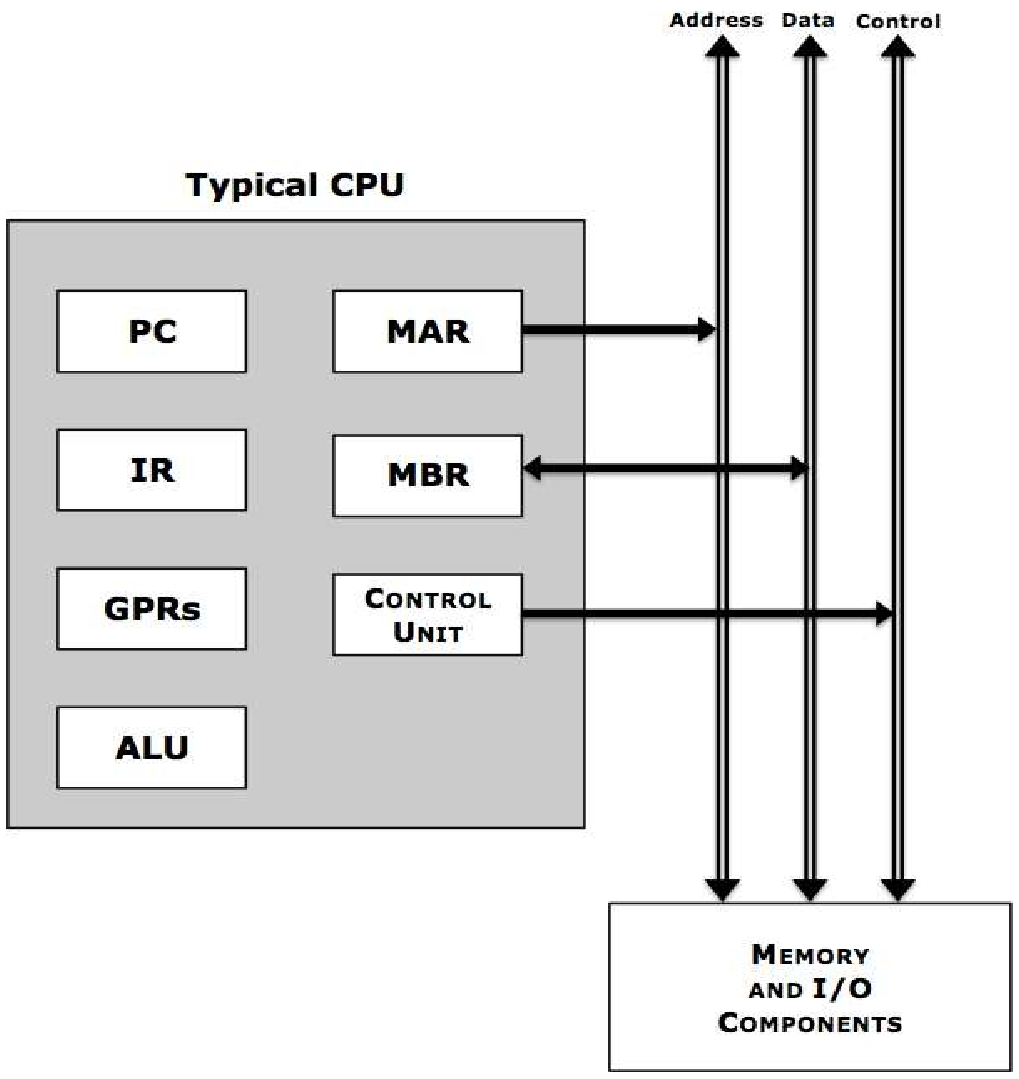
MICRO-OPERATIONS & CONTROL SIGNALS

- A **Program** is a **set of Instructions**.
- An **Instruction**, requires a **set of small** operations called **Micro-Operations**.
- A **Micro-Operation** is a **finite activity performed** by the processor **in one clock cycle**. One clockcycle is also called one **T-state** (Transition state).
- **One Micro-Operation** requires **One T-state**.
- Several **Independent** Micro-Operations can be performed in the same T-state.
- **Control unit generates control signals** to perform these very Micro-Operations.
- To understand Control Units, we must first clearly understand **Micro-Operations**.



Micro-Operations for Instruction Fetching

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 from MBR*
	PC 🡐 PC + 1	*PC gets incremented*

As we can see in the above table, two Micro-Operations can take place in the same T-state i.e.

