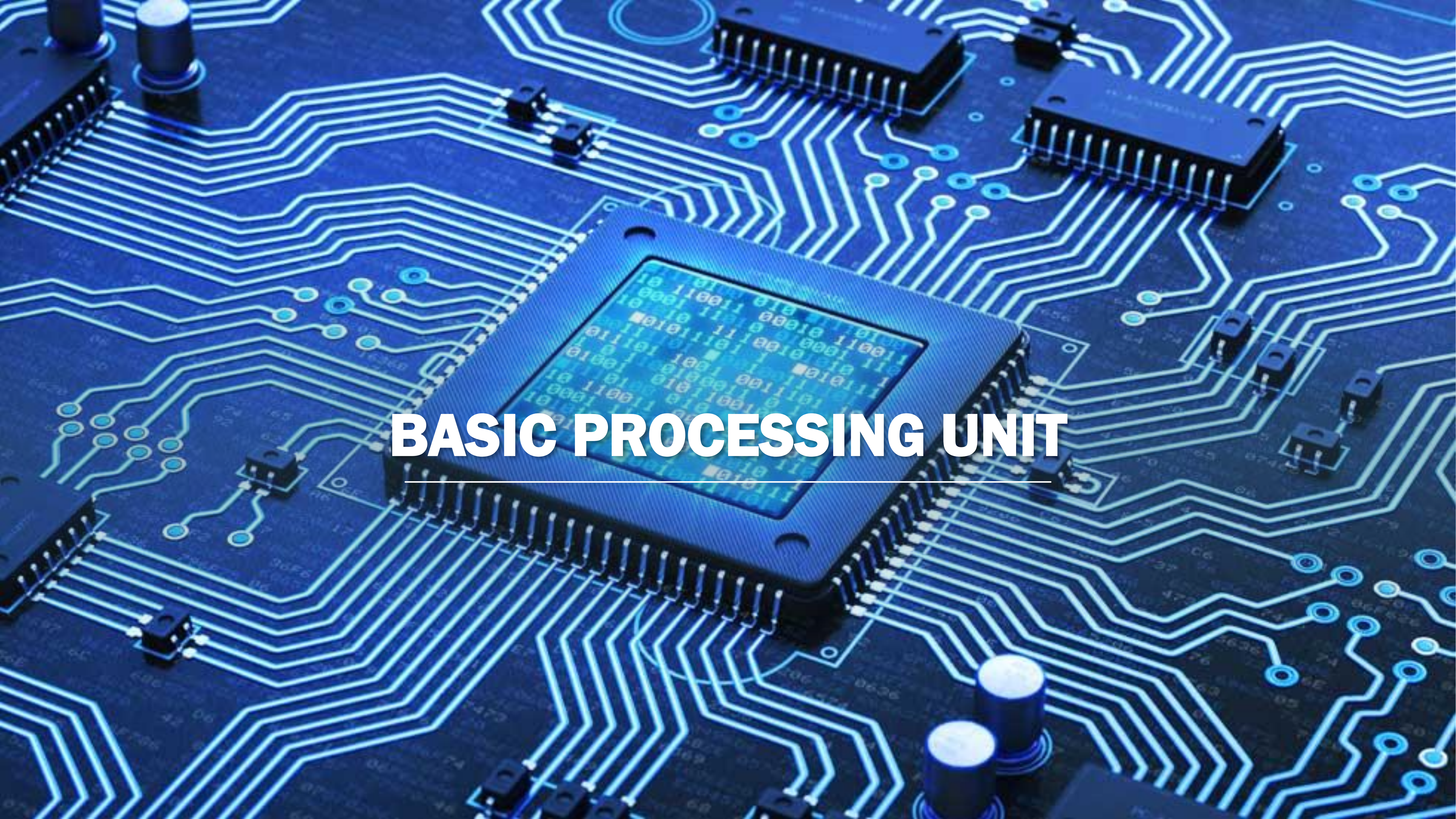


BASIC PROCESSING UNIT

WHAT WE HAVE STUDIED SO FAR?

- What is a computer and how it works?
- Different types of computers.
- Basic Functional Units of a computer.
- Performance of a computer.
- Number representation in computer – 1’s, 2’s complement systems
- What is an Instruction and what typical steps computer takes to execute an instruction?
- Assembly Programming

WHAT NOW?

- In this unit, we will mostly focus on the processing unit of a computer which executes machine instructions and coordinates the activities of other units.
- We will examine CPU's internal structure and how it performs the tasks of fetching, decoding, and executing instructions of a program.

EXECUTION STEPS

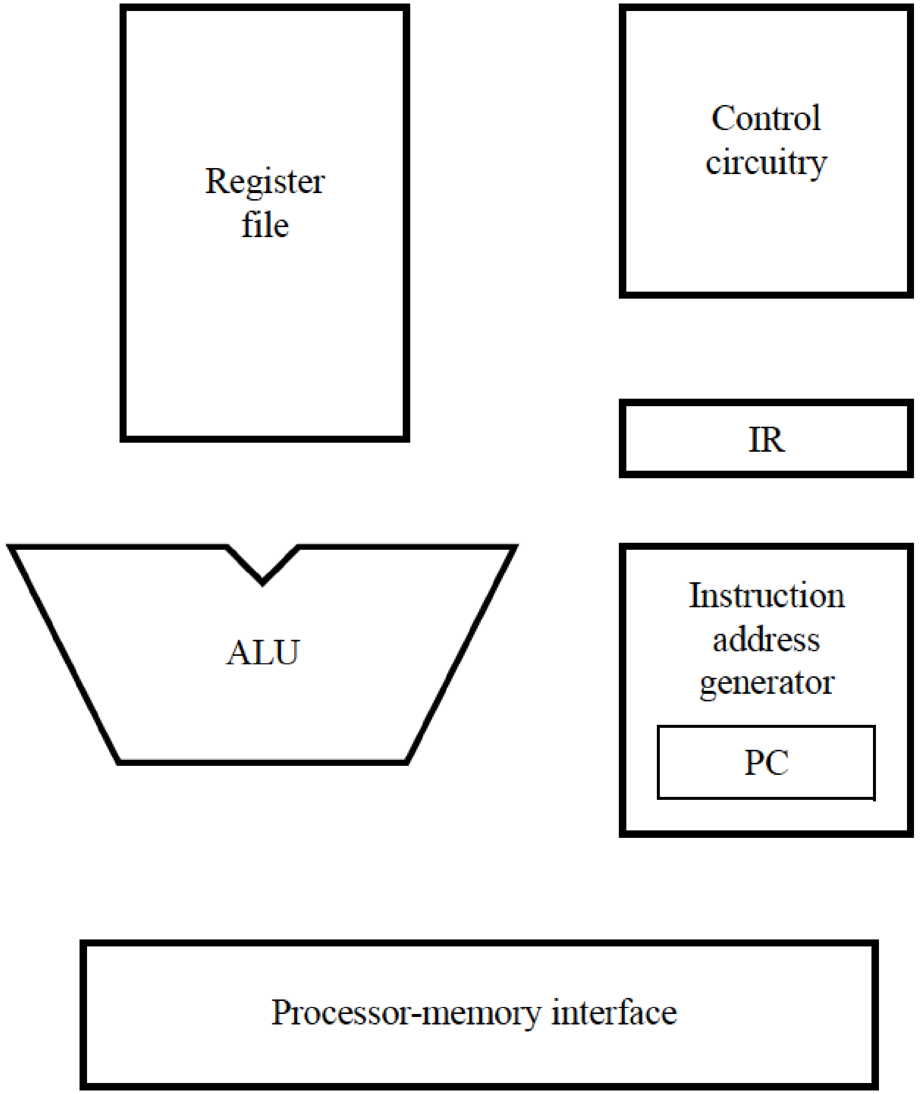
- To execute a program, the processor fetches one instruction at a time and perform the operation specified by the instruction.
- Instructions are fetched from the successive memory locations until a branch or jump instruction is encountered.
- The processor keeps track of the address of the memory location containing the next instruction to be fetched using the program counter, PC.
- After fetching an instruction, the contents of the PC are updated to point to the next instruction in sequence. A branch instruction may cause a different value to be loaded into the PC.
- After fetching the instruction, it's placed into another register called *instruction register,* IR, from where it is interpreted, or decoded by the processor's control circuitry. IR holds the register until its execution is completed.

EXECUTION STEPS

- Fetching an instruction and loading it into the IR is usually referred as *instruction fetch phase.*
- After placing instruction into IR, processor decodes the instruction which is referred as *instruction decode phase.*
- Performing the operation specified in the instruction is referred as *instruction execution phase.*

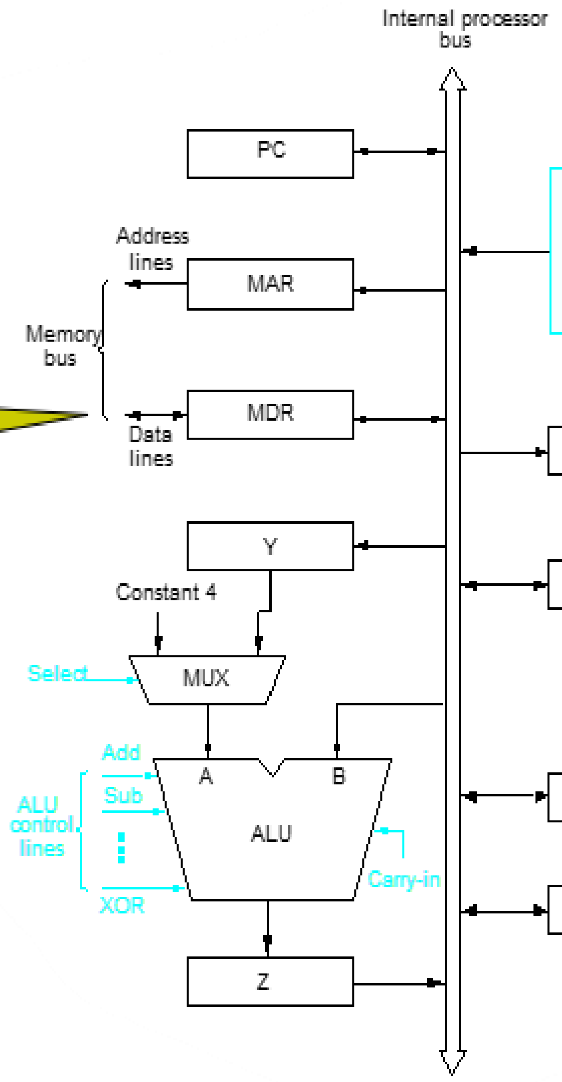
PROCESSOR'S BUILDING BLOCKS

- PC provides instruction address.
- Instruction is fetched into IR.
- Instruction address generator updates PC.
- Control circuitry interprets instruction and generates control signals to perform the actions needed.
- ALU performs the computation required in by instruction.
- These building blocks of CPU can be interconnected with each other in many ways.



