Control Unit Operation

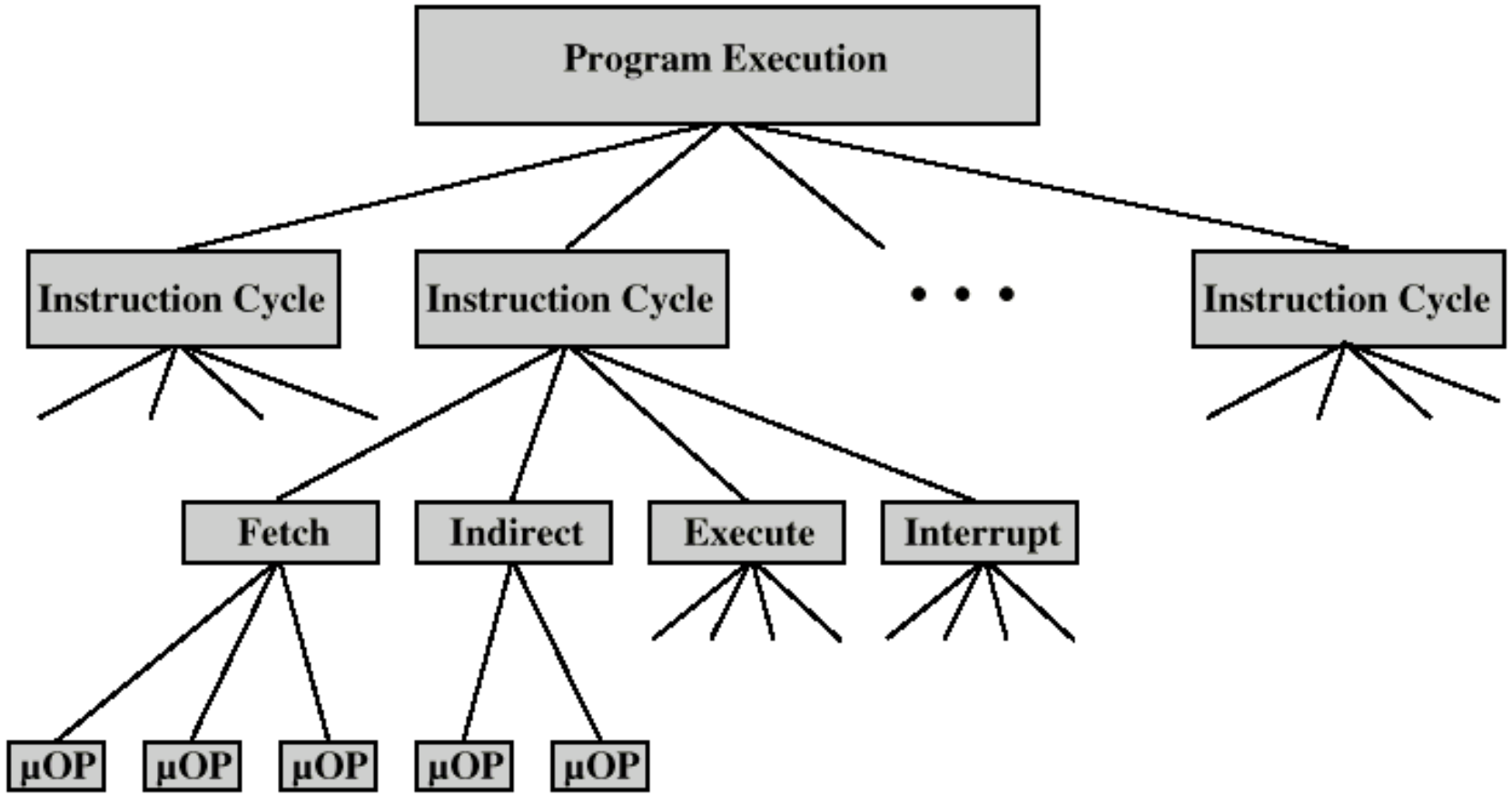
Module 4: The Central Processing

Basic Processing Units: Fundamental concepts,
Instruction Sequencing,
Execution cycle,
Hardwired control,
Micro programmed control.

Micro-Operations

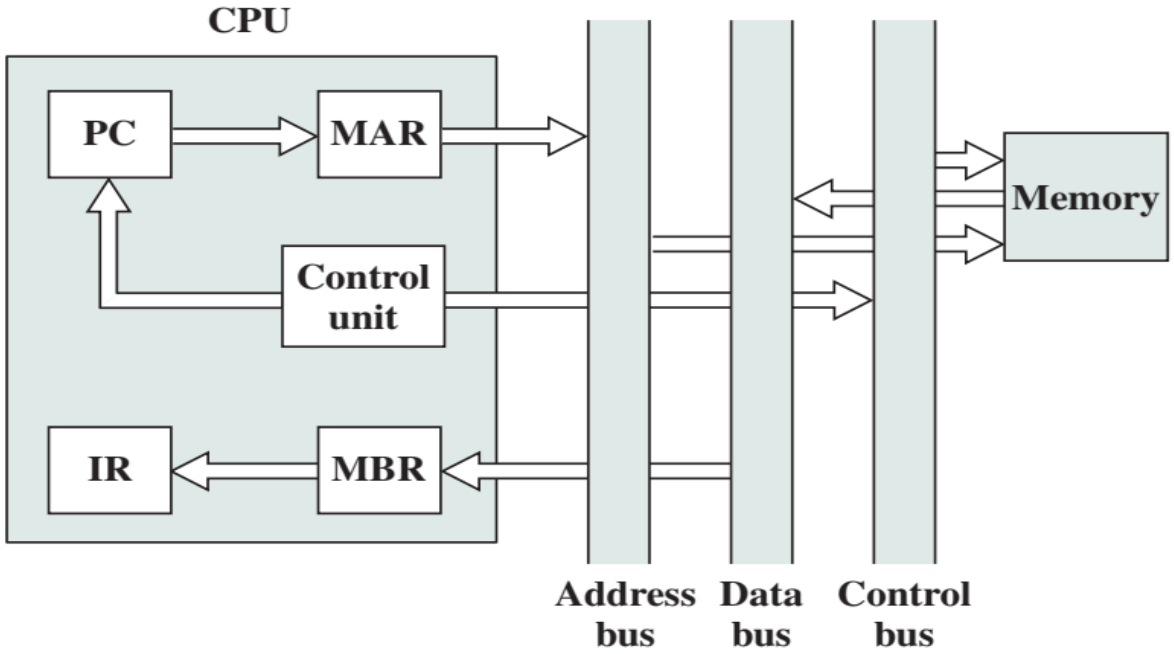
- A computer executes a program
- Fetch/execute cycle
- Each cycle has a number of steps called as micro-operations
- Each step does very little called atomic operation of CPU

Constituent Elements of Program Execution



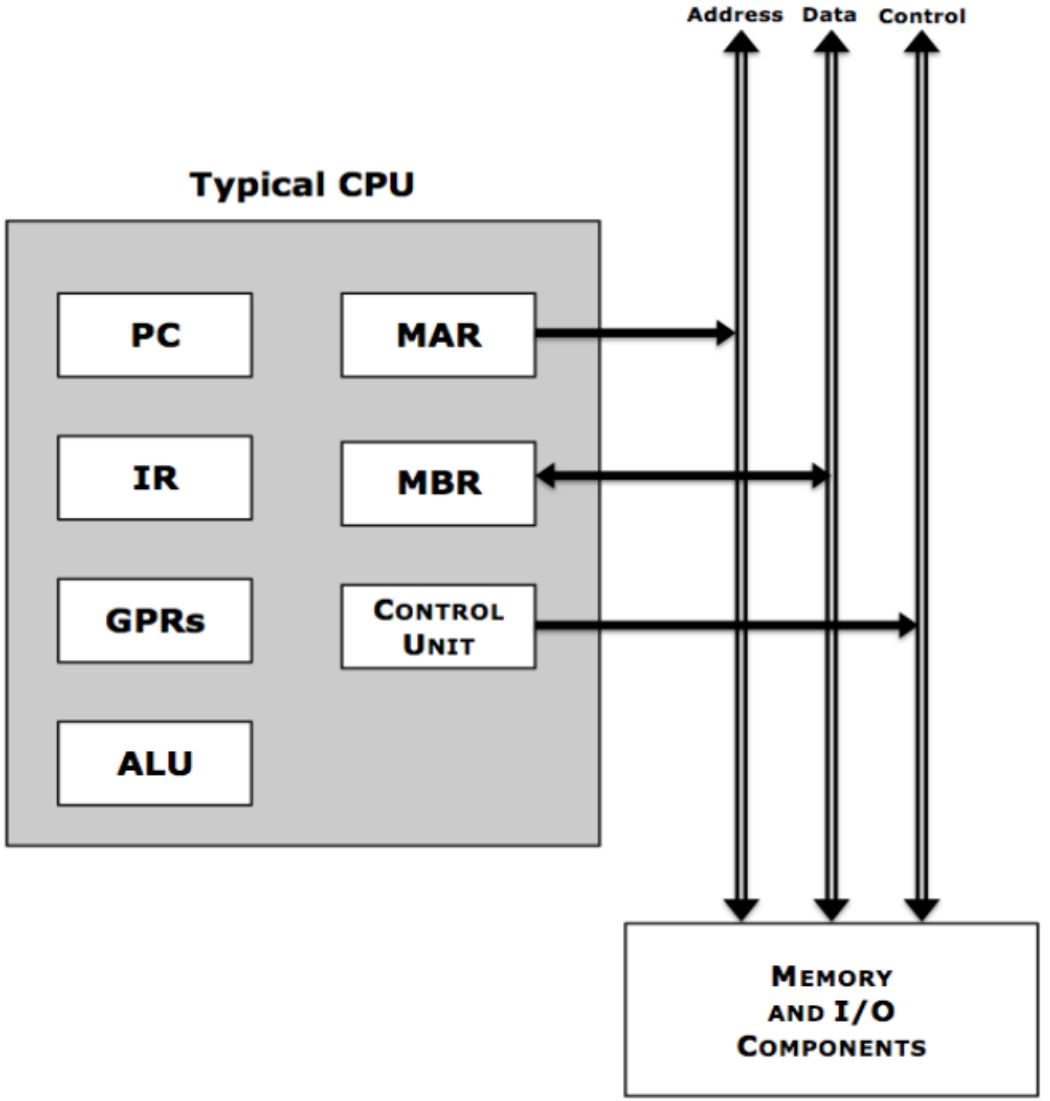
Fetch - 4 Registers

- Memory Address Register (MAR)
  - Connected to address bus
  - Specifies address for read or write op
- Memory Buffer Register (MBR)
  - Connected to data bus
  - Holds data to write or last data read
- Program Counter (PC)
  - Holds address of next instruction to be fetched
- Instruction Register (IR)
  - Holds last instruction fetched



