microinstruction



(b) Vertical microinstruction

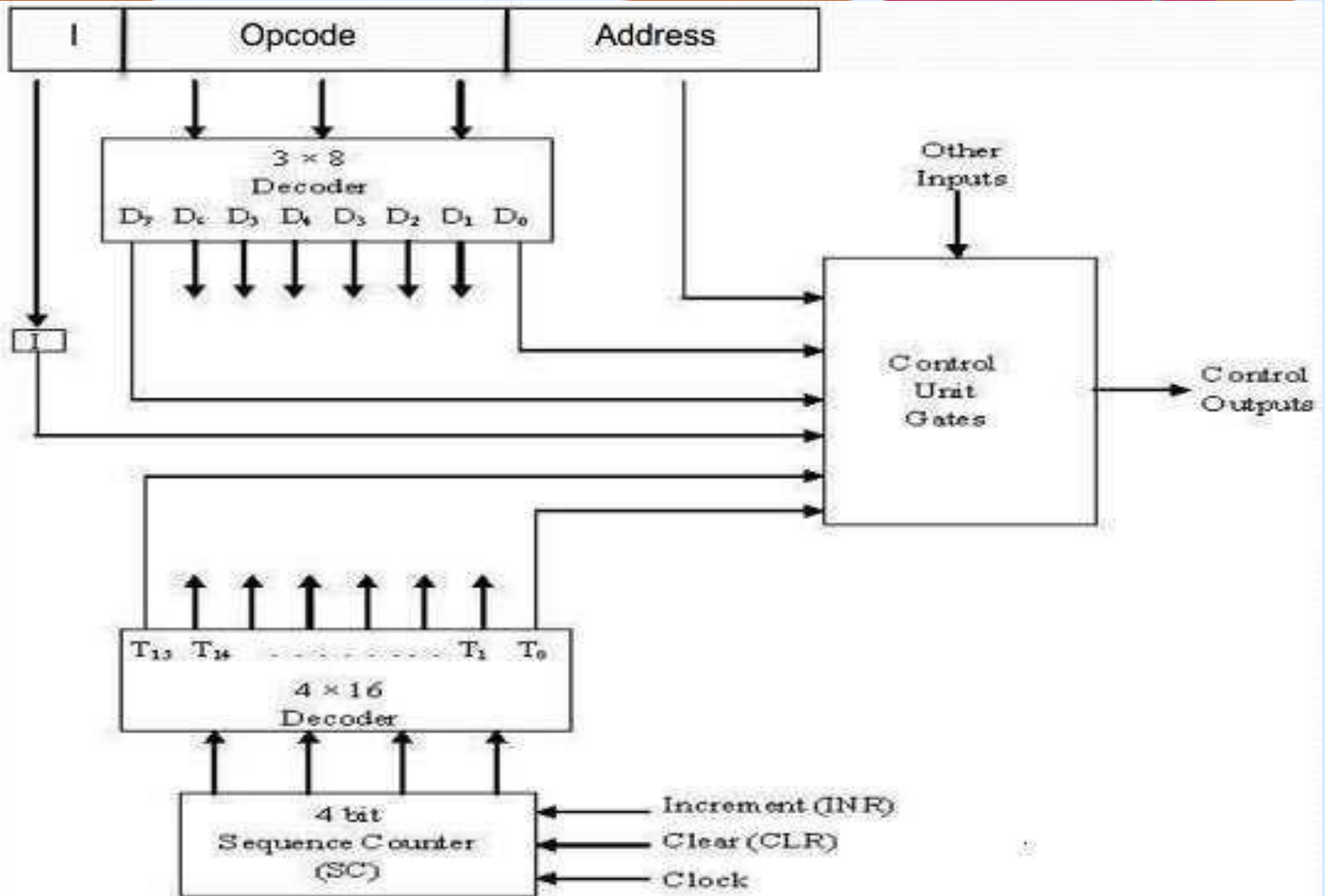
Hardwired Control Unit

- ❖ Hardwired control units are implemented through use of sequential logic units or circuits like gates , flip flops , decoders in hardware.
- ❖ Hardwired control units are generally faster than micro-programmed designs.
- ❖ This architecture is preferred in reduced instruction set computers (RISC) as they use a simpler instruction set.
- ❖ The hardwired approach has become less popular as computers have evolved as at one time, control units for CPUs were ad-hoc logic, and they were difficult to design.