SINGLE-BUS DATAPATH ORGANIZATION

MDR HAS TWO INPUTS AND TWO OUTPUTS



Datapath

SINGLE-BUS DATAPATH

- In single-bus datapath, Arithmetic and Logic Unit (ALU) and all the registers are interconnected via a single common bus. This bus is processor’s internal bus which is different from external bus which connects processor to the memory and I/O devices.
- The data and address lines of the external memory bus are connected to the internal processor bus via MDR and MAR respectively.
- Register MDR has two inputs and two outputs since data can be loaded into MDR from either memory bus or from processor bus.
- The control lines of the memory bus are connected to the instruction decoder and control logic unit. This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus.

SINGLE-BUS DATAPATH

- The number and use of the processor registers $R_0, \ldots, R_{n-1}$ vary considerably from one processor to another. Registers are provided for general-purpose use by a programmer but there are some special registers also like *stack pointer (SP) or program counter (PC).*
- There may be some extra registers like Y, Z and TEMP which are not visible to a programmer. These registers are used as a temporary storage by processor during execution of some instructions. These registers are never referenced by any instruction of a program.
- The multiplexer MUX selects either output of register Y or a constant value 4 to be provided as input A of the ALU. The constant 4 is used to increment the contents of the program counter. The control input *select* of MUX can be referred as *select4* for selecting 4 or *selectY* for selecting Y.

SINGLE-BUS DATAPATH

- As an instruction execution progresses, data are transferred from one register to another, often passing through the ALU to perform some arithmetic or logic operations.
- The instruction decoder and control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register.
- The decoder generates the control signals needed to select the registers involved and direct the transfer of data.
- The registers, the ALU, and the interconnecting bus are collectively referred to as the *datapath.*

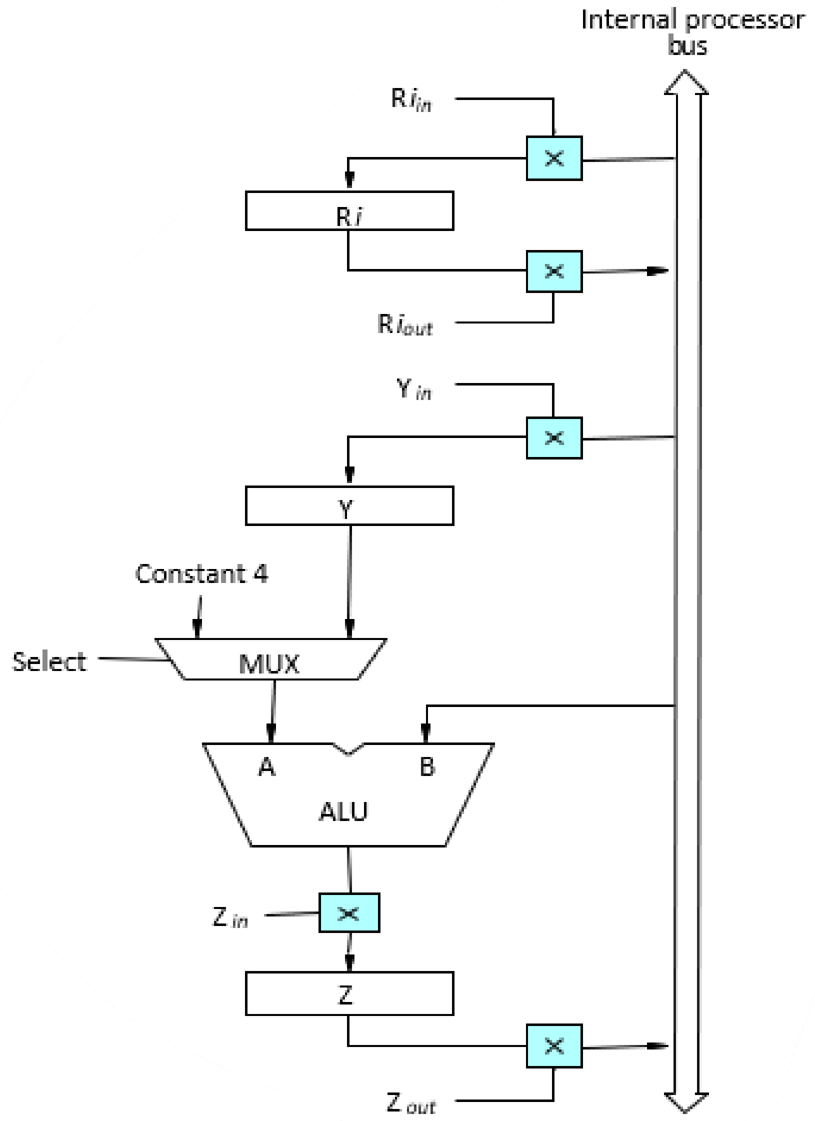
DATA TRANSFER IN REGISTERS

- Instruction execution involves a sequence of steps in which data are transferred from one register to another.
- For each register, two control signals are used to place the contents of that register on the bus or to load the data on the bus into the register.
- The input and output ports of a register Ri are connected to the bus via switches controlled by the signals Ri_{in} and Ri_{out} , respectively.
- When Ri_{in} is set to 1, the data on the bus are loaded into Ri and when Ri_{out} is set to 1, the contents of register Ri is loaded on the bus.

DATA TRANSFER IN REGISTERS

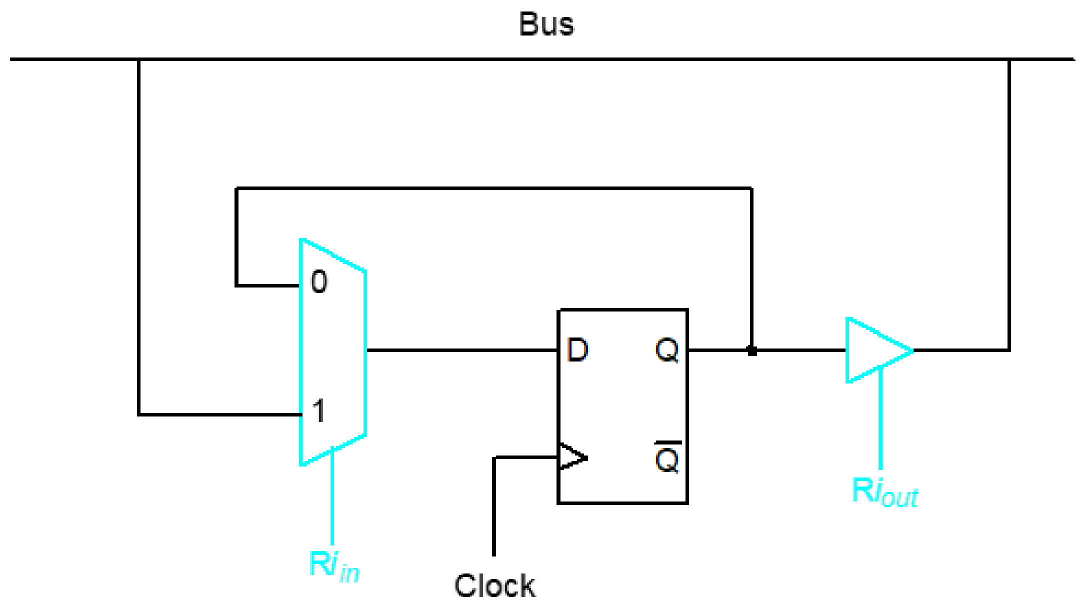
Suppose we want to transfer the contents of register R1 to register R4. This can be done as following–

- Enable the output port of register R1 by setting $R1_{out}$ to 1. This will copy the contents of R1 on the processor bus.
- Enable the input of register R4 by setting $R4_{in}$ to 1. This loads data from the processor bus into register R4.