IR🡐 MBR and PC🡐 PC + 1

This is because they are completely independent of each other.
In fact, PC becoming PC + 1 can also be performed in the 2nd T-state while the instruction is being fetched from the memory through the data bus into MBR.
This is shown below.

Micro-Operations for Instruction Fetching (Alternate Method)

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*
	PC 🡐 PC + 1	*PC gets incremented*
T3:	IR 🡐 MBR	*IR gets instruction from MBR*

Please Note

PC 🡐 PC + 1 cannot take place in the 1st T-state.
That's because, in the 1st T-state, PC is providing the address on the address bus, through MAR. If at the same time PC gets incremented, then the incremented address will be put on the address bus.

Now that we know how an instruction is fetched, we can proceed further and learn Micro-Operations for various instructions, of different Addressing Modes.

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 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.*

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*

Execution of Complete Instruction

We have discussed about four different types of basic operations:

- Fetch information from memory to CPU
- Store information to CPU register to memory
- Transfer of data between CPU registers.
- Perform arithmetic or logic operation and store the result in CPU registers.

To execute a complete instruction we need to take help of these basic operations and we need to execute these operation in some particular order.

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

Execution of this instruction requires the following action :

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

Steps	Actions
1.	PC_{out}, MAR_{in}, Read, Clear Y, Set carry -in to ALU, Add, Z_{in}
2.	Z_{out}, PC_{in}, Wait For MFC
3.	MDR_{out}, Ir_{in}
4.	Address-field- of-IR_{out}, MAR_{in}, Read
5.	$R1_{out}$, Y_{in}, Wait for MFC
6.	MDR_{out}, Add, Z_{in}
7.	Z_{out}, $R1_{in}$
8.	END

instruction execution proceeds as follows:

In Step1:
The instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a read request to memory.

To perform this task first of all the contents of PC have to be brought to internal bus and then it is loaded to MAR.To perform this task control circuit has to generate the PC_{out} signal and MAR_{in}signal.

After issuing the read signal, CPU has to wait for some time to get the MFC signal. During that time PC is updated by 1 through the use of the ALU. This is accomplished by setting one of the inputs to the ALU (Register Y) to 0 and the other input is available in bus which is current value of PC.

At the same time, the carry-in to the ALU is set to 1 and an add operation is specified.

In Step 2:

The updated value is moved from register Z back into the PC. Step 2 is initiated immediately after issuing the memory Read request without waiting for completion of memory function. This is possible, because step 2 does not use the memory bus and its execution does not depend on the memory read operation.

In Step 3:

Step3 has been delayed until the MFC is received. Once MFC is received, the word fetched from the memory is transferred to IR (Instruction Register), Because it is an instruction. Step 1 through 3 constitute the instruction fetch phase of the control sequence.

The instruction fetch portion is same for all instructions. Next step inwards, instruction execution phase takes place.

As soon as the IR is loaded with instruction, the instruction decoding circuits interprets its contents. This enables the control circuitry to choose the appropriate signals for the remainder of the control sequence, step 4 to 8, which we referred to as the execution phase. To design the control sequence of execution phase, it is needed to have the knowledge of the internal structure and instruction format of the PU. Secondly , the length of instruction phase is different for different instruction.

In this example , we have assumed the following instruction format :

opcode	M	R

In Step 4

The contents of the PC are transferred to register Y.

In Step 5 :

The destination field of IR, which contains the address of the register R1, is used to transfer the contents of register R1 to register Y and wait for Memory function Complete. When the read operation is completed, the memory operand is available in MDR.

In Step 6 :

The result of addition operation is performed in this step.

In Step 7:

The result of addition operation is transfered from temporary register **Z** to the destination register **R1** in this step.

In step 8 :

It indicates the end of the execution of the instruction by generating End signal. This indicates completion of execution of the current instruction and causes a new fetch cycle to be started by going back to step 1.

Execution of a Complete Instruction

Add (R3), R1

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Figure 7.6. Control sequence for execution of the instruction Add (R3),R1.