Fetch Sequence

- Address of next instruction is in PC
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC = PC+1 (in parallel with data fetch from memory)
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches



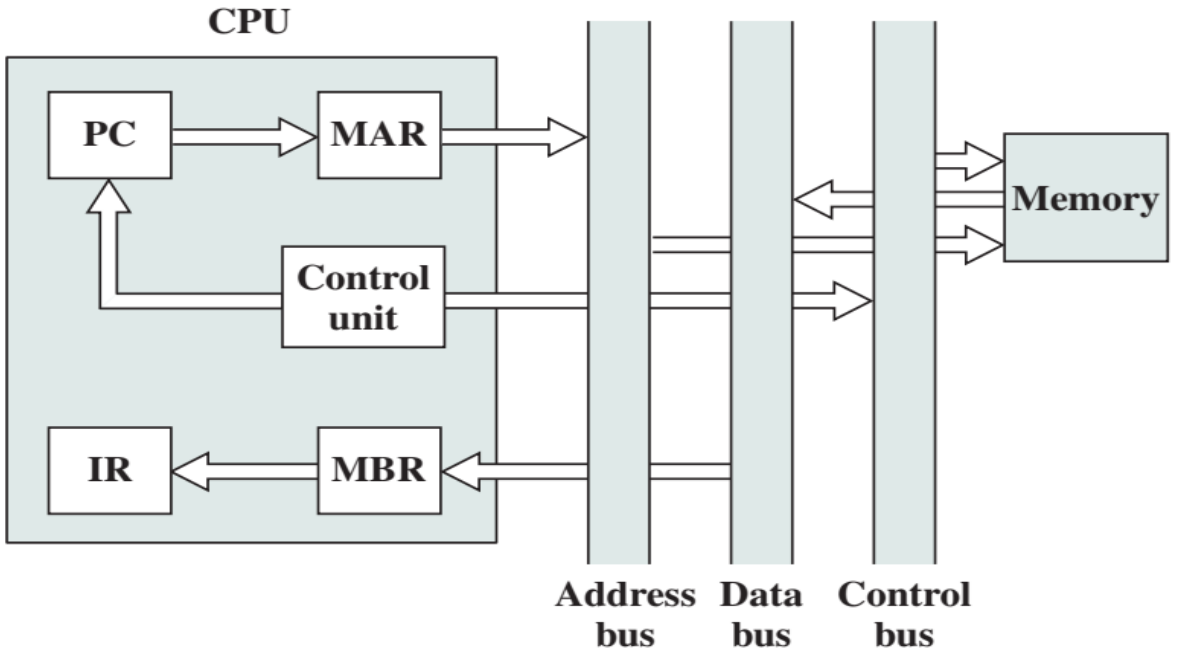
Fetch Sequence (symbolic)

- t1: MAR <- (PC)
- t2: MBR <- (memory)
  PC <- (PC) +1
- t3: IR <- (MBR)

(tx = time unit/clock cycle)

or

- t1: MAR <- (PC)
- t2: MBR <- (memory)
- t3: PC <- (PC) +1
  IR <- (MBR)



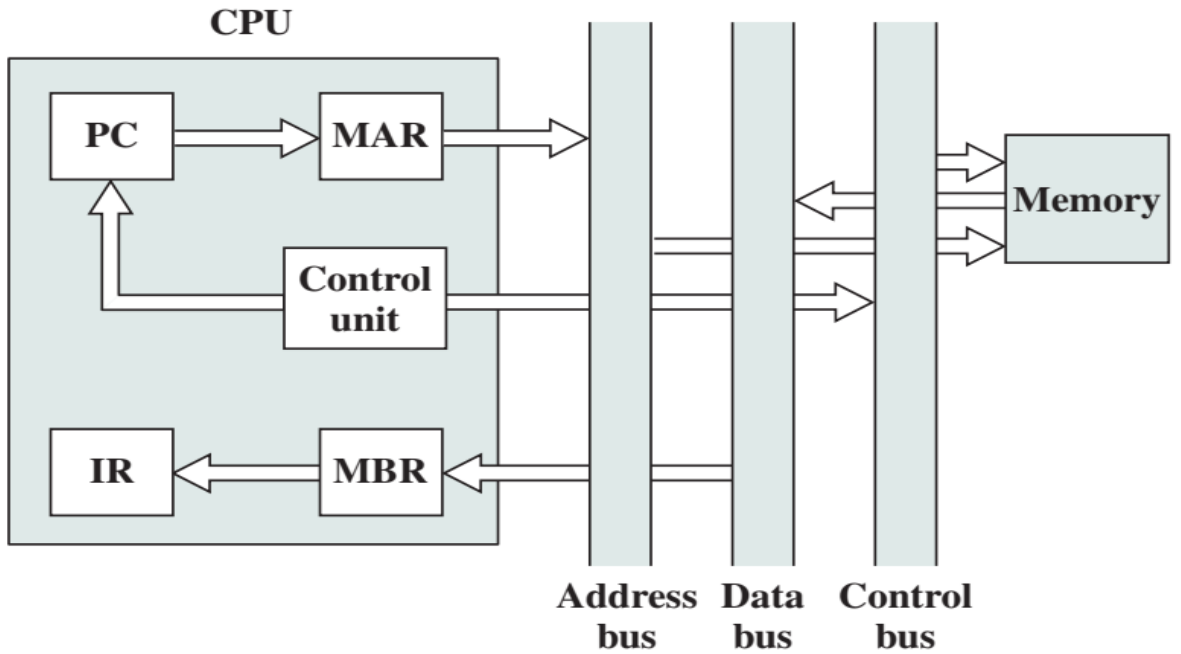
Rules for Clock Cycle Grouping

- Proper sequence must be followed
  - MAR <- (PC) must precede MBR <- (memory)
- Conflicts must be avoided
  - Must not read & write same register at same time
  - MBR <- (memory) & IR <- (MBR) must not be in same cycle
- Also: PC <- (PC) +1 involves addition
  - Use ALU
  - May need additional micro-operations

Indirect Cycle

- MAR <- (IR(address)) - address field of IR
- MBR <- (memory)
- IR(address) <- (MBR(address))

- MBR contains an address
- IR is now in same state as if direct addressing had been used

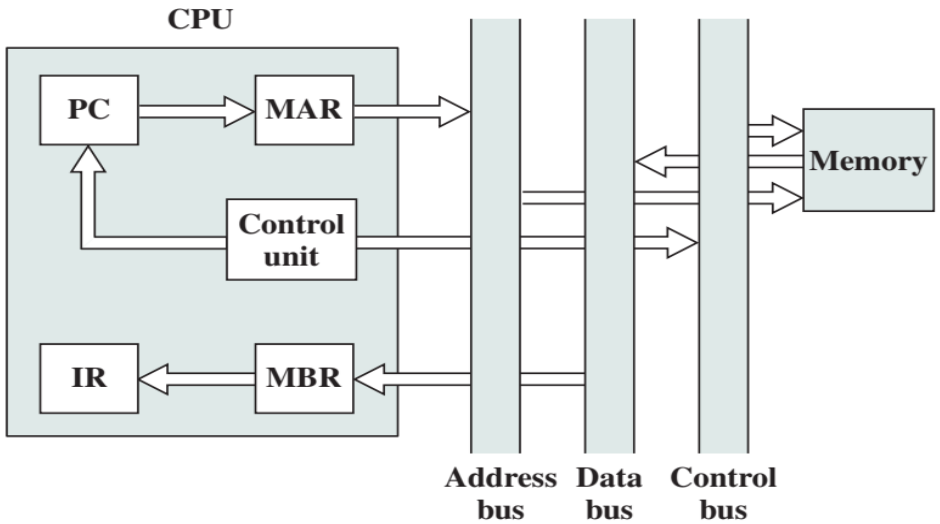


Execute Cycle (ADD)

- Different for each instruction
- e.g. ADD R1,X - add the contents of location X to Register 1 , result in R1
- t1: MAR <- ($IR_{address}$)
- t2: MBR <- (memory)
- t3: R1 <- R1 + (MBR)
- Note no overlap of micro-operations