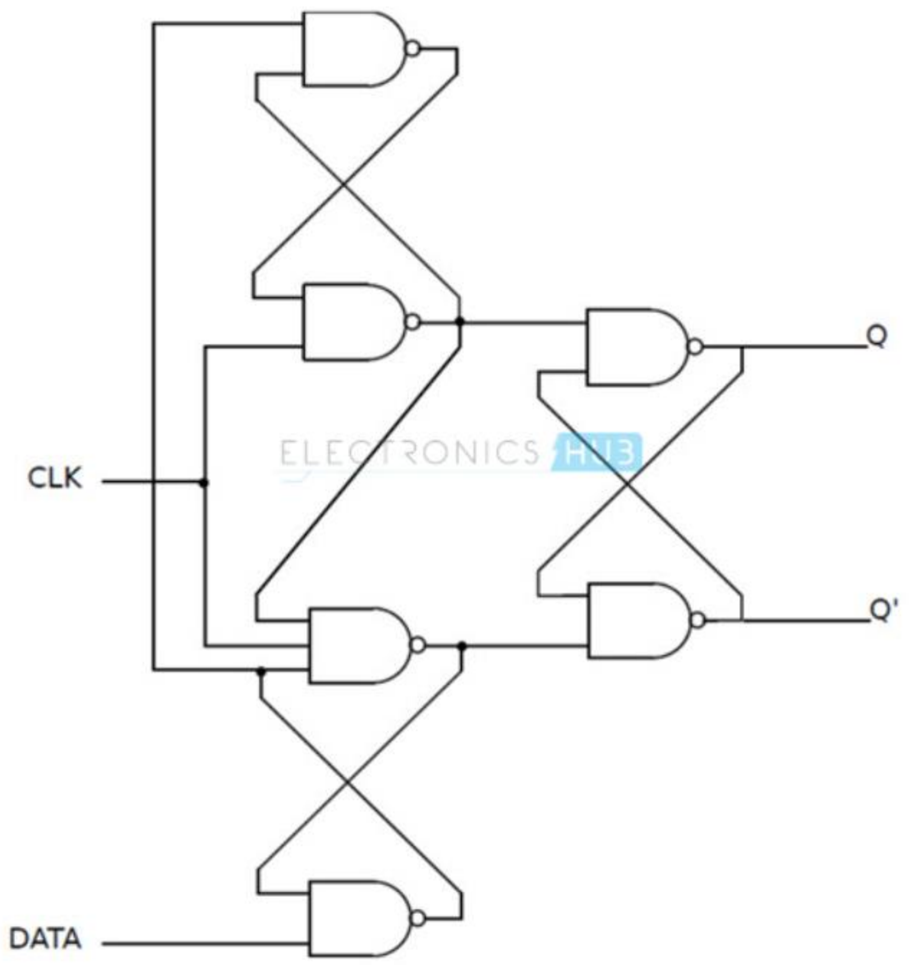


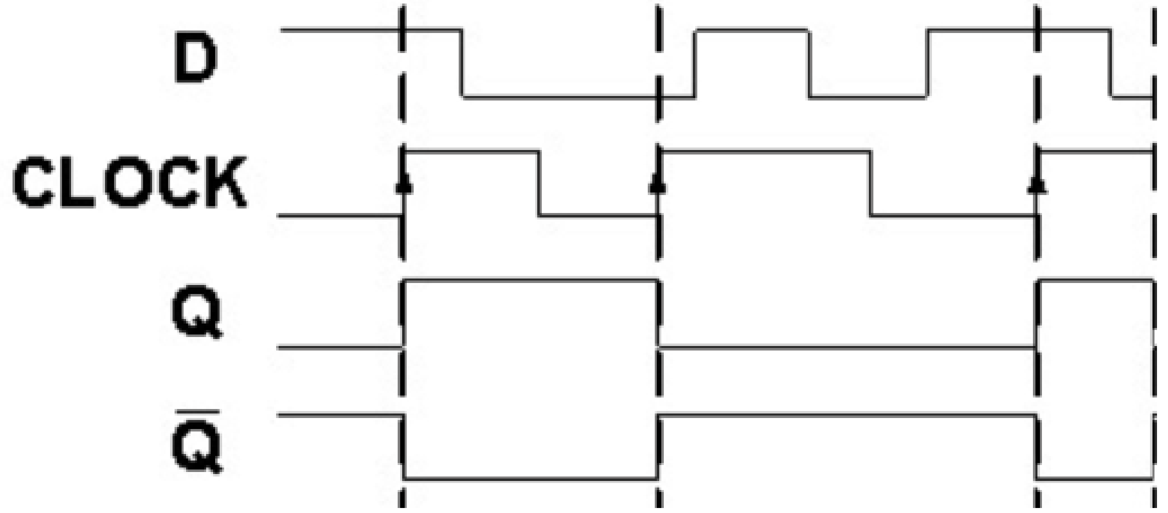
DATA TRANSFER IN REGISTERS

- All the operations and data transfers within the processor take place within time periods defined by the *processor clock.*
- The control signals that manages a particular transfer are transferred at the start of the clock cycle.
- The registers consists of edge-triggered flip-flops. Hence data will be transferred at the active edge of clock.



DESIGN OF REGISTER



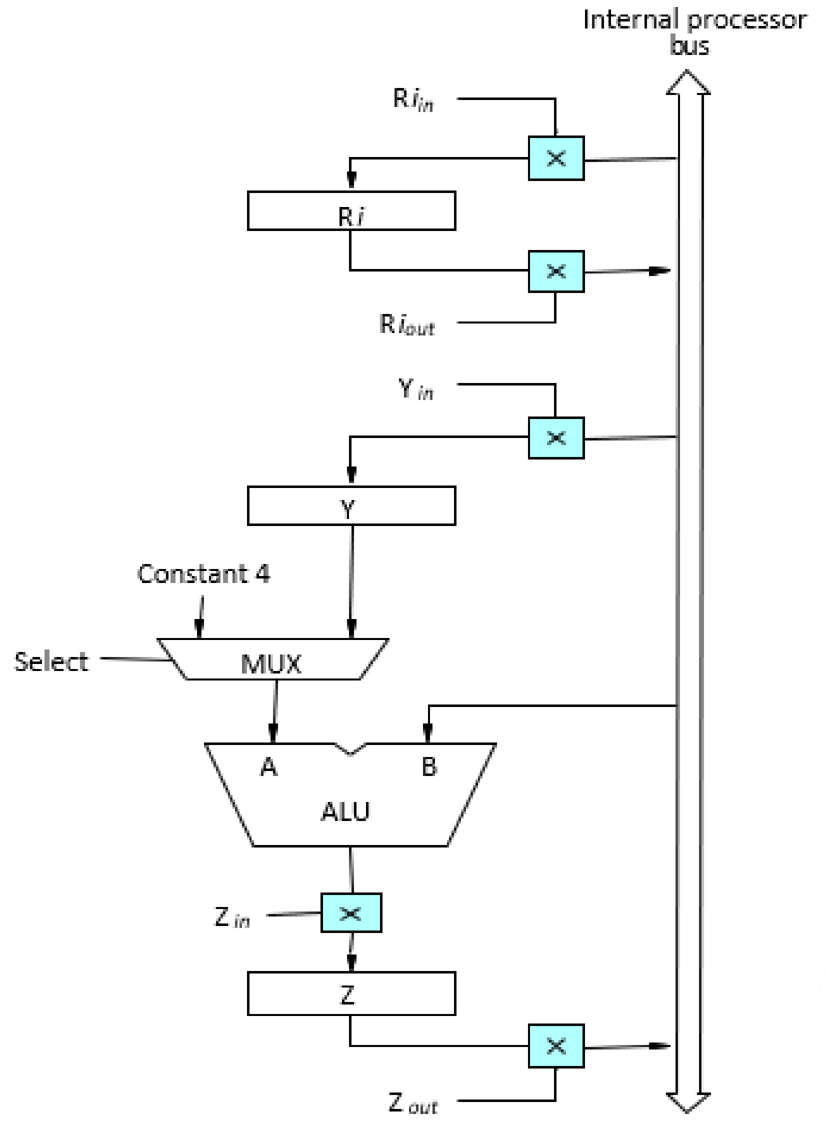


DATA TRANSFER IN REGISTERS

- Apart from transferring data at rising or active edge of clock, we can also use both rising and falling edges of clock for timing data transfer.
- When edge-triggered flip-flops are not used, two or more clock signals may be needed to guarantee proper data transfer. This is called *multiphase clocking.*

PERFORMING AN ARITHMETIC OPERATION

- The ALU is a combinational circuit that has no internal storage. It performs arithmetic and logic operations on two operands applied to its input ports.
- In single bus CPU organization, one input port of ALU is connected to multiplexer and other is directly connected to the bus. And the result is temporarily stored into register Z.

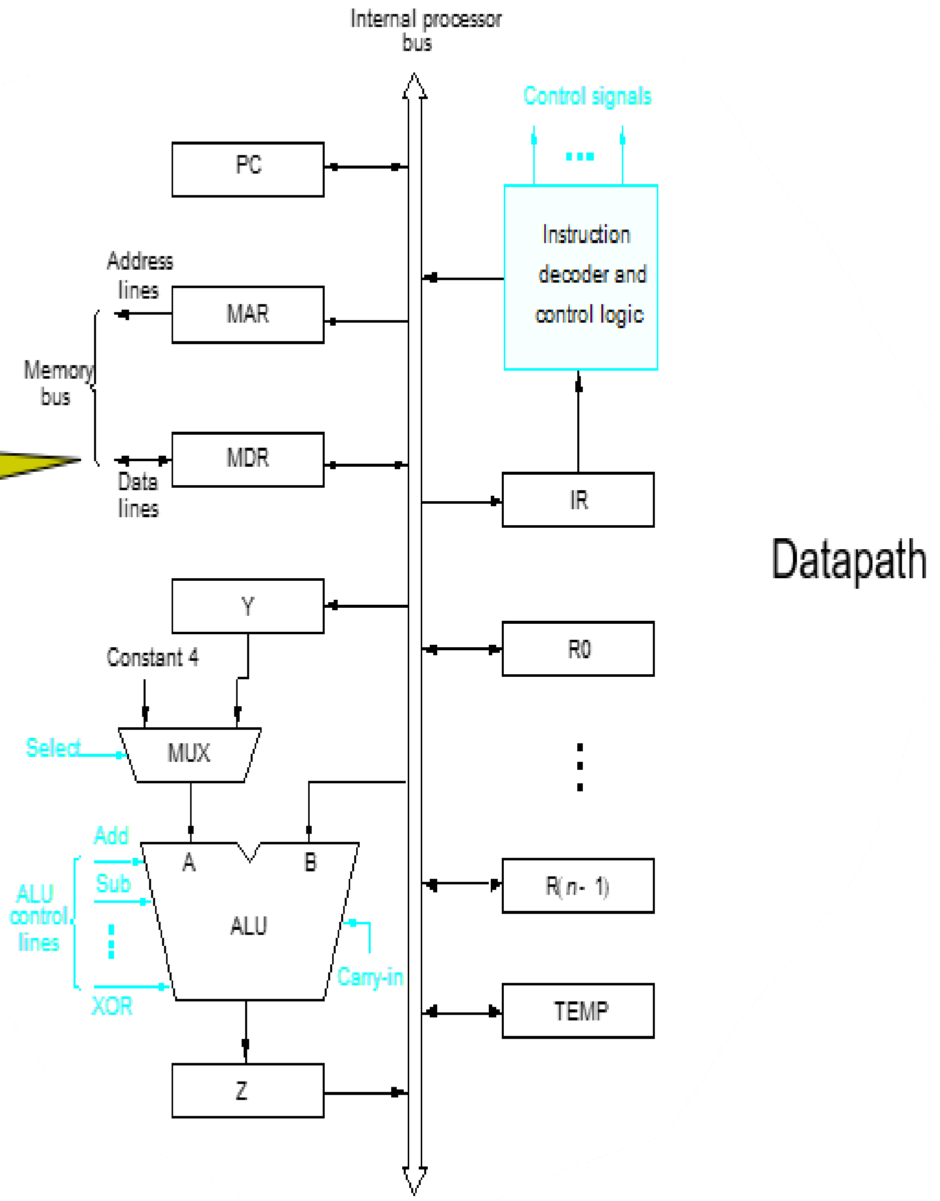


PERFORMING AN ARITHMETIC OPERATION

- A sequence of operations or control signals to add contents of register R1 and register R2 and store results in R3 can be given as follows –