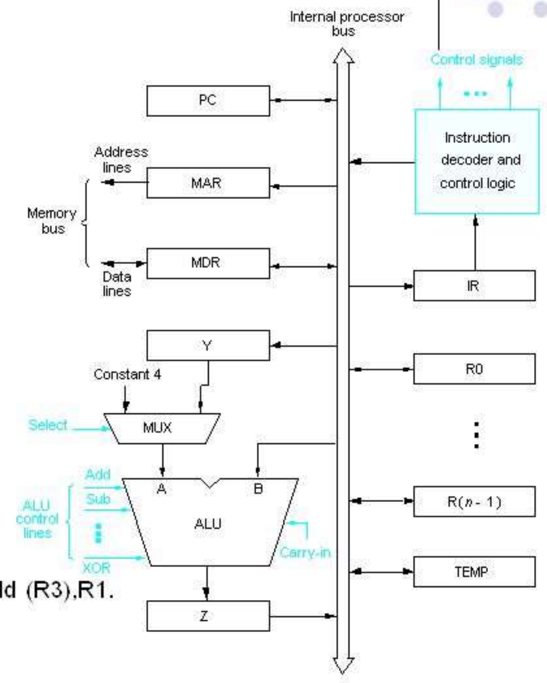
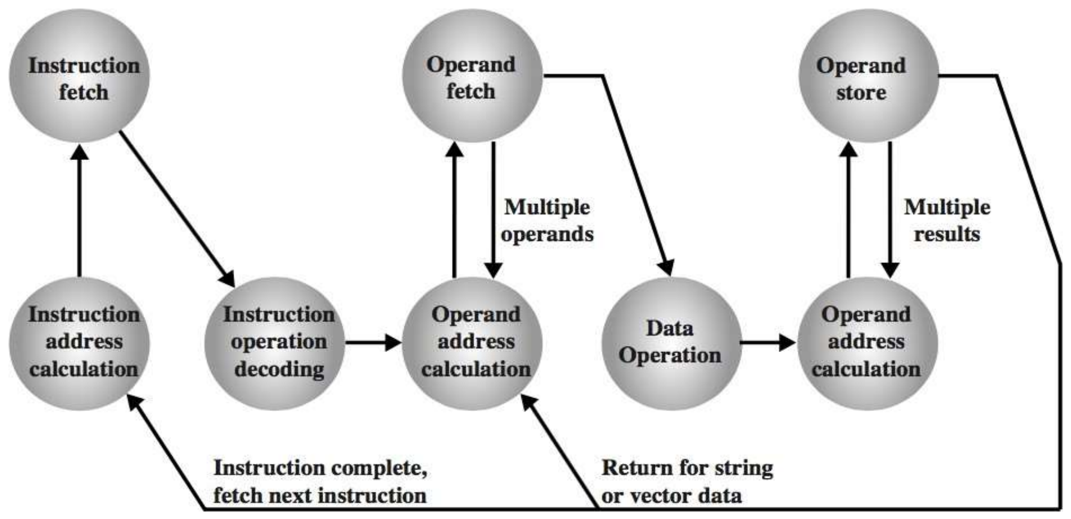


Figure 7.1. Single-bus organization of the datapath inside a proc

INSTRUCTION CYCLE

An instruction cycle is the complete process of fetching, decoding and executing the instruction.


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

1) **PC gives the address to fetch** an instruction from the memory.
2) Once fetched, the **instruction opcode is decoded**.
3) This **identifies**, if there are **any operands to be fetched** from the memory.
4) The **operand address is calculated.**
5) **Operands are fetched** from the memory.
6) Now the **data operation is performed** on the operands, and a **result is generated**.
7) If the result has to be stored in a **register**, the **instruction ends** here.
8) If the **destination is memory**, then first the **destination address has to be calculated**.
9) The result is then **stored in the memory**.
10) Now the current instruction has **been executed**.
11) **Side by side PC is incremented** to calculate address of the next instruction.
12) The above instruction cycle then **repeats for further instructions**.

HARDWIRED CONTROL UNIT

- Here control signals are produced by hardware.
- There are three types of Hardwired Control Units

STATE TABLE METHOD

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 everyinstruction.
5) 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	$Z_{M,1}$	$Z_{M,2}$	…	$Z_{M,N}$

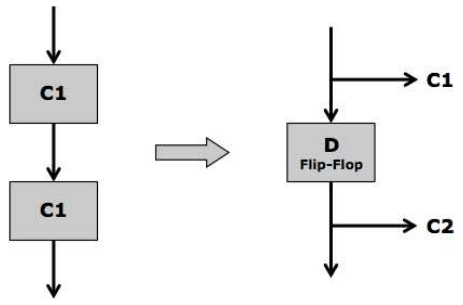
$Z_{1,1}$: Control Signal to be produced in T-state (T_1) of Instruction (I_1)

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

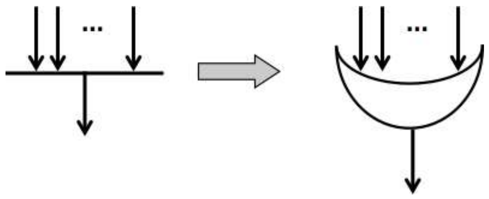
DRAWBACK:
As the number of instructions increase, the circuit becomes bigger and hence more complicated.
As a tabular approach is used, instead of a logical approach (flowchart), there are duplications of many circuit elements in various instructions.