Interrupt Cycle

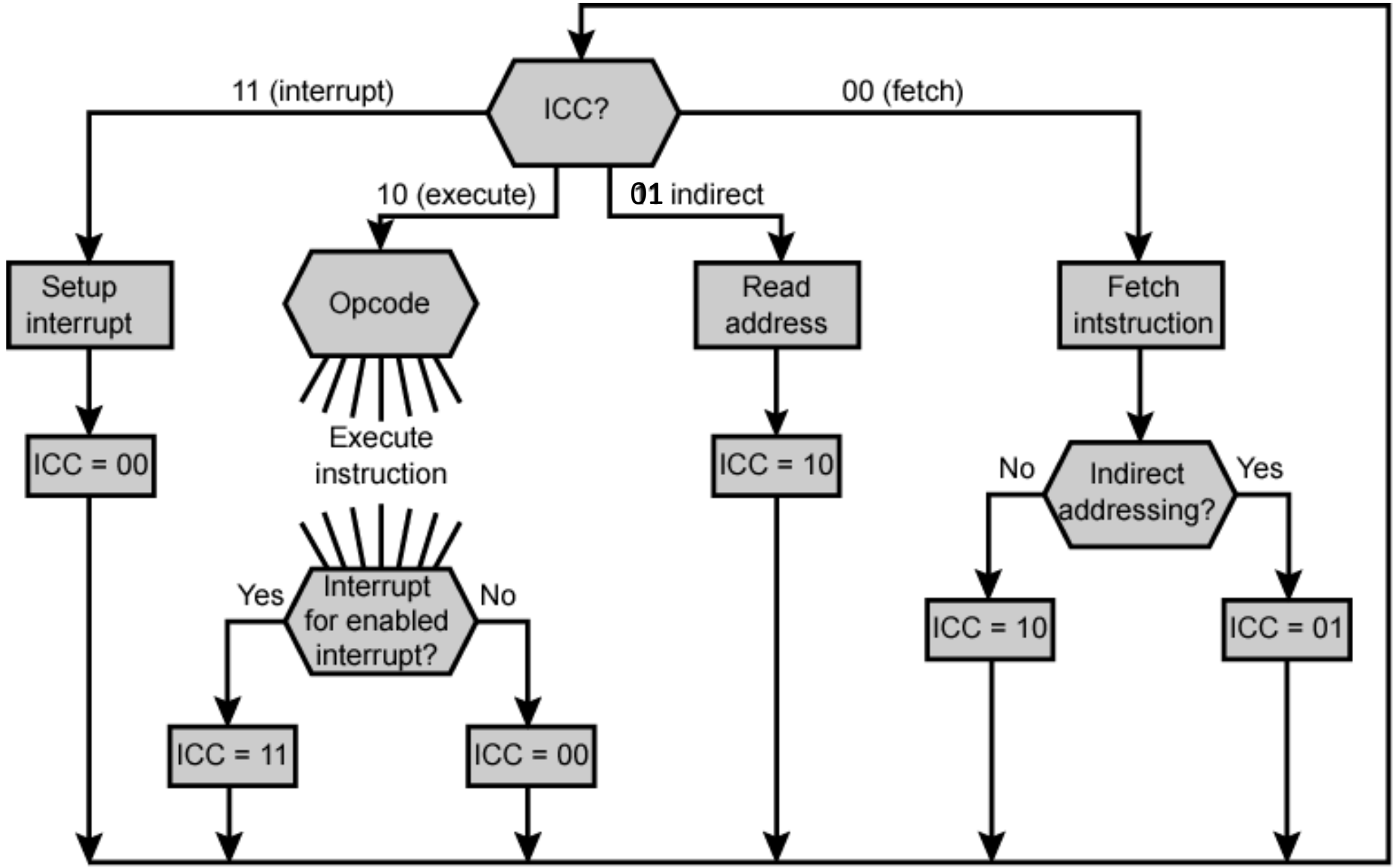
- t1: MBR <-(PC)
- t2: MAR <- save-address (Address where the content of current PC will be stored)
- PC <- Interrupt service routine-address
- t3: memory <- (MBR) (Content of MBR ie. Previous PC will be stored in Mem)
- This is a minimum
  - May be additional micro-ops to get addresses
  - N.B. saving context is done by interrupt handler routine, not micro-ops



Instruction Cycle

- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
  - Instruction cycle code (ICC) designates which part of cycle processor is in
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

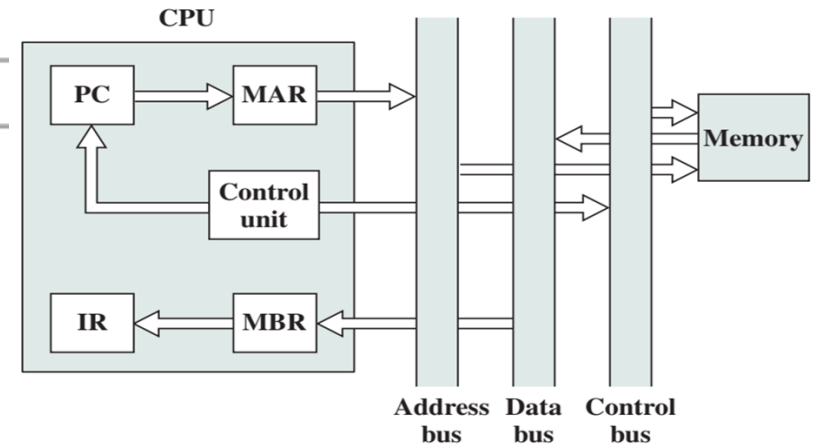
Flowchart for Instruction Cycle



Micro-Operations for Immediate Addressing Mode

E.g.: MOV R1, 25H; R1 register gets the immediate value 25H

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, 25H" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	R1 ← 25H (IR)	*R1 register gets the value 25H from IR*



Micro-Operations for Register Addressing Mode

E.g.: MOV R1, R2; R1 register gets the data from Register R2

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, R2" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	R1 ← R2	*R1 register gets the value from R2 Register*

Micro-Operations for Direct Addressing Mode

E.g.: MOV R1, [2000H]; R1 register gets the data from memory location 2000H

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, [2000H]" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← IR (2000H)	*MAR gets the address 2000H from IR*
T5:	MBR ← Memory ([2000H])	*MBR gets contents of location 2000H from Memory.*
T6:	R1 ← MBR ([2000H])	*Register R1 gets contents of memory location 2000H from MBR*

Micro-Operations for Implied Addressing Mode

E.g.: STC; Set the Carry Flag; (CF ← 1).

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "STC" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	CF ← 1	*Carry Flag in the Flag Register becomes 1*

Micro-Operations for Indirect Addressing Mode

E.g.: MOV R1, [R2]; R1 register gets the data from memory location pointed by R2

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, [R2]" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← Memory ([R2])	*MBR gets contents of location pointed by R2 from Memory.*
T6:	R1 ← MBR ([R2])	*Register R1 gets contents of memory location pointed by R2 from MBR*
		In the exam, once, Add R1, [R2] was asked. *Everything else will be same. Only change: T6: R1 ← R1 + MBR*

Micro-Operations for Indirect Addressing Mode

E.g.: MOV [R2], R1; R1 register stores data into memory location pointed by R2

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV [R2], R1" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← R1	*R1 puts data into MBR to store it in the memory location pointed by R2.*

Functional Requirements

- Define basic elements of processor
- Describe micro-operations processor performs
- Determine functions control unit must perform

Basic Elements of Processor

- ALU
- Registers
- Internal data paths
- External data paths
- Control Unit

Types of Micro-operation

- Transfer data between registers
- Transfer data from register to external
- Transfer data from external to register
- Perform arithmetic or logical operations

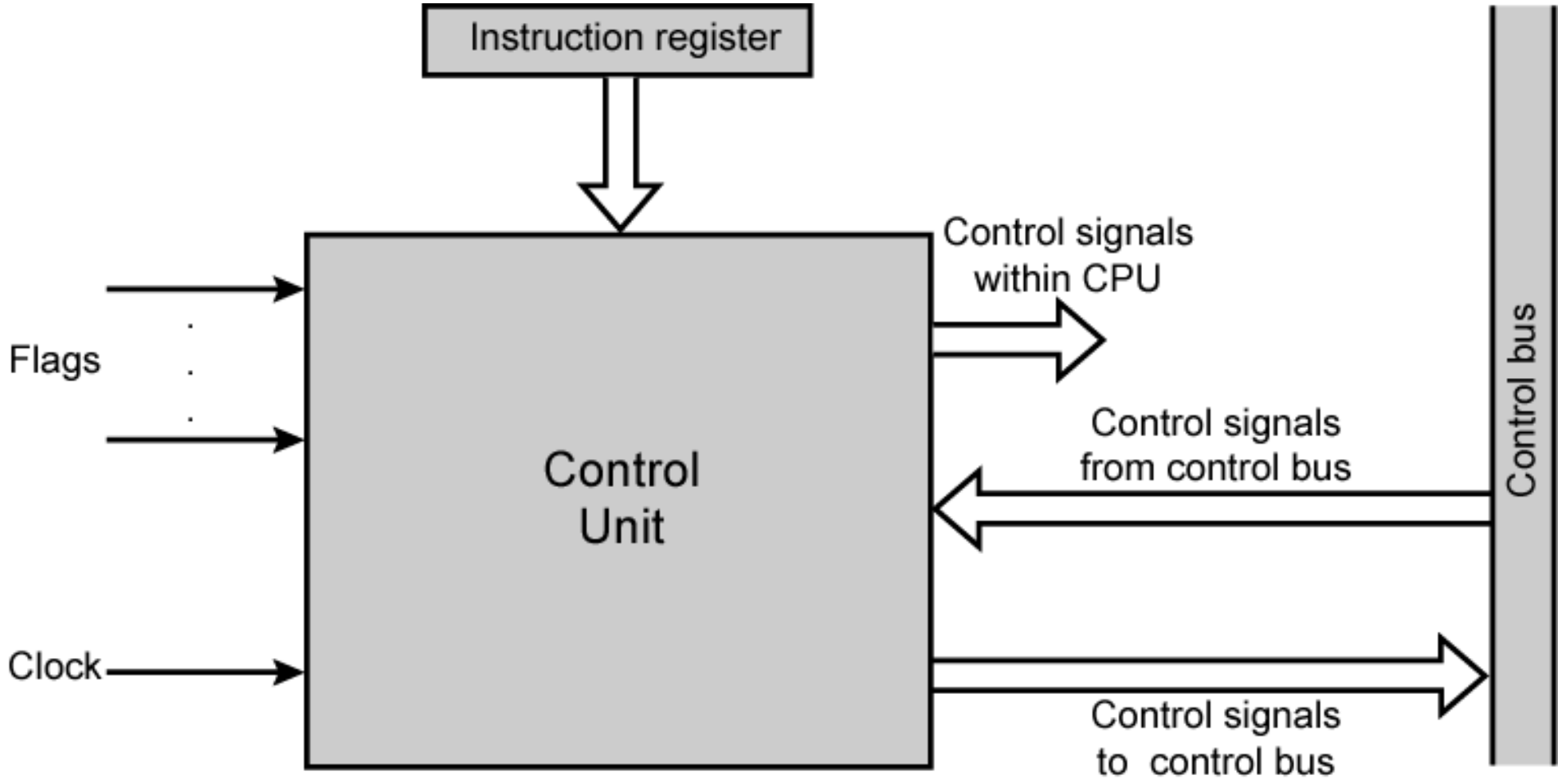
Functions of Control Unit

- Sequencing
  - Causing the CPU to step through a series of micro-operations
- Execution
  - Causing the performance of each micro-op
- This is done using Control Signals