CONSIST OF:

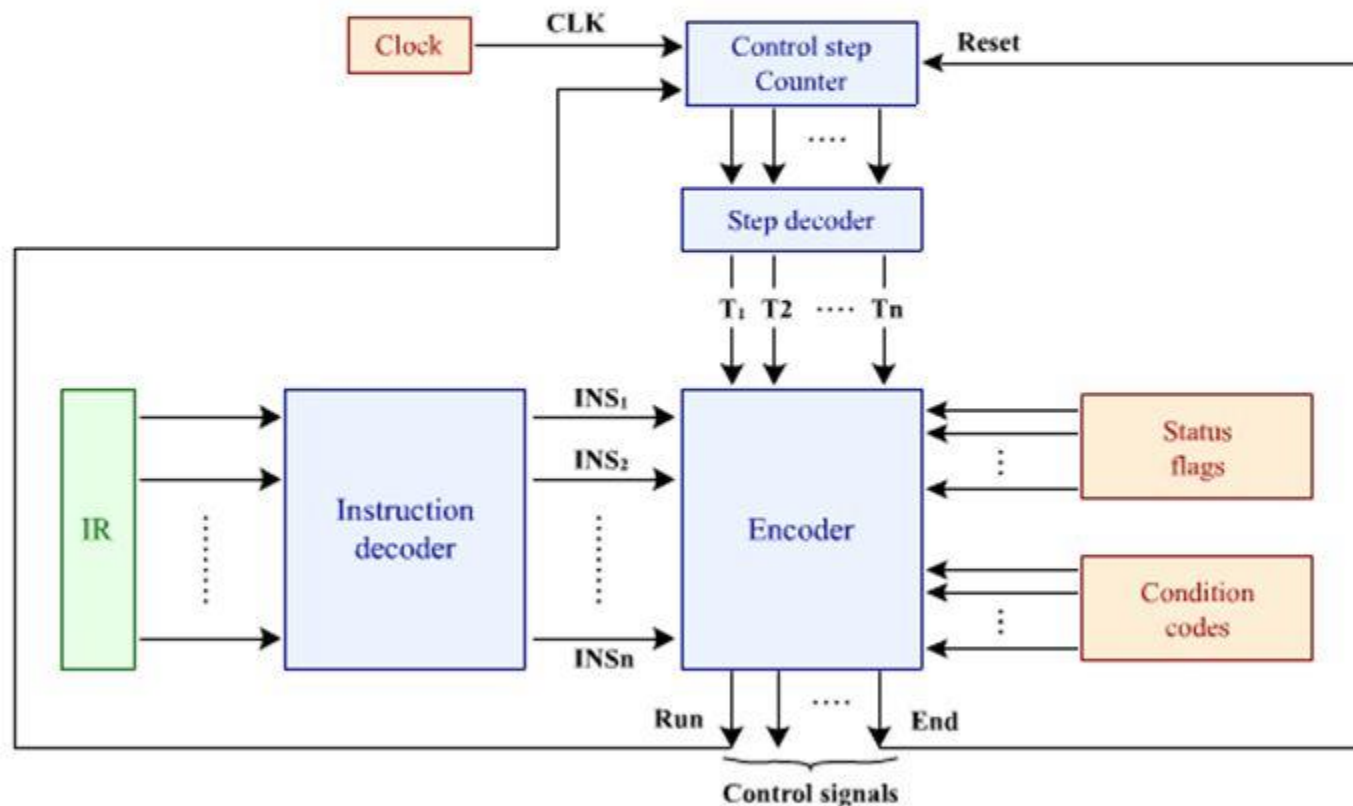
Control unit consist of a:

- ❖ **Instruction Register**
- ❖ **Number of Control Logic Gates,**
- ❖ **Two Decoders**
- ❖ **4-bit Sequence Counter**