DELAY ELEMENT METHOD

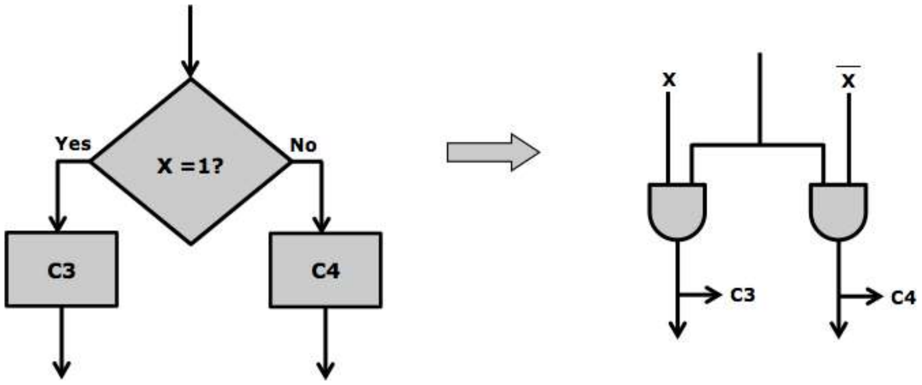
1) Here the behavior of the control unit is represented in the form of a flowchart.
2) Each step in the flowchart represents a control signal to be produced.
3) Once all steps of a particular instruction, are performed, the complete instruction gets executed.
4) Control signals perform Micro-Operations, 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**.



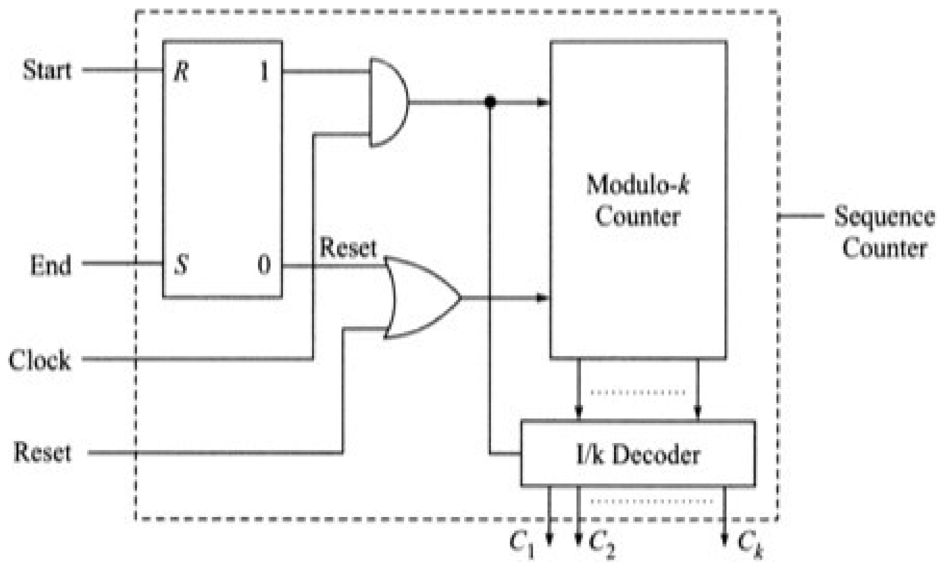
10) A **decision box** is replaced by a set of two complementing **AND gates**



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.