Control Signals-Input

- Clock
  - One micro-instruction (or set of parallel micro-instructions) per clock cycle
- Instruction register
  - Op-code for current instruction
  - Determines which micro-instructions are performed
- Flags
  - State of CPU
  - Results of previous operations
- From control bus
  - Interrupts
  - Acknowledgements

Model of Control Unit



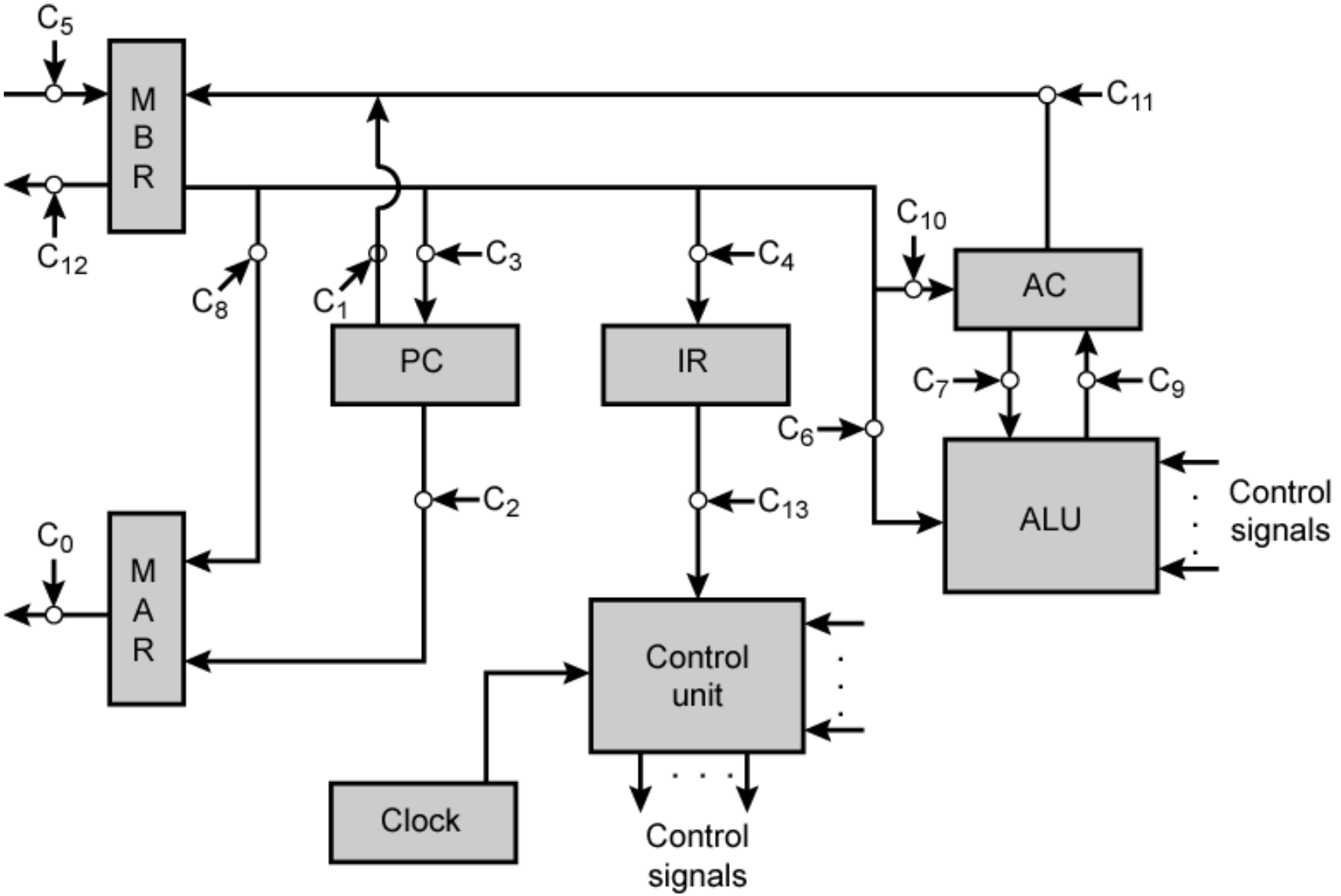
Control Signals - output

- Within CPU
  - Cause data movement
  - Activate specific functions
- Via control bus
  - To memory
  - To I/O modules

Example Control Signal Sequence - Fetch

- MAR <- (PC)
  - Control unit activates signal to open gates between PC and MAR
- MBR <- (memory)
  - Open gates between MAR and address bus
  - Memory read control signal
  - Open gates between data bus and MBR

Data Paths and Control Signals



Micro operations and control signals

	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R = Read control signal to system bus.

C_W = Write control signal to system bus.

For simplicity, the data and control paths for incrementing the PC and for loading the fixed addresses into the PC and MAR are not shown.

Control unit implementation

Implementation of Control unit is broadly of two types

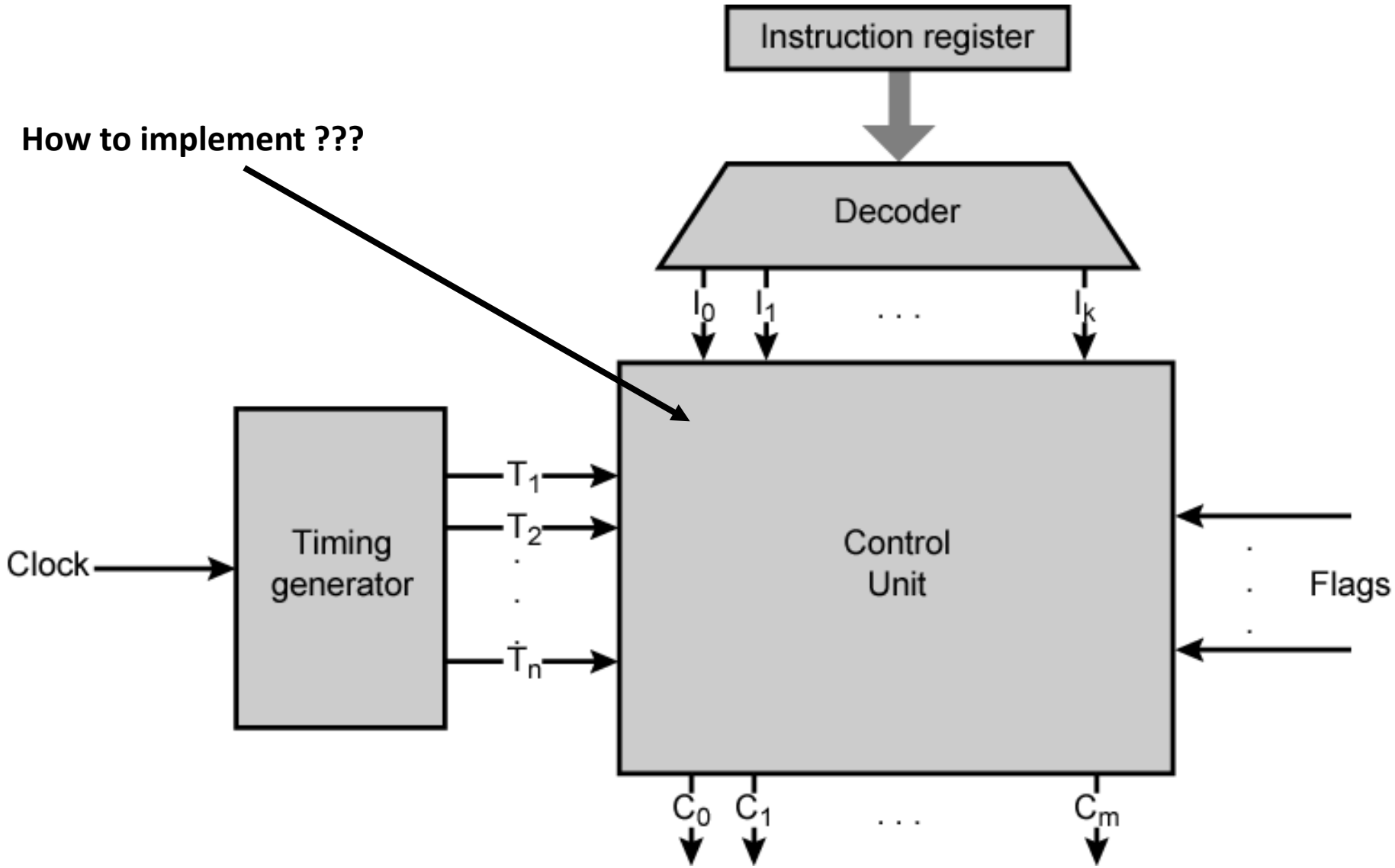
- Hardwired implementation (RISC)
- Microprogrammed implementation (CISC)

Hardwired Implementation

Control unit inputs

- Flags and control bus
  - Each bit means something
- Instruction register
  - Op-code causes different control signals for each different instruction
  - Unique logic for each op-code
  - Decoder takes encoded input and produces single output
  - n binary inputs and 2^n outputs
- Clock
  - Repetitive sequence of pulses
  - Useful for measuring duration of micro-ops
  - Must be long enough to allow signal propagation
  - Different control signals at different times within instruction cycle
  - Need a counter with different control signals for t1, t2 etc.

Control Unit with Decoded Inputs



4:16 DECODER

I1	I2	I3	I4	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15	O16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Hardwired control unit methods

State table method:

T-States	I1	I2	I3	----	In
T1	C11	C12	C13	---	C1n
T2	C21	C22	C23	---	C2n
---	---	---	---	---	---
Tm	Cm1	Cm2	Cm3	---	Cmn

1) It is the most basic type of hardwired control unit.
2) Here the behavior 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 every instruction.
5) A circuit is then constructed based on every column of this table, for each instruction.

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 many circuit elements in various instructions.