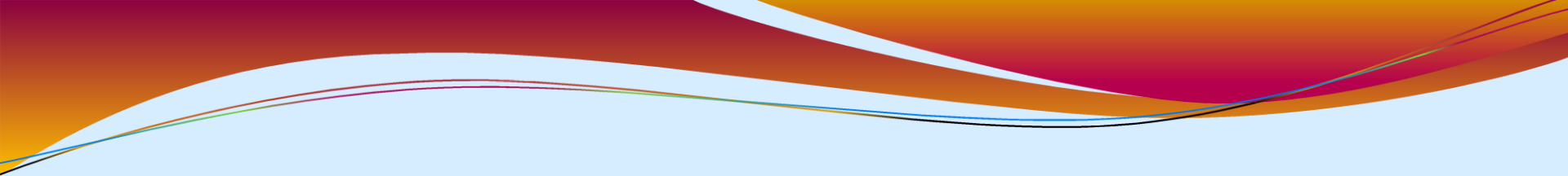
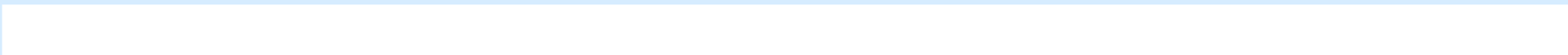


Hardwired Control Unit Design

- ❖ The general hardwired control unit organization is shown below:



- ❖ **An instruction read from memory is placed in the instruction register (IR).**
- ❖ **The instruction register is divided into three parts: the I bit, operation code, and address part.**
- ❖ **First 12-bits (0-11) to specify an address, next 3-bits specify the operation code (opcode) field of the instruction and last left most bit specify the addressing mode I.**
- ❖ **I = 0 for direct address**
- ❖ **I = 1 for indirect address**

- 
- ❖ **First 12-bits (0-11) are applied to the control logic gates.**
 - ❖ **The operation code bits (12 – 14) are decoded with a 3 x 8 decoder.**
 - ❖ **The eight outputs (D0 through D7) from a decoder goes to the control logic gates to perform specific operation.**
 - ❖ **Last bit 15 is transferred to a I flip-flop designated by symbol I.**
- 

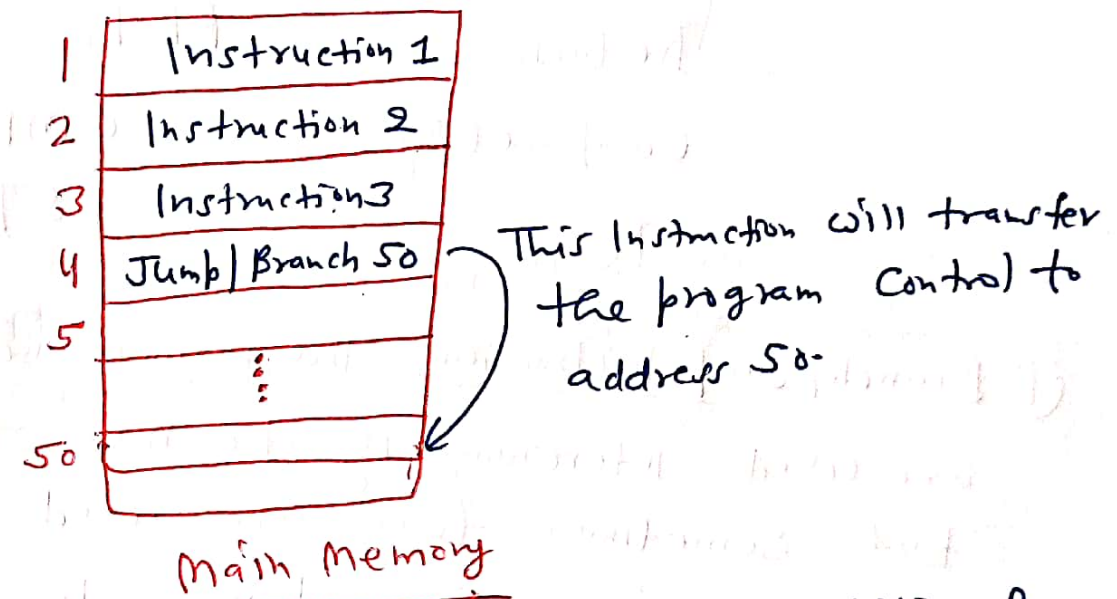
COMPARISON BETWEEN HARDWIRED AND MICROPROGRAMMED CONTROL UNIT

ATTRIBUTE	HARDWIRED CONTROL UNIT	MICRO-PROGRAMMED CONTROL UNIT
SPEED	FAST	SLOW
COST OF IMPLEMENTATION	MORE	CHEAPER
FLEXIBILITY	NOT FLEXIBLE, DIFFICULT TO MODIFY FOR NEW INSTRUCTION	FLEXIBLE, NEW INSTRUCTIONS CAN BE ADDED
ABILITY TO HANDLE COMPLEX INSTRUCTION	DIFFICULT	EASIER
DECODING	COMPLEX	EASY
APPLICATION	RISC MICROPROCESSOR	CISC MICROPROCESSOR
INSTRUCTION SET SIZE	SMALL	LARGE
CONTROL MEMORY	ABSENT	PRESENT
CHIP AREA REQUIRED	LESS	MORE