1. $R1_{out}, Y_{in}$
2. $R2_{out},\ SelectY,\ ADD,\ Z_{in}$
3. $Z_{out}, R3_{in}$

MDR HAS TWO INPUTS AND TWO OUTPUTS



Datapath

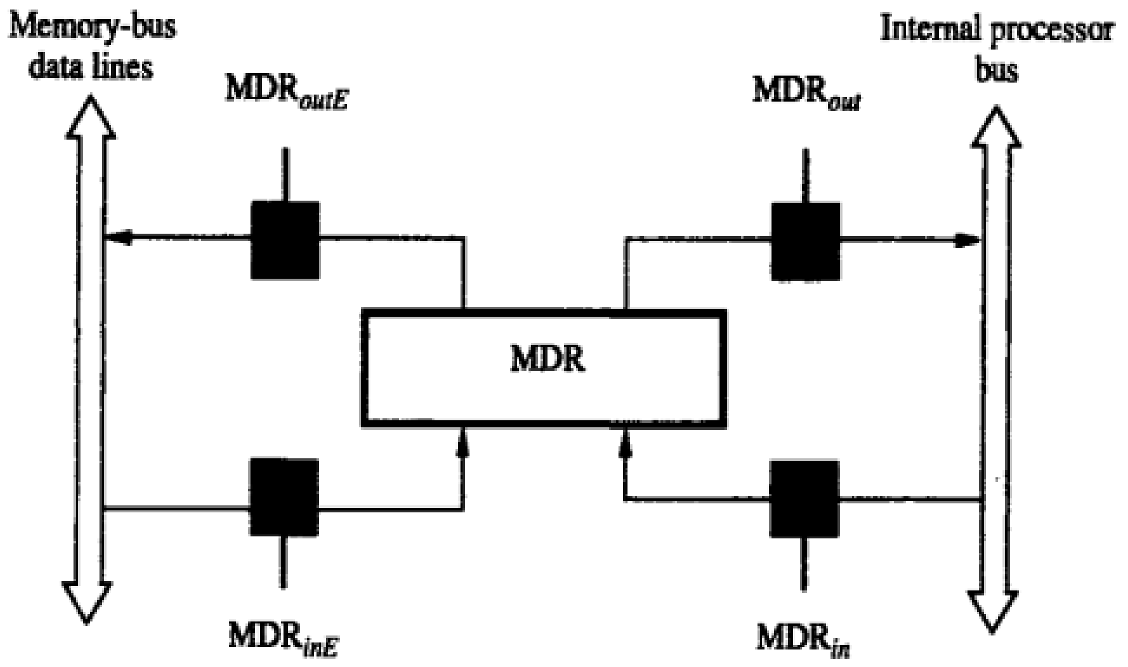
PERFORMING ARITHMETIC OPERATION

- The signals whose names are given in any step are activated for the duration of clock cycle corresponding to that step. And all other signals are inactive.
- In step 1, the output of register R1 and the input of register Y are enabled, causing the contents of R1 to be transferred over the bus to Y.
- In step 2, the multiplexer's *Select* signal is set to *SelectY,* causing the multiplexer to gate the contents of register Y to input A of the ALU. At the same time, the contents of register R2 are copied into bus which can be read by input B of ALU.
- The *ADD* control signal will tell ALU to perform ADD operation on its inputs and the result will be loaded into register Z as *Zin* is set to 1.
- In step 3, the contents of register Z are transferred to the destination register R3.

1. $R1_{out}, Y_{in}$
2. $R2_{out}, \ SelectY, \ ADD, \ Z_{in}$
3. $Z_{out}, R3_{in}$

FETCHING A WORD FROM MEMORY

- To fetch a word of information from memory, the processor transfers the address of the memory location into *MAR* register and uses control lines of memory bus to indicate that a read operation needs to be performed.
- The data read from memory is stored in the MDR register first then it can be transferred to desired register.
- MDR has four control signals $MDR_{in,}\ and\ MDR_{out}$ for the connection to internal bus. $MDR_{inE}\ and\ MDR_{outE}$ for the connection to memory bus.



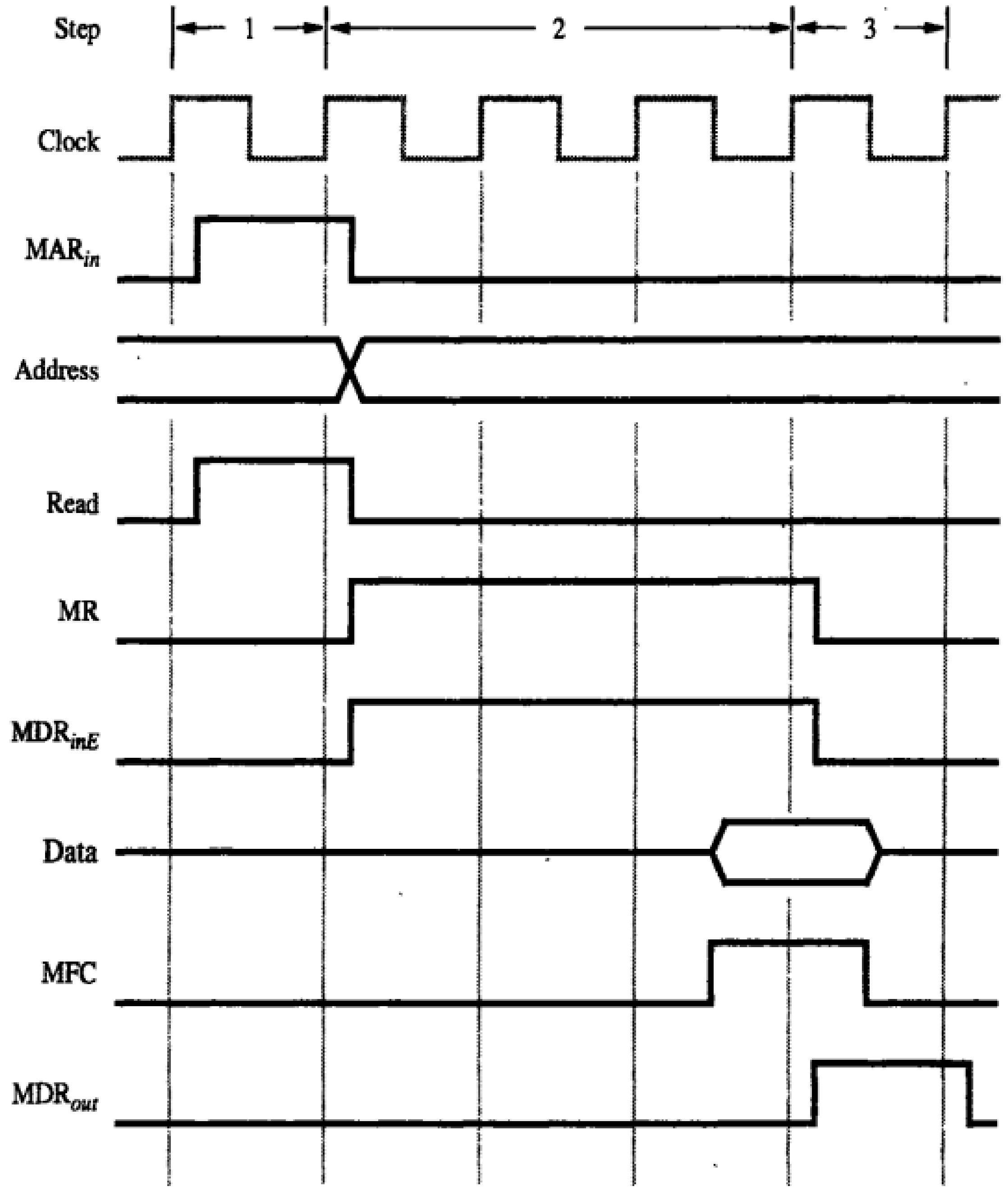
FETCHING A WORD FROM MEMORY

- During the memory read and write operations, the timing of internal processor operations must be coordinated with the response of the addressed device on the memory bus.
- The processor completes one internal data transfer in one clock cycle but the speed of operation of the memory device can vary depending upon the device.
- To accommodate the variability in response time, a control signal called *Memory-Function-Completed (MFC)* is used.
- Whenever memory read is completed and data is ready on memory bus line then memory device sets signal *MFC* to 1 to let processor know.