Hardwired Control Unit Logic

- For each control signal, to derive a Boolean expression of that signal as a function of the inputs
- Let us consider a **single control signal, C5**, which causes data to be read from the external data bus into the MBR
- Let us define two new **control signals, P and Q**, that have the following interpretation:

  PQ = 00 Fetch Cycle

  PQ = 11 Interrupt Cycle

  PQ = 10 Execute Cycle

  PQ = 01 Indirect Cycle

State table Logic Example

- Then C5 can be defined as:

$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2$$

- That is, the control signal C5 will be asserted during the second time unit of both the fetch and indirect cycles.
- This is not complete
- C5 is also needed during the execute cycle. For our simple example, let us assume that there are only three instructions that read from memory: LDA, ADD and AND. Now we can define C5 as

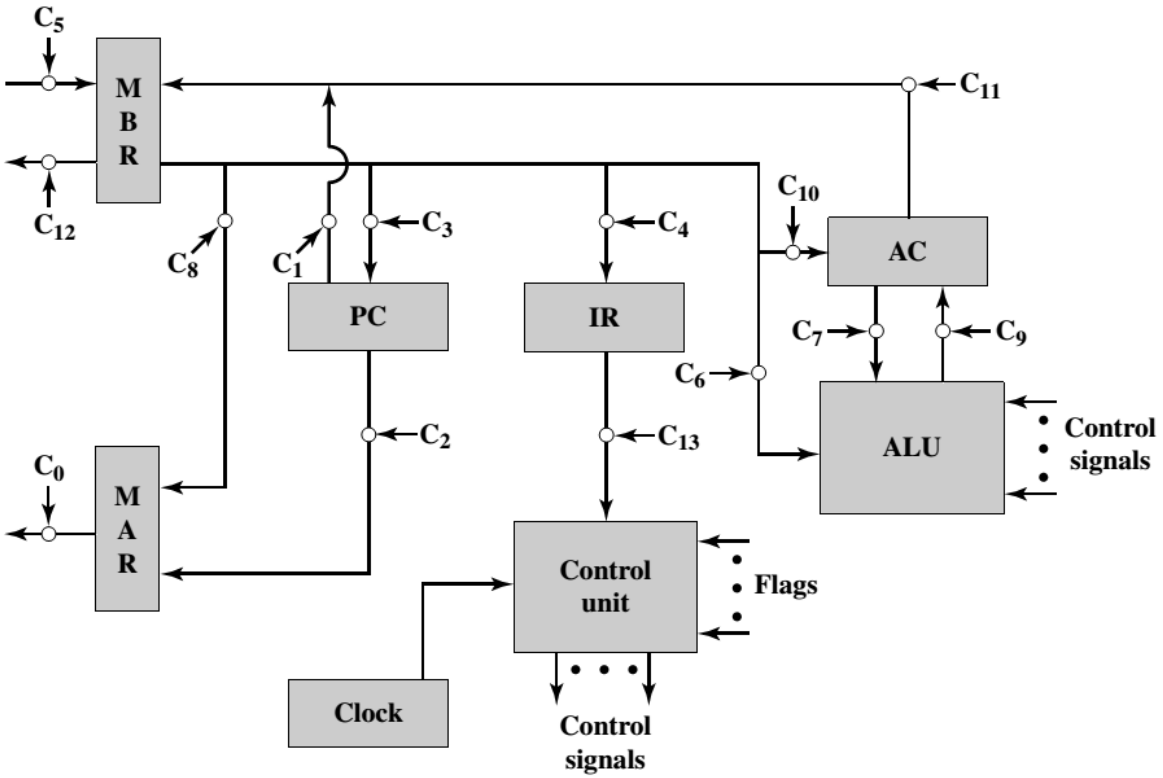
$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2 + P \cdot \overline{Q} \cdot (LDA + ADD + AND) \cdot T_2$$

- Is it that simple?
- No. In a modern complex processor, the number of Boolean equations needed to define the control unit is very large.

	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

PQ = 00 Fetch Cycle
PQ = 11 Interrupt Cycle
PQ = 10 Execute Cycle
PQ = 01 Indirect Cycle

*Hence an efficient and simpler approach, known as **Flow chart/ delay element method** is usually used*



	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

- Referring to the figures above, depict the control signals required for the execution cycle of LOAD 200 and STORE 300. **[5]**
- Express the Boolean expression for each of the control signals. **[5]**
- Implement the Boolean expressions for each of the control signals using minimum number of gates. **[5]**

4: a: LOAD

t_1	MAR ← (IRaddress)	C_8
t_2	MBR ← (Memory)	C_5, C_R
t_3	AC ← (MBR)	C_{10}

[2.5]

STOR

t_1	MAR ← (IRaddress)	C_8
t_2	MBR ← AC	C_{11}
t_3	Memory ← MBR	C_{12}, C_W

[2.5]

b: Suppose. LOAD → A and STOR → B

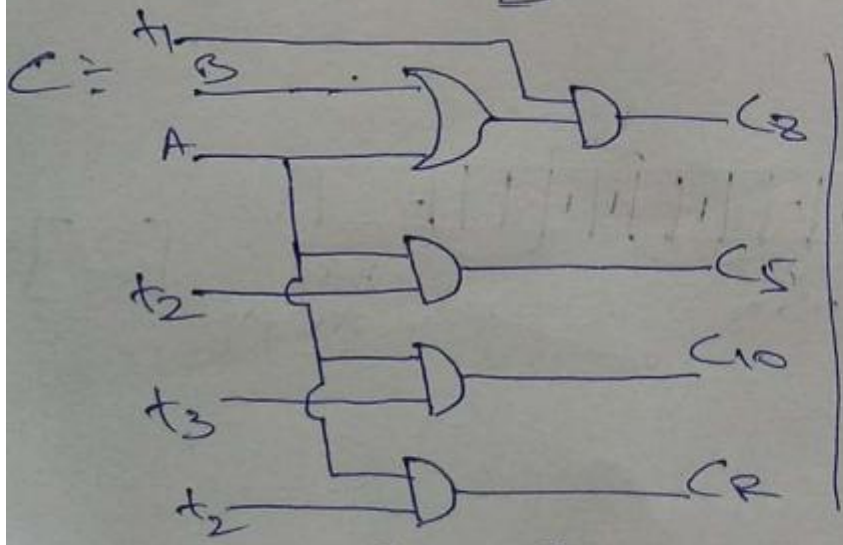
$C_8 = At_1' + Bt_1$
$C_5 = At_2$
$C_{10} = At_3$
$C_R = At_2$

[2.5]

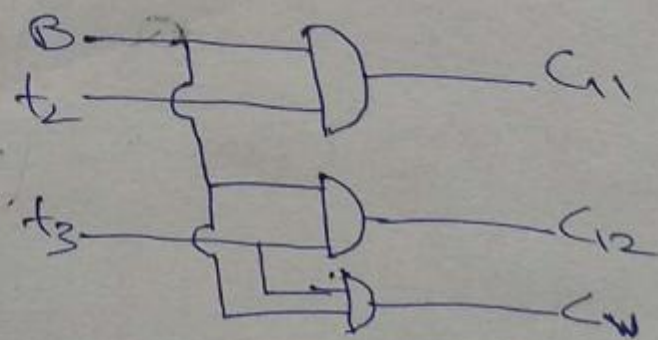
$C_{11} = Bt_2$
$C_{12} = Bt_3$
$C_W = Bt_3$

[2.5]

c:



[2.5]



[2.5]

Micro-Operations for Indirect Addressing Mode

E.g.: MOV R1, [R2]; R1 register gets the data from memory location pointed by R2

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, [R2]" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← Memory ([R2])	*MBR gets contents of location pointed by R2 from Memory.*
T6:	R1 ← MBR ([R2])	*Register R1 gets contents of memory location pointed by R2 from MBR*
		In the exam, once, Add R1, [R2] was asked. *Everything else will be same. Only change: T6: R1 ← R1 + MBR*

Micro-Operations for Indirect Addressing Mode

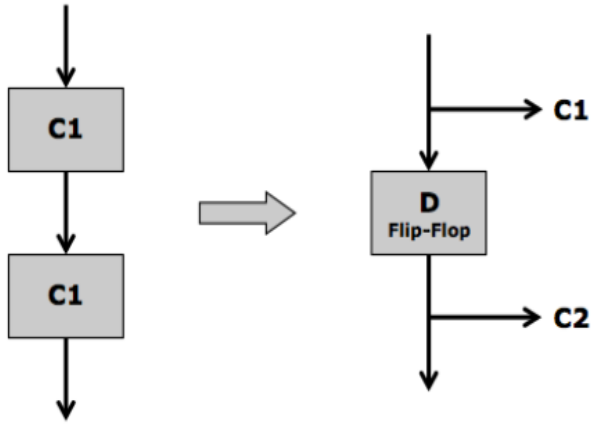
E.g.: MOV [R2], R1; R1 register stores data into memory location pointed by R2

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV [R2], R1" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← R1	*R1 puts data into MBR to store it in the memory location pointed by R2.*

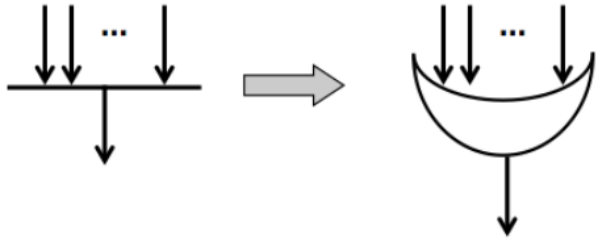
Flow chart/ Delay element method:

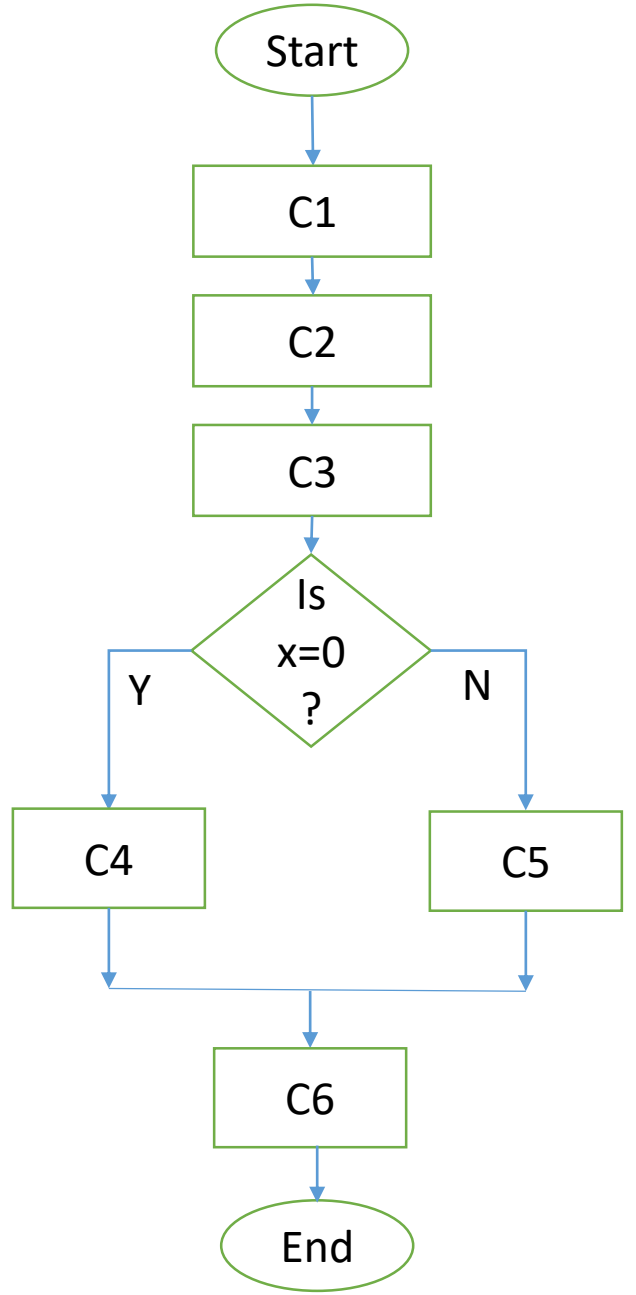
DELAY ELEMENT METHOD

1) Here the behavior 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**.

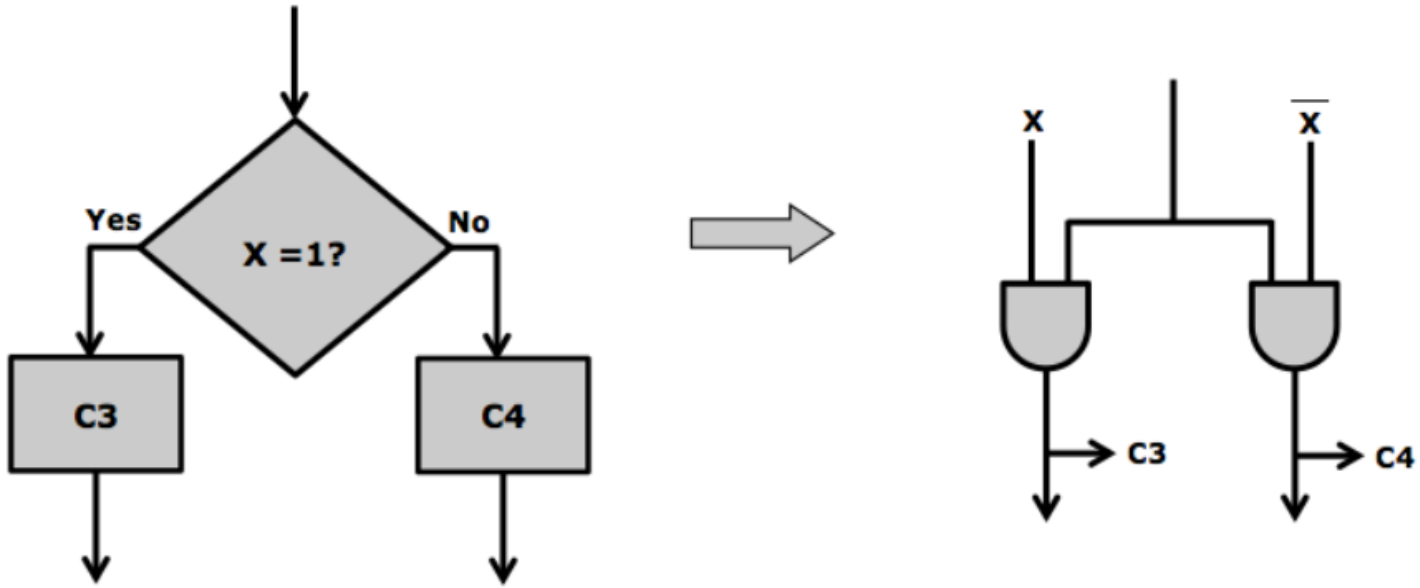


8) Of all D Flip-Flops only one will be active at a time. So the method is also called **"One Hot Method"**.
9) In a **multiple entry point**, to combine two or more paths, we use an **OR gate**.





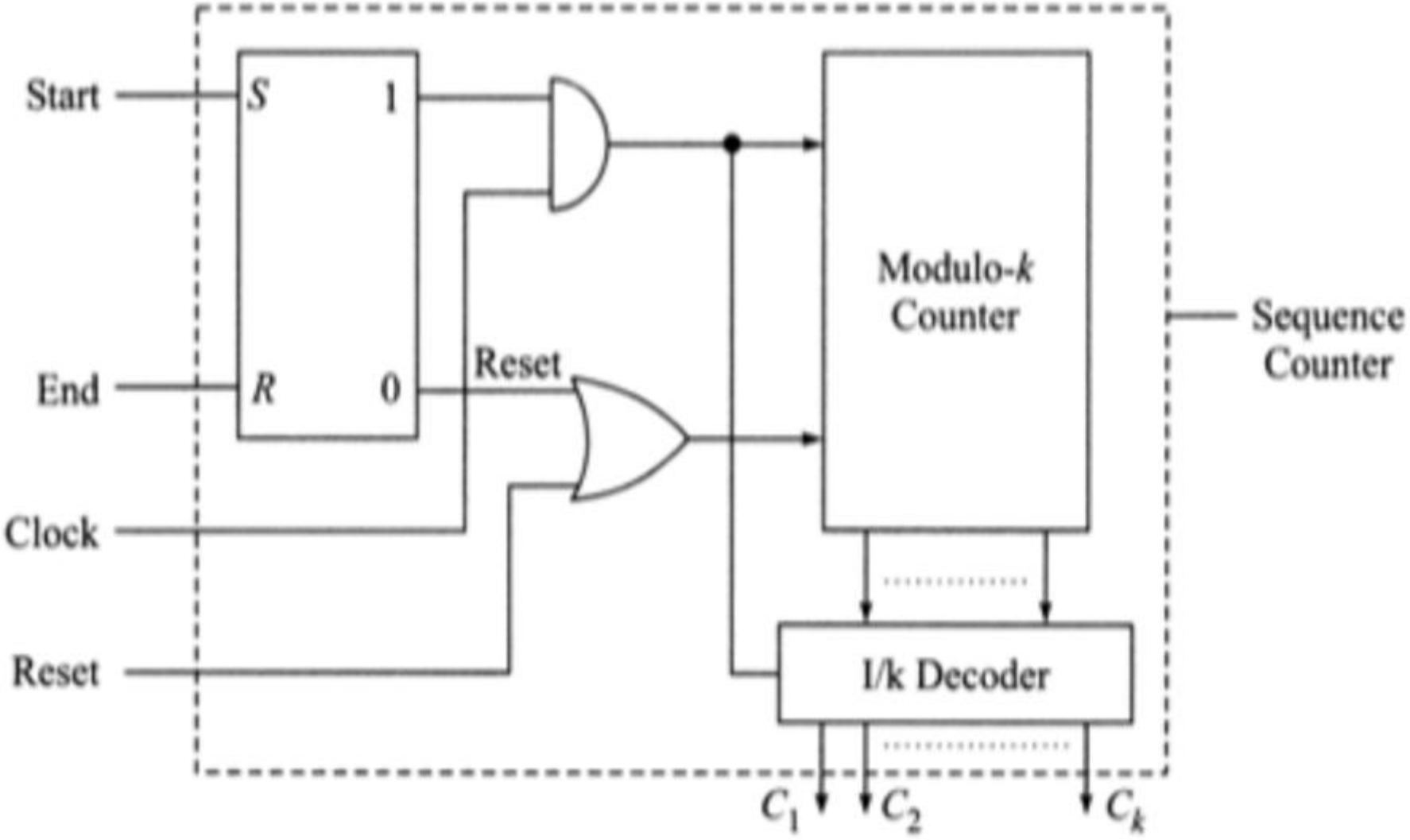
10)A **decision box** is replaced by a set of two complementing **AND gates**



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.
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.
3) First a flowchart is made representing the behavior of a control unit.
4) It is then **converted into a circuit** using the same principle of AND & OR gates.
5) We need a **delay of 1 T-state** (one clock cycle) between every two **consecutive control signals**.
6) That is achieved by the above circuit.
7) If there are **"k" number of distinct steps** producing control signals, we employ a **"mod k"** and **"k" output decoder**.
8) The counter will **start counting** at the beginning of the instruction.
9) The "clock" input via an AND gate ensures each count will be generated **after 1 T-state**.
10) The count is given to the **decoder** which **triggers the generation of "k" control signals**, each after a delay of 1 T-state.
11) When the **instruction ends**, the **counter is reset** so that next time, it begins from the first count.