Program Control (Unit-3)

①

Program Control: A program control instruction is a instruction, which when executed, may change the address value in the program counter and causes the control of the program to be shifted to some other location rather than next location.



- * Program Control Instructions may be Conditional or Unconditional.
- * Unconditional program Control instructions causes the program control to be shifted to the new address without any condition.
- * Conditional program Control instructions causes the program control to be shifted to the new addresses only when certain status condition is met, else the program goes to next memory location in sequence.

Types of Program Control Instructions

(2)

<u>Name</u>	<u>Mnemonic</u>
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (By subtraction)	CMP
Test	TST

① Branch/Jump Instruction: The branch and Jump Instructions are used interchangeably to mean the same thing. But sometimes they are used interchangeably to ~~mean the same thing~~ denote different addressing modes. The branch is usually a one-address instruction. It is represented as **BR/JMP**.

- * Branch and jump instructions may be conditional or unconditional.
- * An unconditional branch instruction ~~specifies~~ causes a branch to the specific address without any condition.
- * The conditional branch instruction specifies a condition such as branch if positive or branch if zero.

* If the Condition is met, the PC is ③ loaded with the branch address and the next Instruction is taken from this address.

* If the Condition is not met, the Program Counter (PC) is not changed and the next Instruction is taken from the next location in sequence.

② SKIP Instruction: The SKIP Instruction does not need an address field and is therefore a Zero address Instruction.

* A Conditional SKIP Instruction will skip the next Instruction, if the Condition is met.

* If the Condition is not met, Control goes with the address of next Instruction
 $PC \leftarrow PC + 2$

③ Call & Return Instructions: These Instructions are used in conjunction with Subroutines.

* Call Instruction with Subroutine address specifies the starting address of the Subroutine.

* Return Instruction with Subroutine transfer the return address from the temporary location into the Program Counter (PC).

④ Compare & test Instructions: The Compare and test Instructions do not change the program sequence directly.

* The Compare Instruction perform a Subtraction ④ between two operands but the result of the operation is not retained. However certain Status bit Conditions are set as a result of the operation.

* The test Instruction performs the logical AND of two operands and updates certain Status Conditions without retaining the result.

Status Bit Conditions

* Status bits are also called Condition Code bits or flag bits.

* Status bits are used to indicate the results from the most recent ALU operations.

* These bits are usually stored in a status register.

Most Common Status bits

① Carry (C) → If result generate carry then $C=1$
If result does not generate carry $C=0$

② Sign (S) → If result is +ve, $S=0$
If result is -ve, $S=1$

③ Zero (Z) → If result is zero, $Z=1$
If result is not-zero, $Z=0$

④ Overflow (V) → If result overflow, $V=1$
If result no-overflow, $V=0$

Subroutine Call & Return:

(5)

* A Subroutine is a Self-Contained Sequence of Instructions that performs a given computational task.

* Each time a Subroutine is called, a branch is executed to the beginning of the Subroutine to start executing its set of Instructions.

* After the Subroutine has been executed, a branch is made back to the main program.

* Subroutine Call/Call Subroutine consists of an operation code together with an address that specifies the beginning of the Subroutine.

- (a) The address of the next Instruction (PC) is stored in a temporary location.
- (b) Control is transferred to the beginning of the Subroutine.

$SP \leftarrow SP - 1$
 $MCSP \leftarrow PC$
 $PC \leftarrow \text{effective address}$
Subroutine

The last Instruction of every
Called "Return from Subroutine"

* Return from Subroutine transfer the return address from the temporary location into the program Counter.

$PC \leftarrow MCSP$

$SP \leftarrow SP + 1$

Pop stack & transfer to PC

Increment Stack pointer

A recursive Subroutine is a Subroutine that calls itself.

Program Interrupt and types of Interrupts

(6)

Program Interrupt: Program Interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.