FETCHING A WORD FROM MEMORY

- We can write control signals for instruction **LOAD R1, (R2)** as follows –
  1. $R2_{out},\ MAR_{in},\ READ$
  2. $MDR_{inE},\ WMFC$ here WMFC is control signals which denotes the wait for MFC signal
  3. $MDR_{out},\ R1_{in}$

FETCHING A WORD FROM MEMORY – TIMING DIAGRAM



EXERCISE: STORING A WORD IN MEMORY

- Write control signals for the instruction **STORE R2, (R1)** –
  1. $R1_{out},\ MAR_{in}$
  2. $R2_{out},\ MDR_{in},\ \ Write$
  3. $MDR_{outE},\ WMFC$ here WMFC is control signals which denotes the wait for MFC signal

EXECUTION OF A COMPLETE INSTRUCTION

- Write control sequence for the complete execution of instruction **ADD R1, R2, R3.**

EXECUTION OF A COMPLETE INSTRUCTION

- Write control sequence for the complete execution of instruction **SUB R1, R2, #6.**

EXECUTION OF A COMPLETE INSTRUCTION

- Write control sequence for the complete execution of instruction **MUL R1, R2, (R3).**

EXECUTION OF A COMPLETE INSTRUCTION

- Now we can write control steps for complete execution of an instruction. Control steps for the instruction **ADD R1, (R2), (R3)** can be written as following -

1. $PC_{out},\ MAR_{in},\ READ,\ Select4,\ ADD,\ Z_{in}$
2. $PC_{in},\ Z_{out},\ MDR_{inE},\ WMFC$
3. $MDR_{out},\ IR_{in}$
4. $R2_{out},\ MAR_{in},\ READ$
5. $MDR_{inE},\ WMFC$
6. $MDR_{out},\ Y_{in}$
7. $R3_{out},\ MAR_{in},\ READ$
8. $MDR_{inE},\ WMFC$
9. $MDR_{out},\ SelectY,\ ADD,\ Z_{in}$
10. $Z_{out},\ R1_{in}$

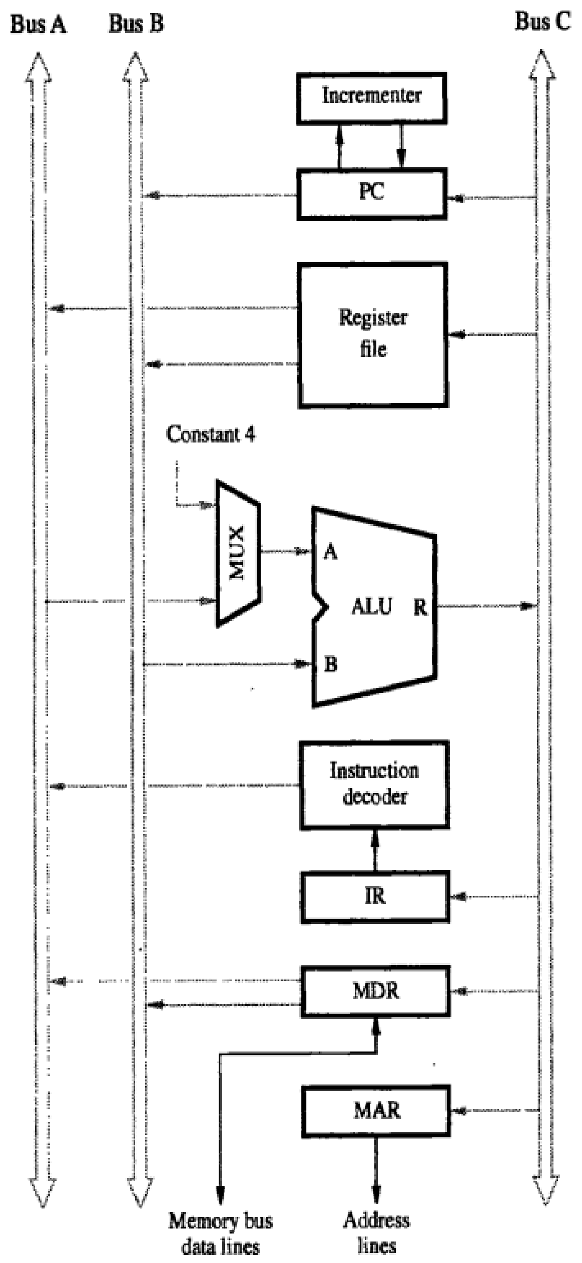
CONTROL SIGNALS FOR BRANCH INSTRUCTION

- Control signals for Branch instruction can written as following –

1. $PC_{out},\ MAR_{in},\ READ,\ Select4,\ ADD,\ Z_{in}$
2. $PC_{in}, Y_{in},\ Z_{out},\ MDR_{inE},\ WMFC$
3. $MDR_{out},\ IR_{in}$
4. $Offset - field - of - IR_{out},\ SelectY,\ ADD,\ Z_{in}$
5. $Z_{out},\ PC_{in}$

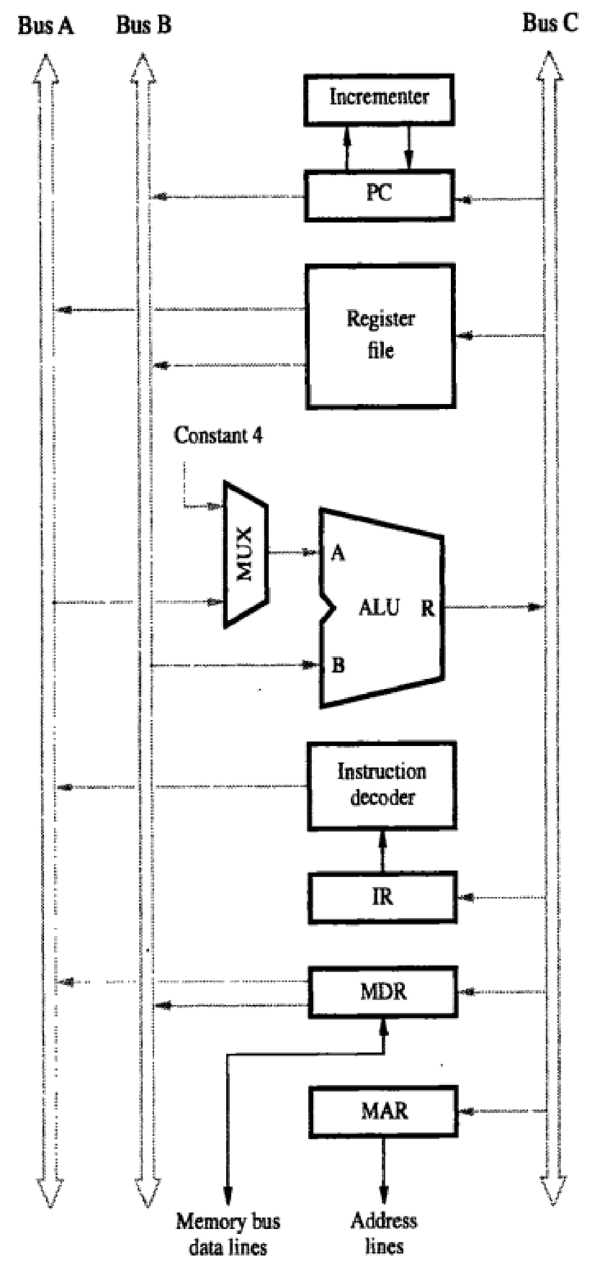
MULTIPLE-BUS ORGANIZATION

- In single-bus organization, overall control sequences for a single instruction are quite long because only one data item can be transferred over the bus in a clock cycle.
- To reduce the number of steps needed, most commercial processors provided multiple internal paths that enable several transfer to take place in parallel.



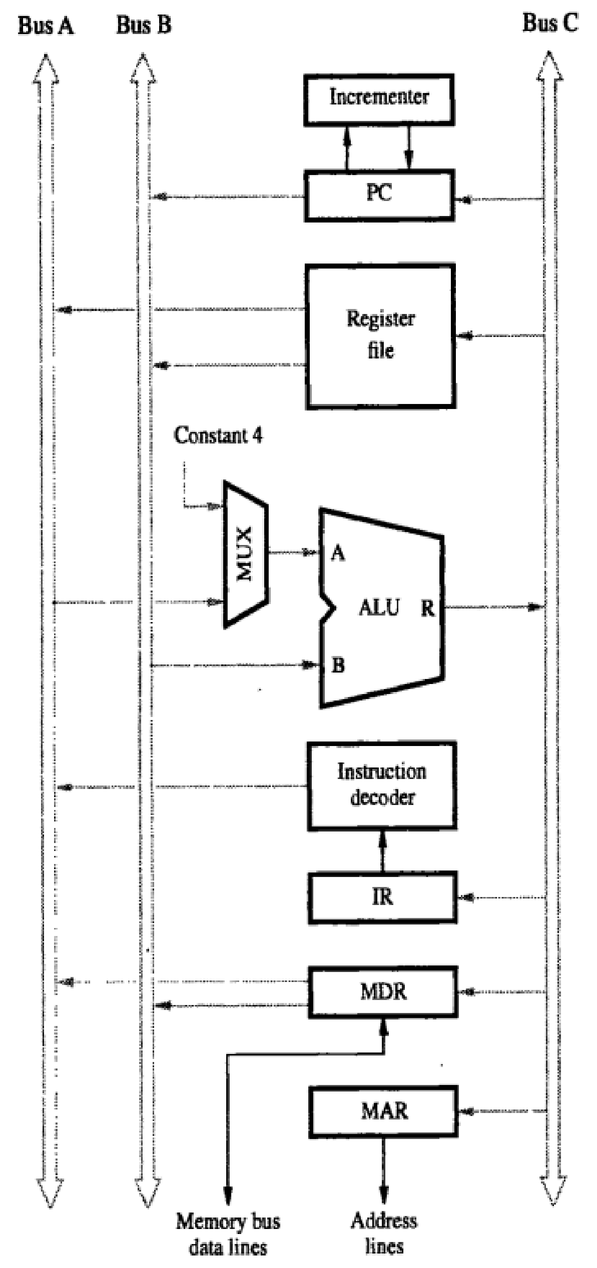
MULTIPLE-BUS ORGANIZATION

- All the general purpose registers are combined into a single block called the *register file.*
- Register file has two output ports allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. And input port allows to write content on 3rd register in same clock cycle.
- Buses A and B are used to transfer the source operands to the A and B inputs of ALU, where an arithmetic or logic operation may be performed.