MICROPROGRAMMED CONTROL UNIT

WILKES' DESIGN FOR A MICROPROGRAMMED CONTROL UNIT

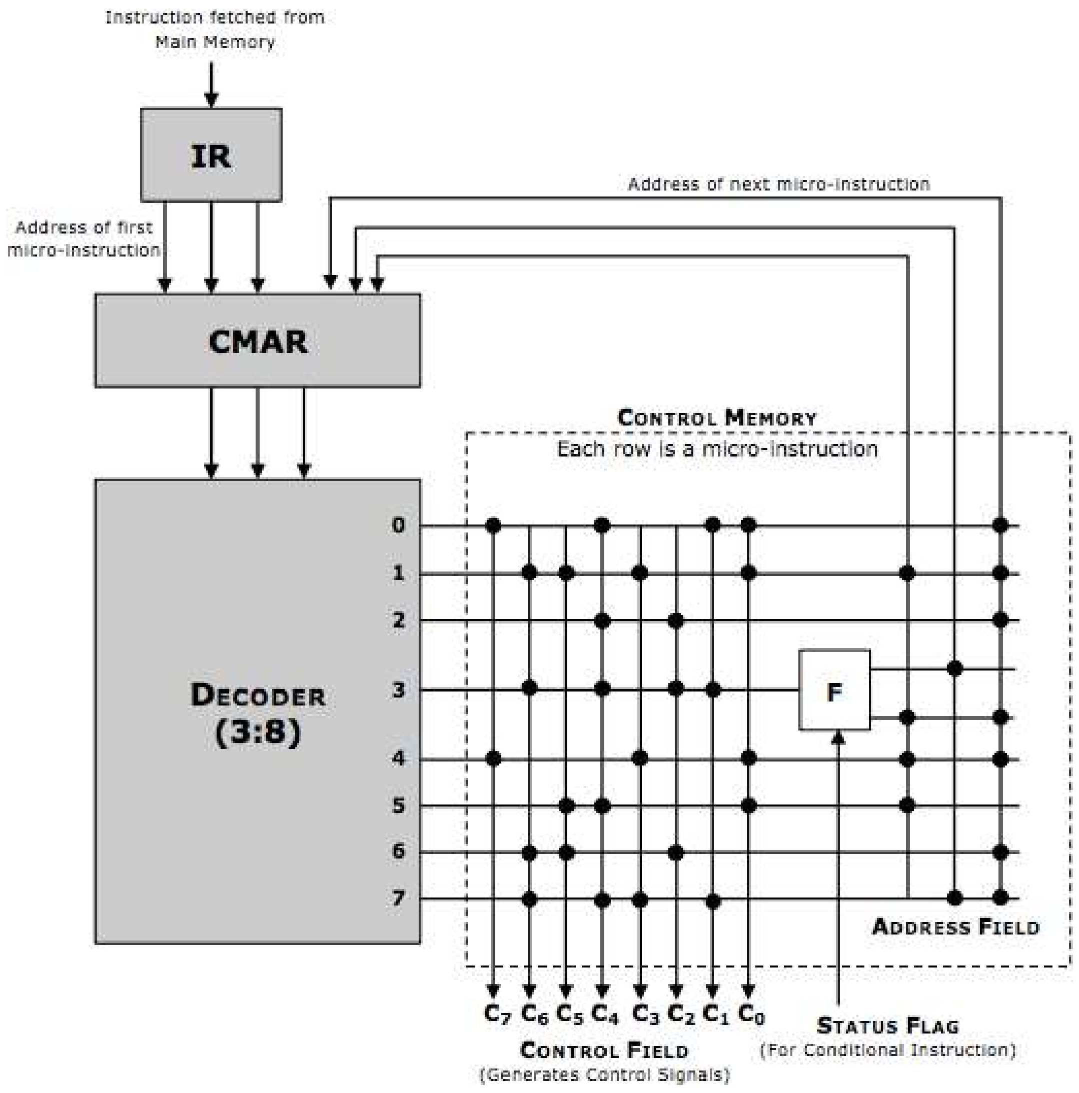
1) Microprogrammed Control Unit produces control signals by **software**, using **micro-instructions**
2) A program is a set of instructions.
3) An instruction requires a set of Micro-Operations.
4) **Micro-Operations are performed by control signals**.
5) Instead of generating these control signals 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 beeasily **avoided** by proper **micro-instruction sequencing**.


Instruction fetched from Main Memory
IR
Address of first micro-instruction
Address of next micro-instruction
CMAR
CONTROL MEMORY
Each row is a micro-instruction
DECODER (3:8)
0
1
2
3
4
5
6
7
F
ADDRESS FIELD
C7 C6 C5 C4 C3 C2 C1 C0
CONTROL FIELD
(Generates Control Signals)
STATUS FLAG
(For Conditional Instruction)