- * The Interrupt is usually initiated by an internal or external signal.
- * The address of the Interrupt service program is determined by the hardware.
- * An Interrupt procedure usually stores all the information necessary to define the state of the CPU.

* Types of Interrupts

(a) External Interrupts ^(H/W Interrupt): Comes from I/O devices, from a timing device, from a circuit monitoring the power supply or from any external source.

e.g.: I/O devices requesting transfer of data, I/O device finished transfer of data, Power failure.

(b) Internal Interrupts ^(H/W Interrupt): Internal interrupts arise from illegal or erroneous use of an instruction or data. They are also called traps. Examples

of Internal Interrupts are register overflow, \oplus divide by zero, invalid operation code, stack overflow. These error conditions usually occur as a result of a premature termination of the instruction execution. Both external and internal interrupts are hardware interrupts

(c) Software Interrupts:

* A Software Interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call.

* It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.

* Supervisor Call instruction causes to switch from user mode to supervisor mode.

* Supervisor Call instruction causes a SW interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode.

Define the following: (a) Micro-operation (b) Micro-instruction
(c) Microprogram (d) Microcode

Ans (a) Micro-operation: An elementary digital computer operation.

(b) Micro-instruction: An instruction stored in control memory.

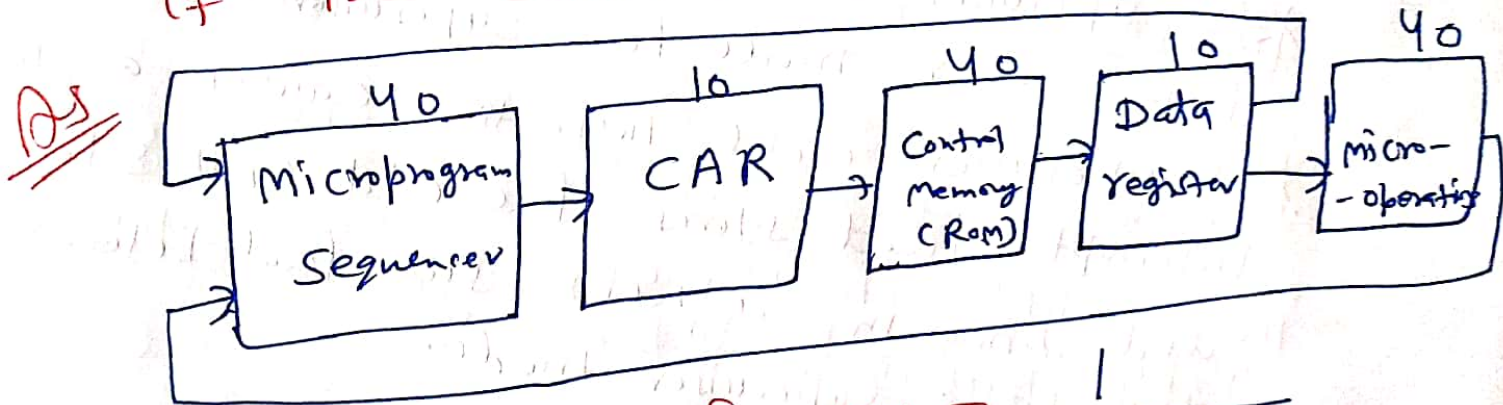
(c) Micro-program: A sequence of micro-instructions.

(d) Micro-code: Micro-code is same as microprogram.

Numerical on microprogrammed control unit (8)

The microprogrammed Control unit has the following propagation delay times: 40 ns to generate the next address, 10 ns to transfer the address into the Control address register, 40 ns to access the Control memory ROM, 10 ns to transfer the microinstruction into the control data register and 40 ns to perform the required micro-operations specified by the control word.

- (i) What is the maximum clock frequency that the control can use?
(ii) What would be the clock frequency be if the control data register is not used?



Ans (i) Max. clock frequency = $\frac{1}{(40 + 10 + 40 + 10) \text{ ns}}$
 $= \frac{1}{100 \times 10^{-9}} = 10 \times 10^6$
 $= \underline{\underline{10 \text{ MHz}}}$

- (ii) If control data register is not used.

clock frequency = $\frac{1}{40 + 10 + 40}$
 $= \frac{1}{90 \times 10^{-9}} = \underline{\underline{11.1 \text{ MHz}}}$