MULTIPLE-BUS ORGANIZATION

- The result of ALU operation can be transferred over bus C.
- If needed, ALU may simply pass one of its two input operands unmodified to bus C. For that purpose, control signal **R=A** or **R=B** can be used.
- Using *Incrementer* eliminates the need of passing PC through ALU for increment.



CONTROL STEPS IN MULTIPLE-BUS ORGANIZATION

- We can write control steps for instruction **ADD R1, R2, R3** as following –

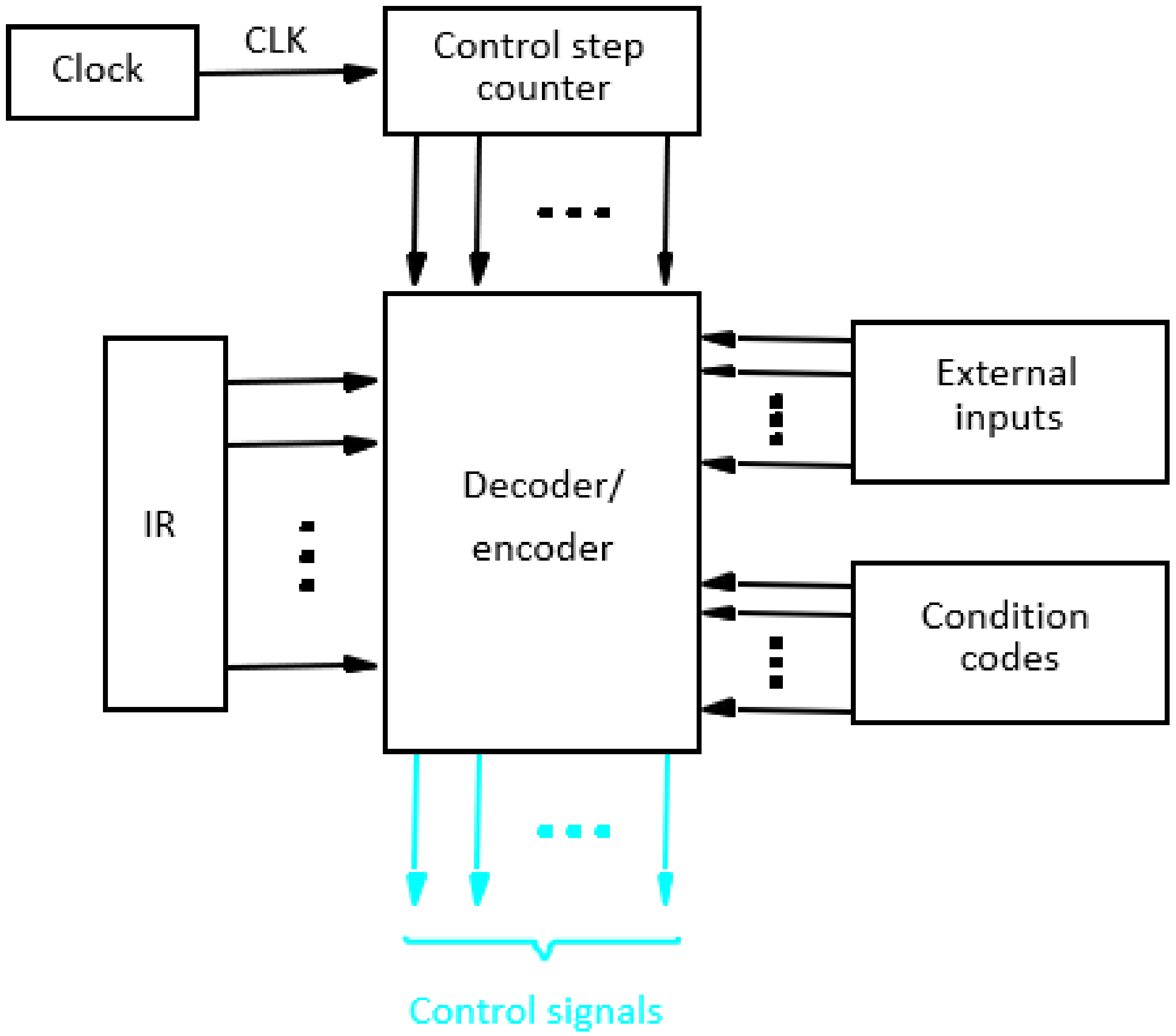
1. $PC_{out},\ R = B,\ MAR_{in},\ READ,\ IncPC$
2. $MDR_{inE},\ WMFC$
3. $MDR_{outB},\ R = B,\ IR_{in}$
4. $R2_{outA},\ R3_{outB},\ SelectA,\ ADD,\ R1_{in},\ End$

HARDWIRED CONTROL UNIT

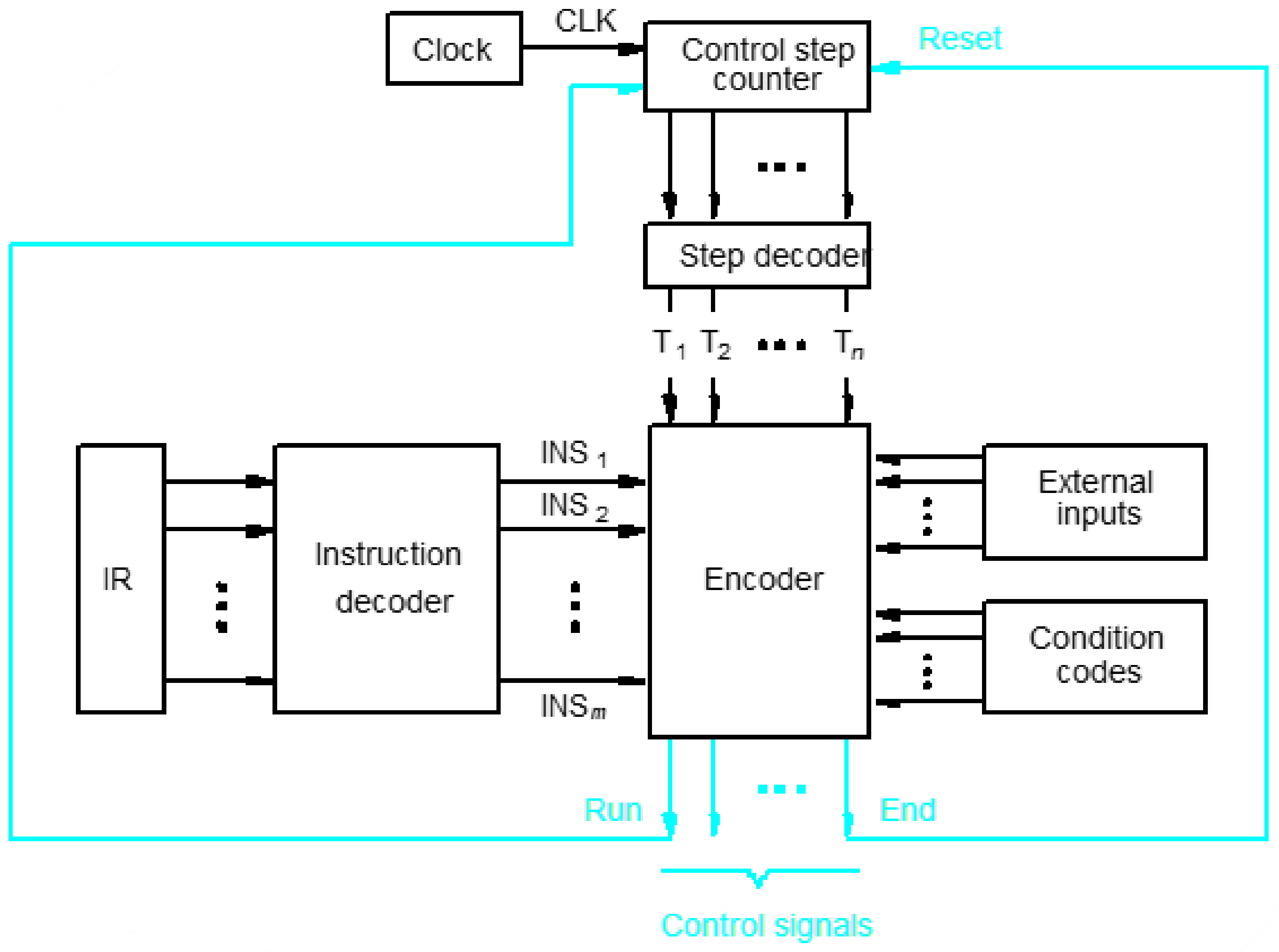
- To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
- There are two types of approaches mostly used by computer designers – hardwired control and microprogrammed control.