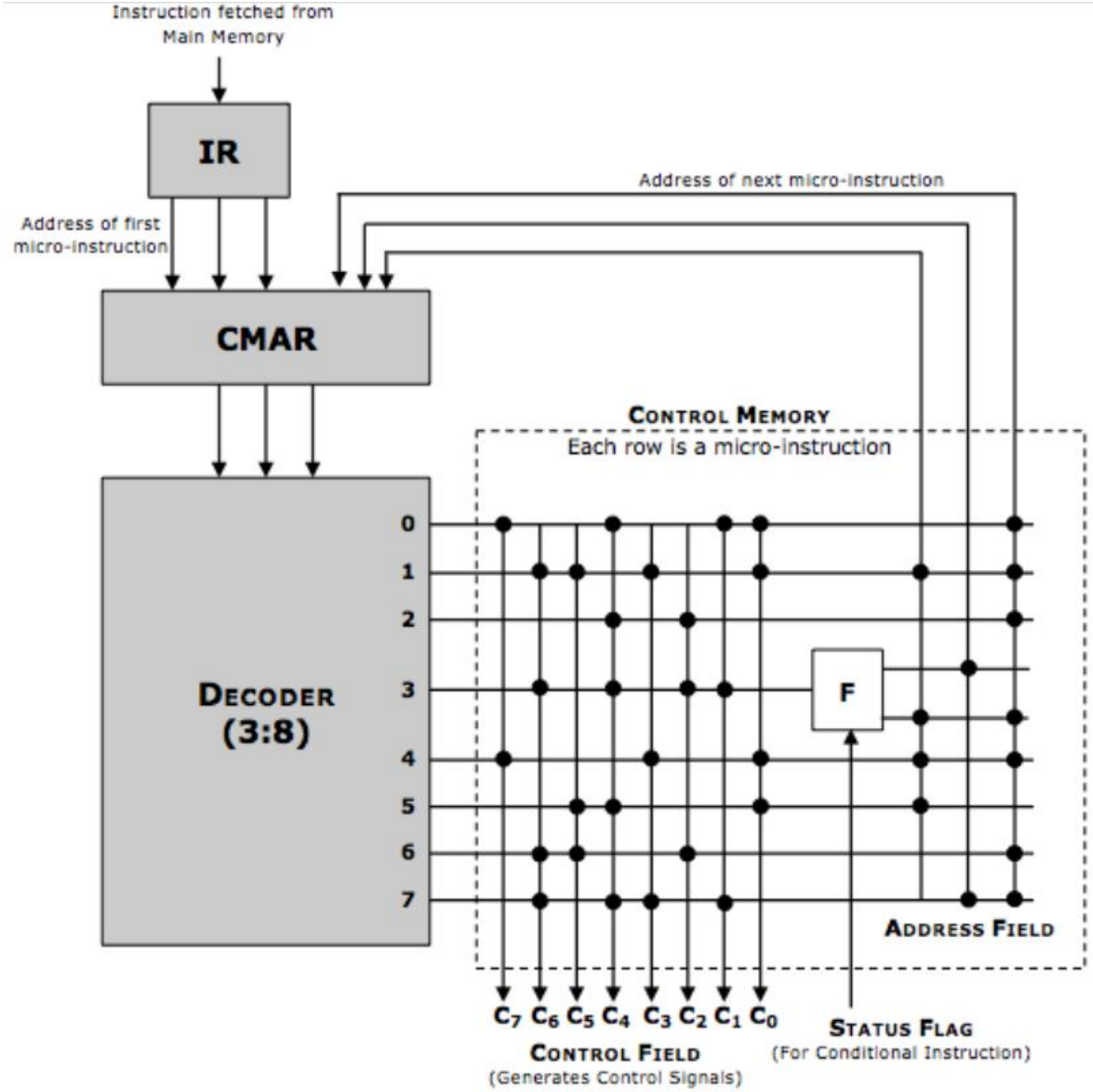
ADVANTAGE:

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.

Wilkes' Design for Micro programmed 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**.
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).
13) CMAR passes the address to the **decoder**.
14) The decoder **identifies the corresponding micro-instruction** from the Control Memory.
15) A micro-instruction has **two fields**: a control filed and an address field.
16) **Control field:** Indicates the **control signals to be generated**.
17) **Address field:** Indicates the **address of the next micro-instruction**.
18) This address is further **loaded into CMAR** to **fetch the next** micro-instruction.
19) For a **conditional micro-instruction**, there are **two address fields**.
20) This is because, the address of the next micro-instruction **depends on the condition**.
21) The condition (true or false) is **decided by the appropriate control flag**.
22) The control memory is usually implemented using FLASH ROM as it is writable yet non volatile.

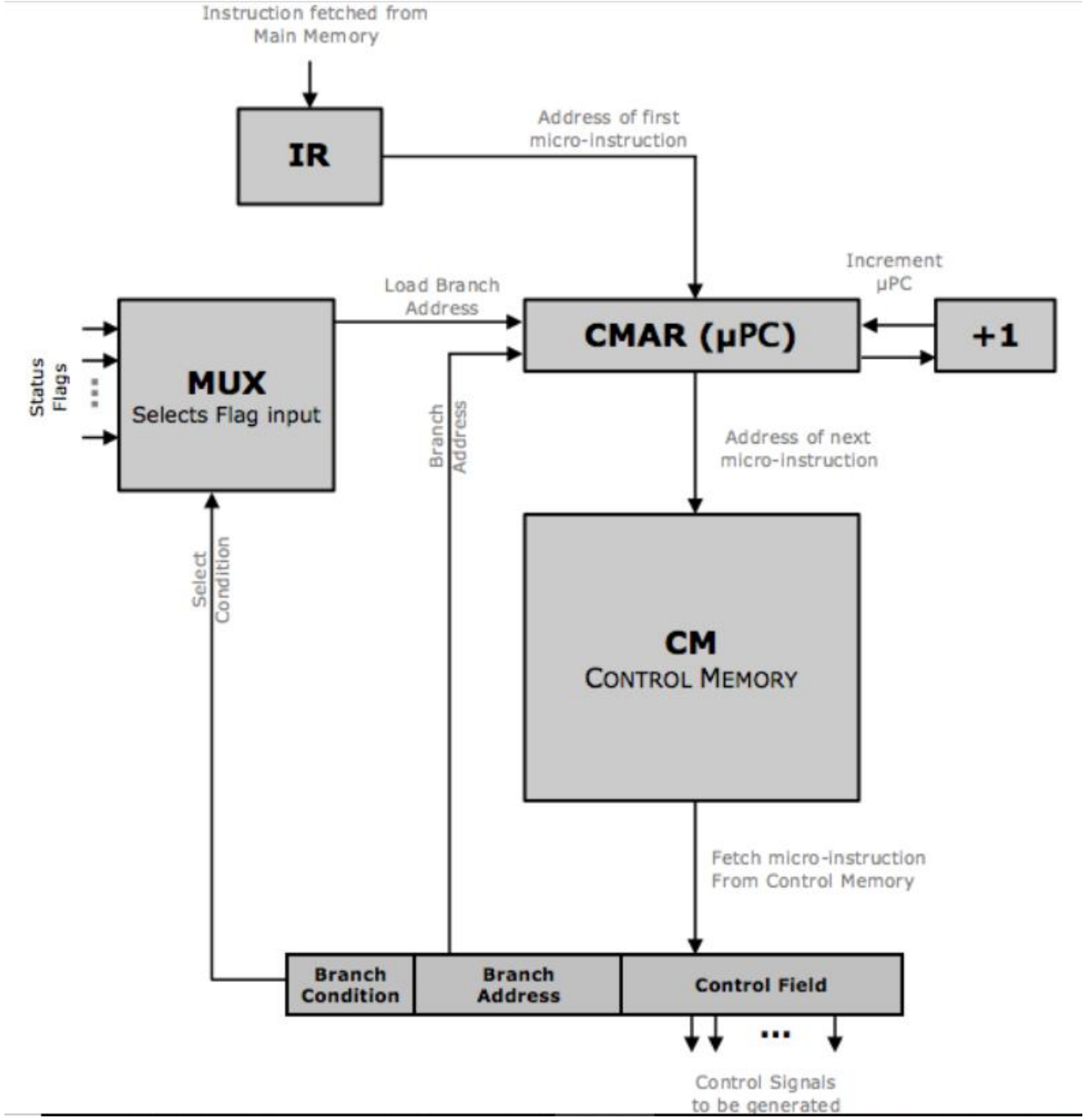
ADVANTAGES

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 be easily **avoided** by proper **micro-instruction sequencing**.

Actual Microprogrammed control unit



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

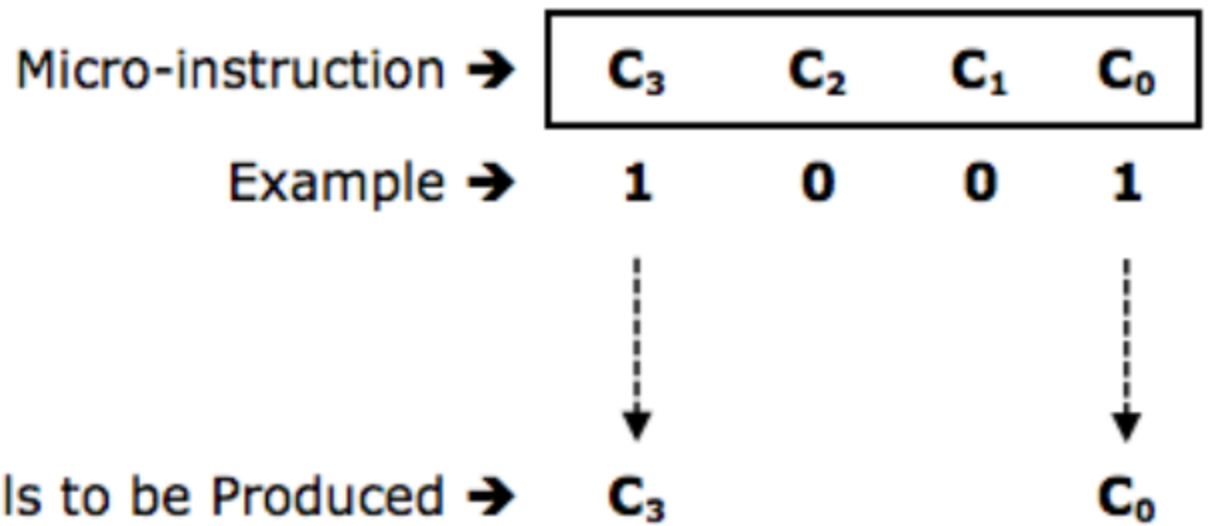
1) **Control memory** has to be present **inside** the processor, **increasing its size.**
2) This also **increases the cost** of the processor.

Micro-instruction Format

The main part of the micro-instruction is its control field.
It determines the control signals to be produced.
It can be of two different formats: Horizontal or Vertical.