HARDWIRED CONTROL

- Which control signal to generate is determined by –
  - Contents of the control step counter
  - Contents of the instruction register
  - Contents of the condition code flags
  - External input signals, such as MFC and interrupt requests



HARDWIRED CONTROL UNIT



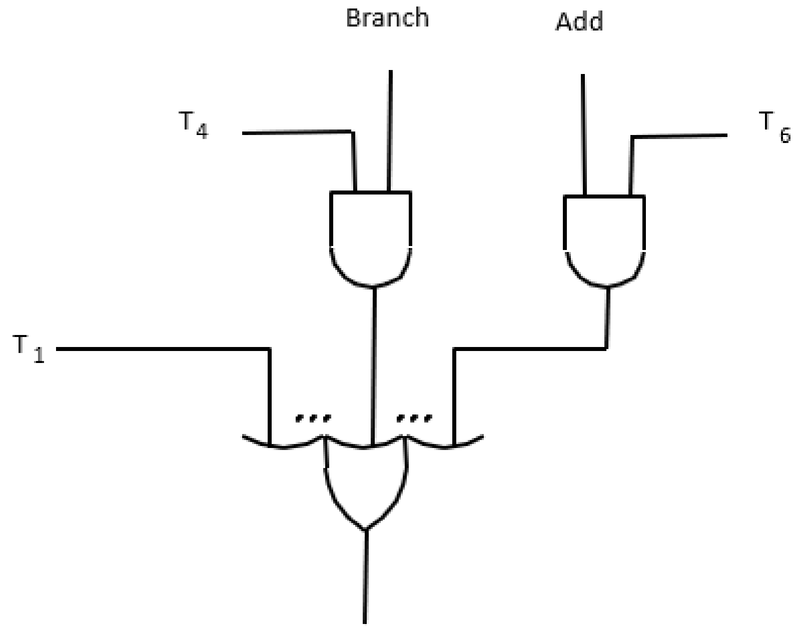
HARDWIRED CONTROL UNIT

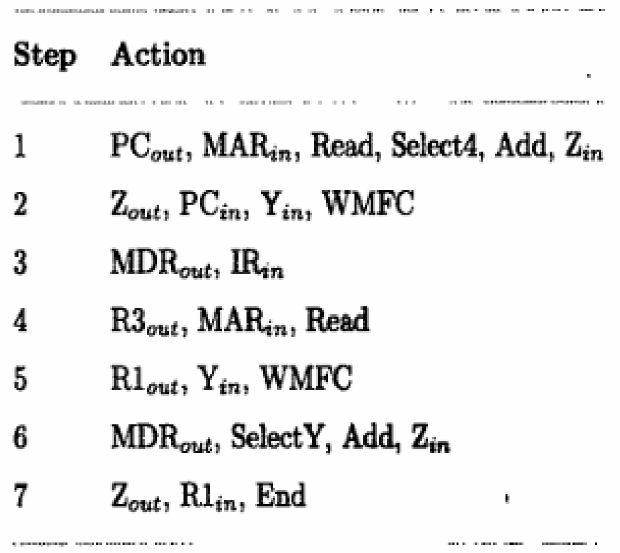
- The step decoder provides a separate signal line for each step or time slot in control sequence.
- The output of instruction decoder consists of a separate line for each machine instruction. For any instruction loaded in the IR, one of the output lines $INS_1, \dots, INS_m$ is set to 1 and all other lines are set to 0.
- The encoder combines all the inputs and generates different control signals like $Y_{in}, PC_{out}, ADD, END,$ and so on.

HARDWIRED CONTROL UNIT: Z_{in}

- For single bus CPU organization, Zin signal can be generated using following circuit diagram -

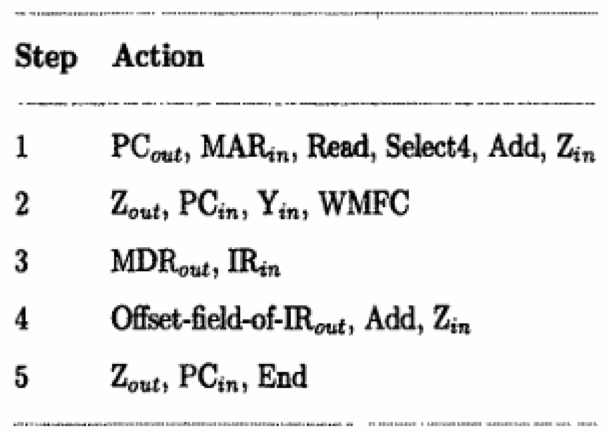
$$Z_{in} = T_1 + T_6 \cdot ADD + T_4 \cdot BR + \ldots$$





Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
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.



Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 7.7 Control sequence for an unconditional Branch instruction.

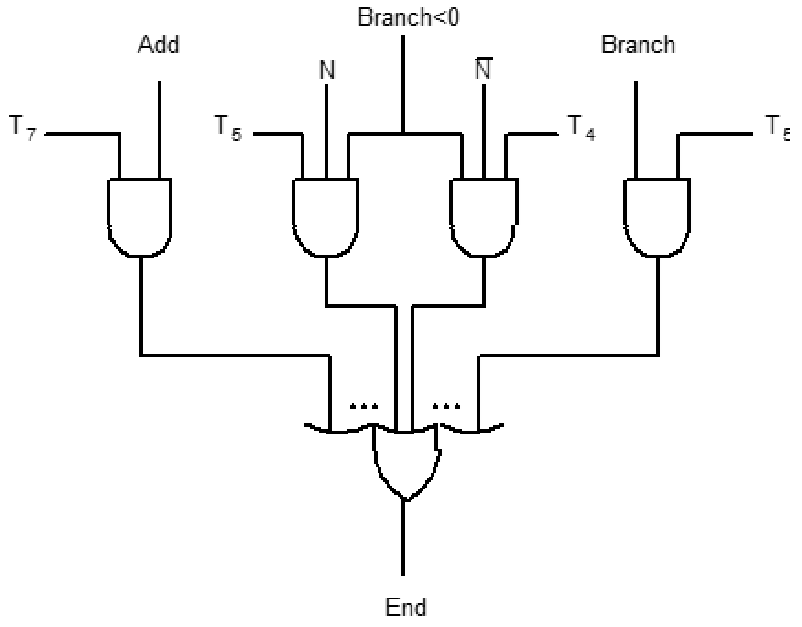
HARDWIRED CONTROL UNIT: RUN & END SIGNALS

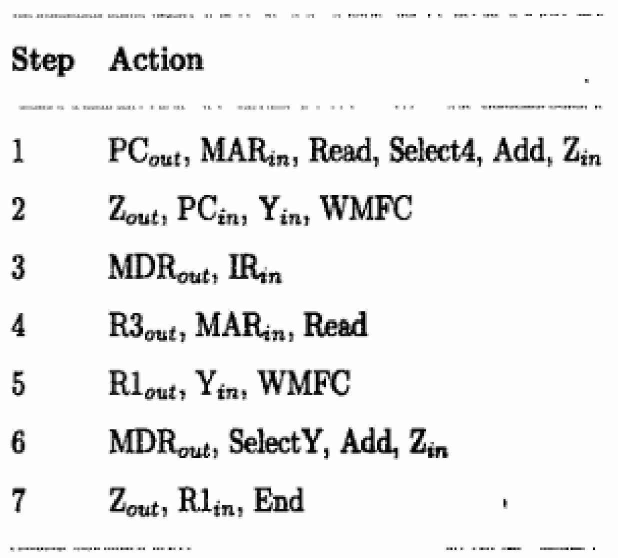
- In hardwired control unit, **RUN** signal is used to update the step counter. When RUN is set to 1, step counter is incremented by 1 at the end of every clock cycle. And when RUN is equal to 0, the counter is paused at current value. This is needed whenever the WMFC signal is issued, to cause the processor to wait for the reply from the memory.
- In hardwired control unit, **END** signal is used to denote the end of current instruction. The end signal starts a new instruction fetch cycle by resetting the control step counter to its starting value.

HARDWIRED CONTROL UNIT: END

- For single bus CPU organization, End signal can be generated using following circuit diagram -

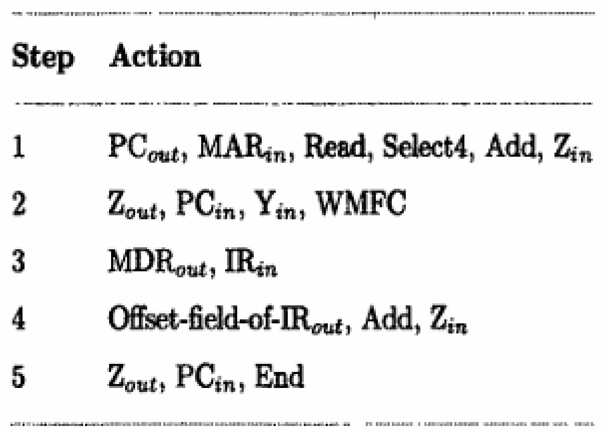
$$End = T_7 \cdot ADD + T_5 \cdot BR + (T_5 \cdot N + T_4 \cdot \overline{N}) \cdot BRN + \ldots$$





Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
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.



Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 7.7 Control sequence for an unconditional Branch instruction.

HARDWIRED CONTROL UNIT

- The control hardware can also be seen as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the conditional flags and the external inputs. The outputs of the state machine are the control signals.
- The sequence of operations carried out by this machine is determined by the wiring of the logic elements hence the name “Hardwired”.
- A controller that uses this approach can operate at high speed.
- Hardwired control unit has little flexibility and the complexity of instruction set it can implement is limited.