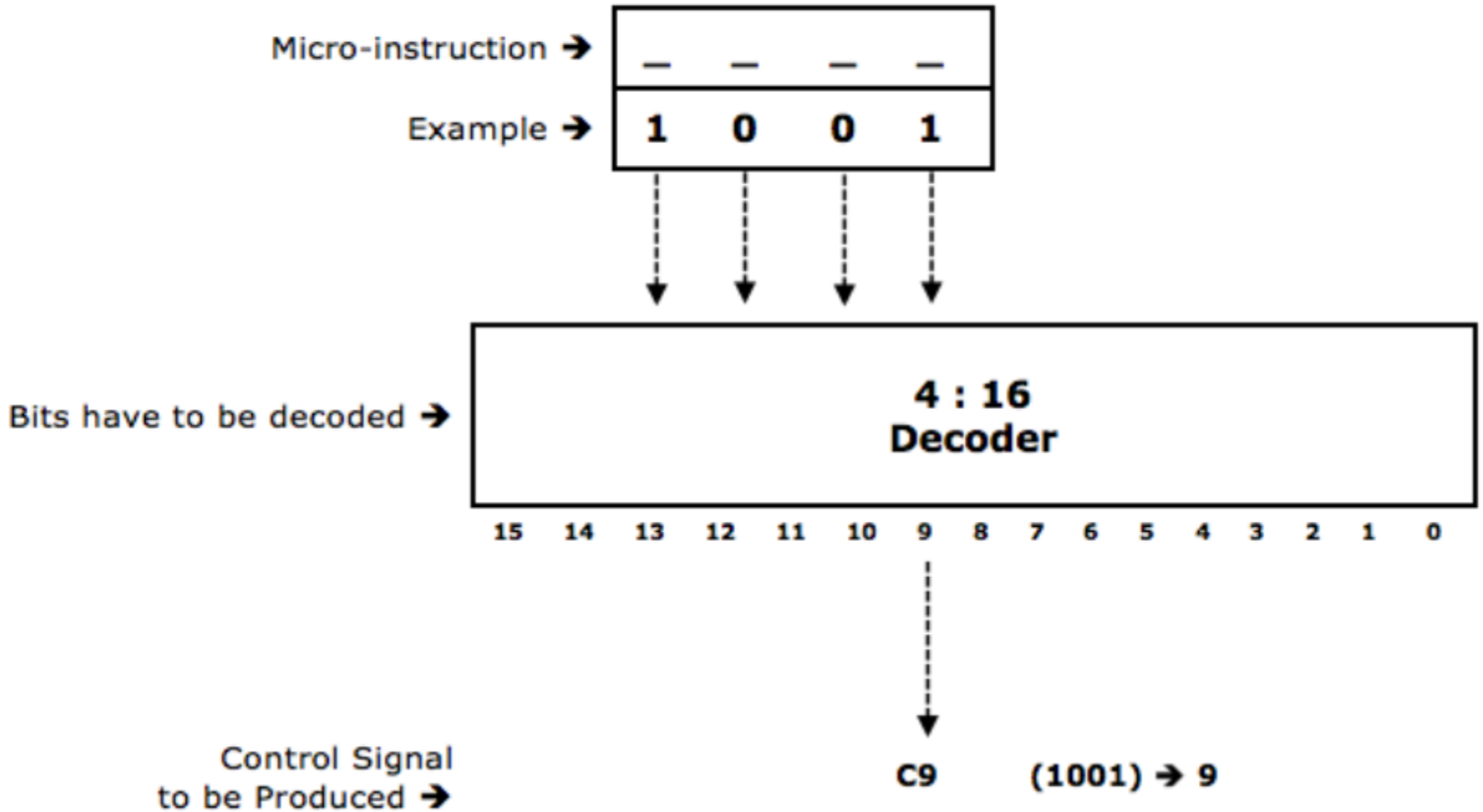
1) Horizontal Micro-Instruction

Here every bit of the micro-instruction corresponds to a control signal.
Whichever bit is "1", that particular control signal will be produced by the micro-instruction.



2) VERTICAL MICRO-INSTRUCTION

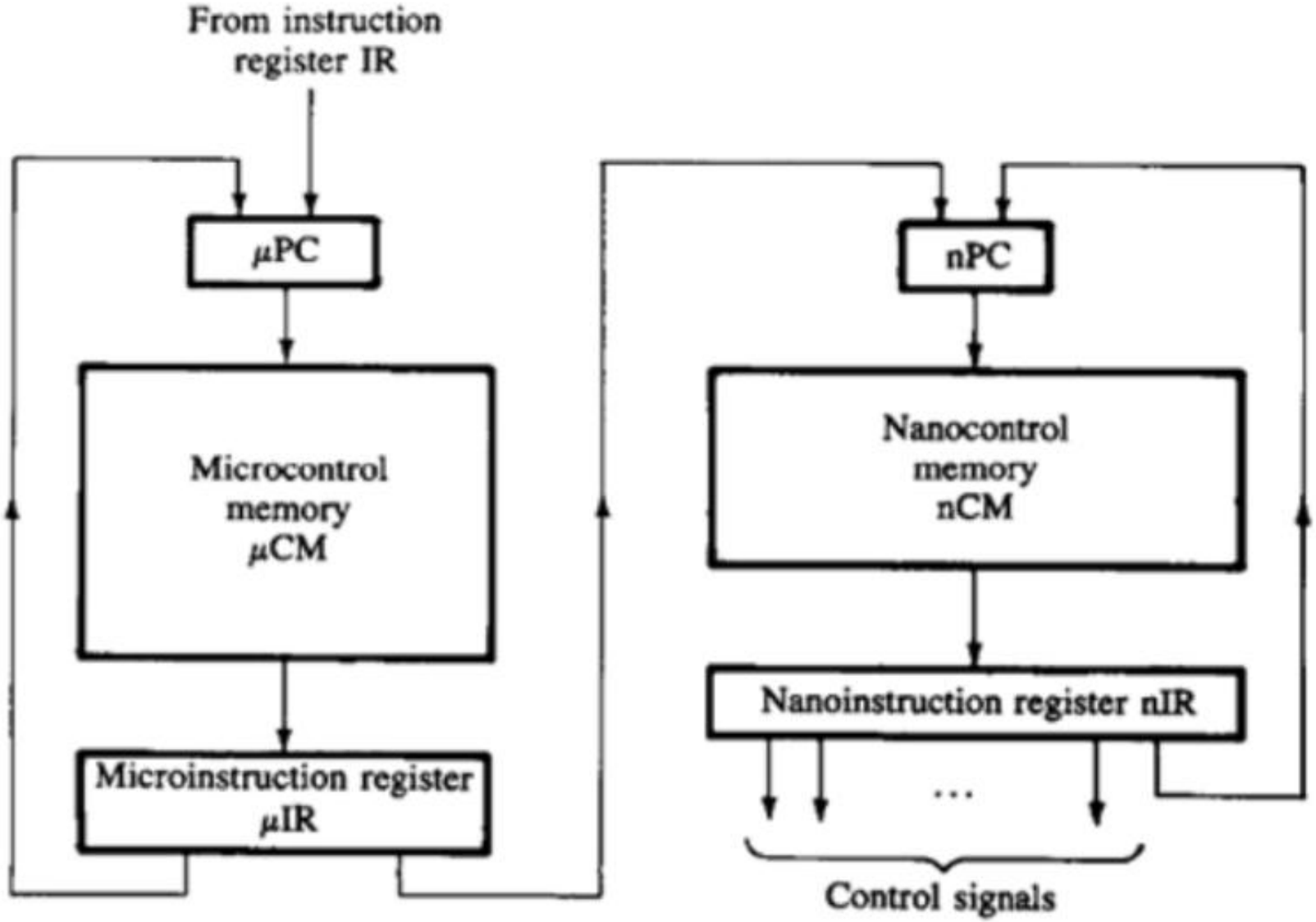
Here bits of the micro-instruction have to be decoded.
The decoded output decides the control signal to be produced.



	HORIZONTAL MICRO-INSTRUCTION	VERTICAL MICRO-INSTRUCTION
1	Every bit of the micro-instruction corresponds to a control signal.	Bits of the micro-instruction have to be decoded to produce control signals.
2	Does not require a decoder.	Needs a decoder.
3	N bits in the micro-instruction will totally produce N control signals.	N bits in the micro-instruction will totally produce 2^N control signals.
4	Multiple control signals can be produced by one micro-instruction.	Only one control signal can be produced by one micro-instruction.
5	As the control signals increase, the micro-instruction grows wider. Hence the Control Memory grows Horizontally.	To produce more control signals, more number of micro-instructions are needed. Hence the Control Memory grows Vertically.
6	Executes faster as no decoding needed.	Executes slower as decoding is needed.
7	Micro-instruction are very wide. Hence Control memory is large.	Micro-instruction are much narrower. Hence Control memory is small.
8	Circuit is simpler as a decoder is not needed.	Circuit is more complex as a decoder is needed.

NANO-PROGRAMMING

1) **Horizontal** µ-instructions can produce **multiple control signals** simultaneously, but are **very wide**.
2) This makes the Control Memory **very large in size**.
3) **Vertical** micro-instructions are **narrow**, **but on decoding** can **produce** only **one control signal**.
4) This makes the Control Memory **small** but the execution is **slow**.
5) Hence a **combination of both techniques** is needed called **Nano-Programming**.
6) Here we have a **two level control memory**.
7) The instruction is fetched from the **main memory into IR**.
8) Using its **opcode** we load **address of its first micro-instruction** into **µPC**,
9) Using this address we **fetch the micro-instruction** from µ-Control Memory (**µCM**) **into µIR**.
10) This is in **vertical form** and has to be decoded.
11) The decoded output **loads a new address** in a Nano program counter (**nPC**).
12) Using this address we **fetch the Nano-instruction** from Nano-Control Memory (**nCM**) into **nIR**.
13) This is in **horizontal form** and can **directly generate control signals**.
14) Such a combination **gives advantage of both techniques**.
15) The size of the Control Memory is **small** as µ-instructions are **Vertical**.
16) Multiple control signals can be **produced simultaneously** as Nano-instructions are **Horizontal**.


From instruction register IR
μPC
Microcontrol memory μCM
Microinstruction register μIR
nPC
Nanocontrol memory nCM
Nanoinstruction register nIR
...
Control signals