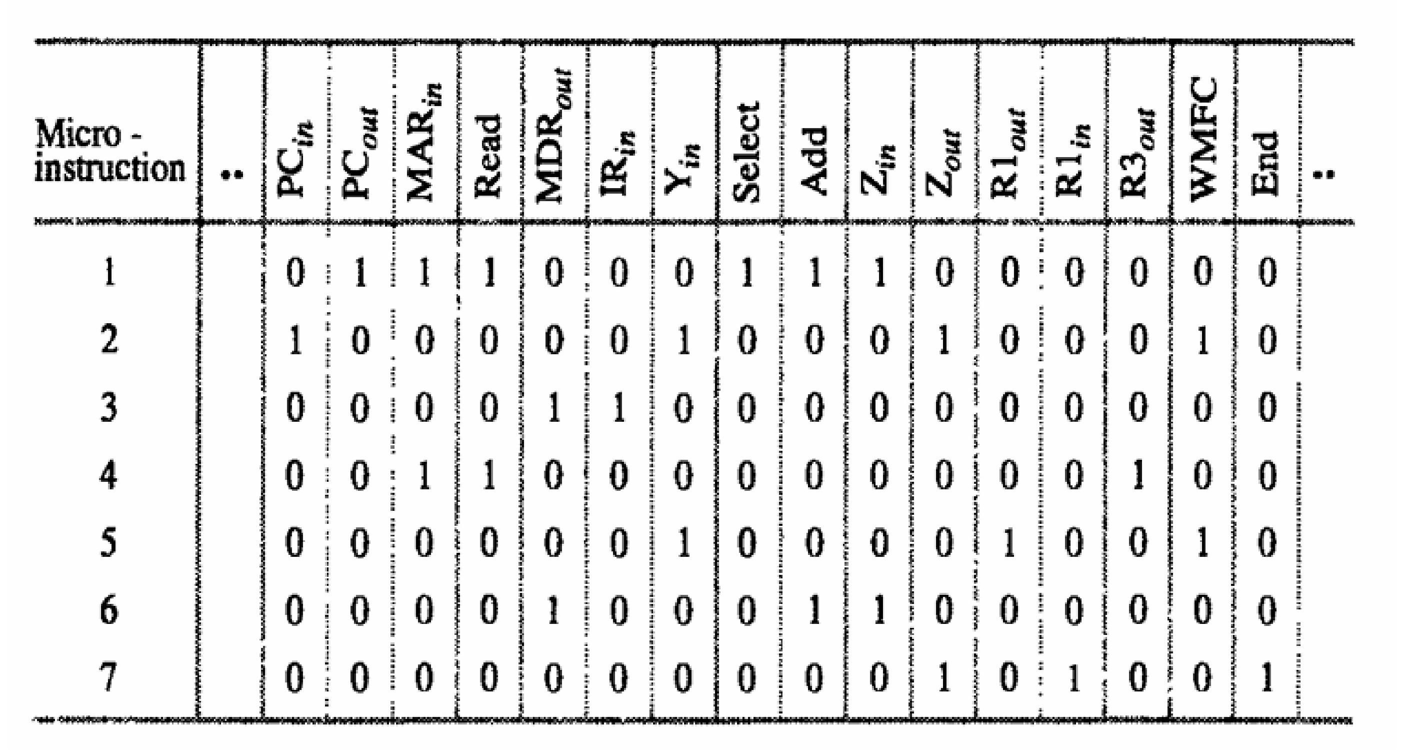
MICROPROGRAMMED CONTROL

- In hardwired control unit, control signals are generated by control step counter and a decoder/encoder circuit which is fast but not much flexible.
- Another approach to generate control signals is based on programs instead of circuits. This approach is called *microprogrammed control unit*. In microprogrammed control unit, control signals are generated by a program similar to machine language programs.

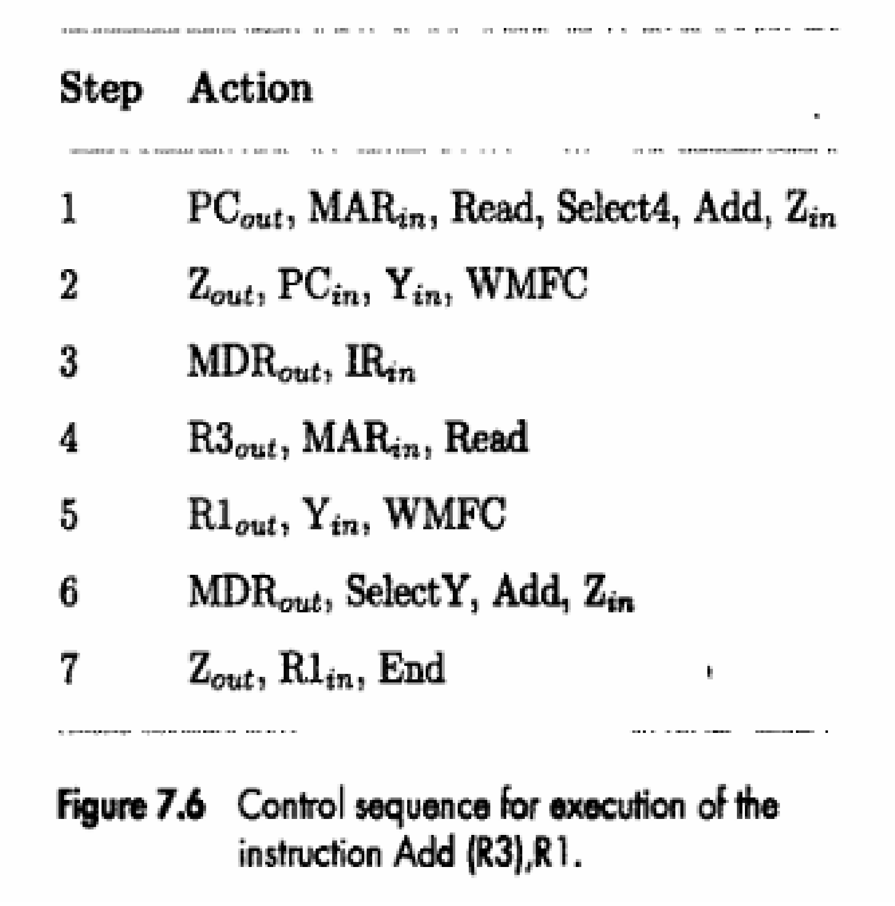
MICROPROGRAMMED CONTROL UNIT

- **Control Word** – Each bit of a control word represents various control signals.
- A sequence of Control Words (CWs) corresponding to the control sequence of a machine instruction is called the *microroutine* for that instruction.
- Individual control words in a microroutine are called *microinstructions.*
- The microroutines for all instructions in the instruction set of a computer are stored in a special memory called *control store.*

MICROPROGRAMMED CONTROL UNIT



Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

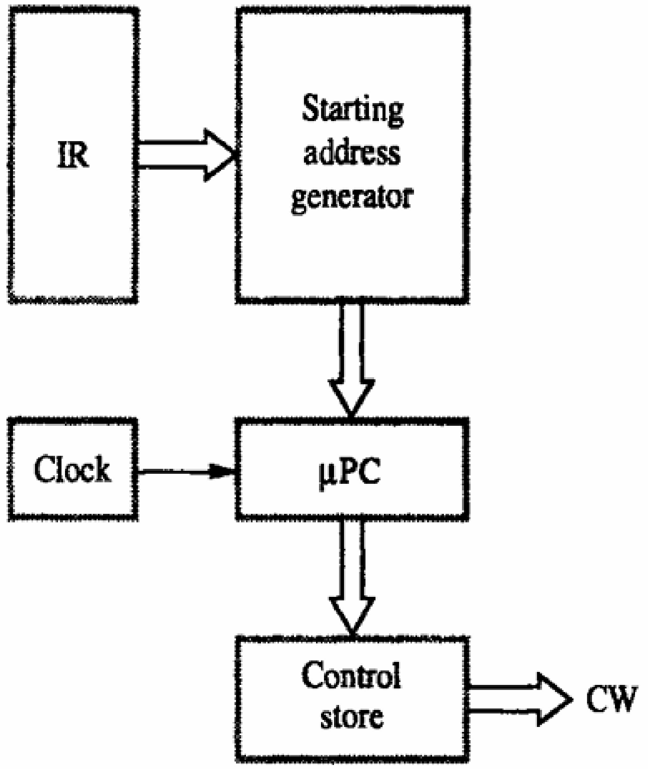


Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

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

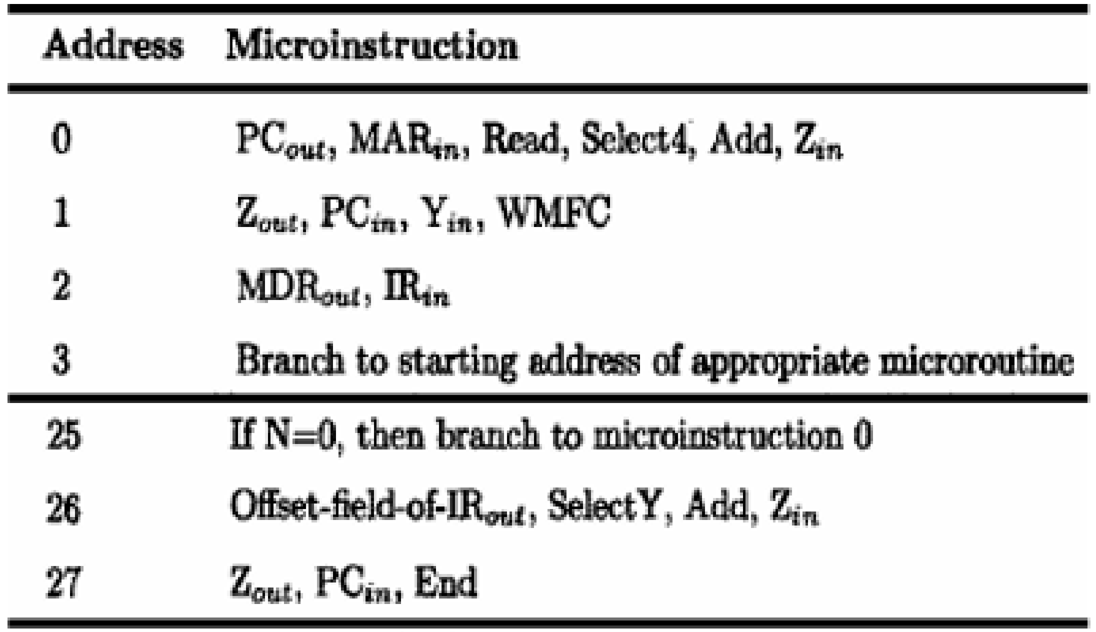
MICROPROGRAMMED CONTROL UNIT

- The control unit can generate the control signals for any instruction by sequentially reading the CWs of the corresponding microroutine from the control store.
- To read the control words sequentially from control store, a *microprogram counter (μPC)* is used.
- Every time a new instruction is loaded into the IR, the output of the block labelled "starting address generator" is loaded into the uPC.
- The uPC is then automatically incremented by the clock, causing successive microinstructions to be read from control store.



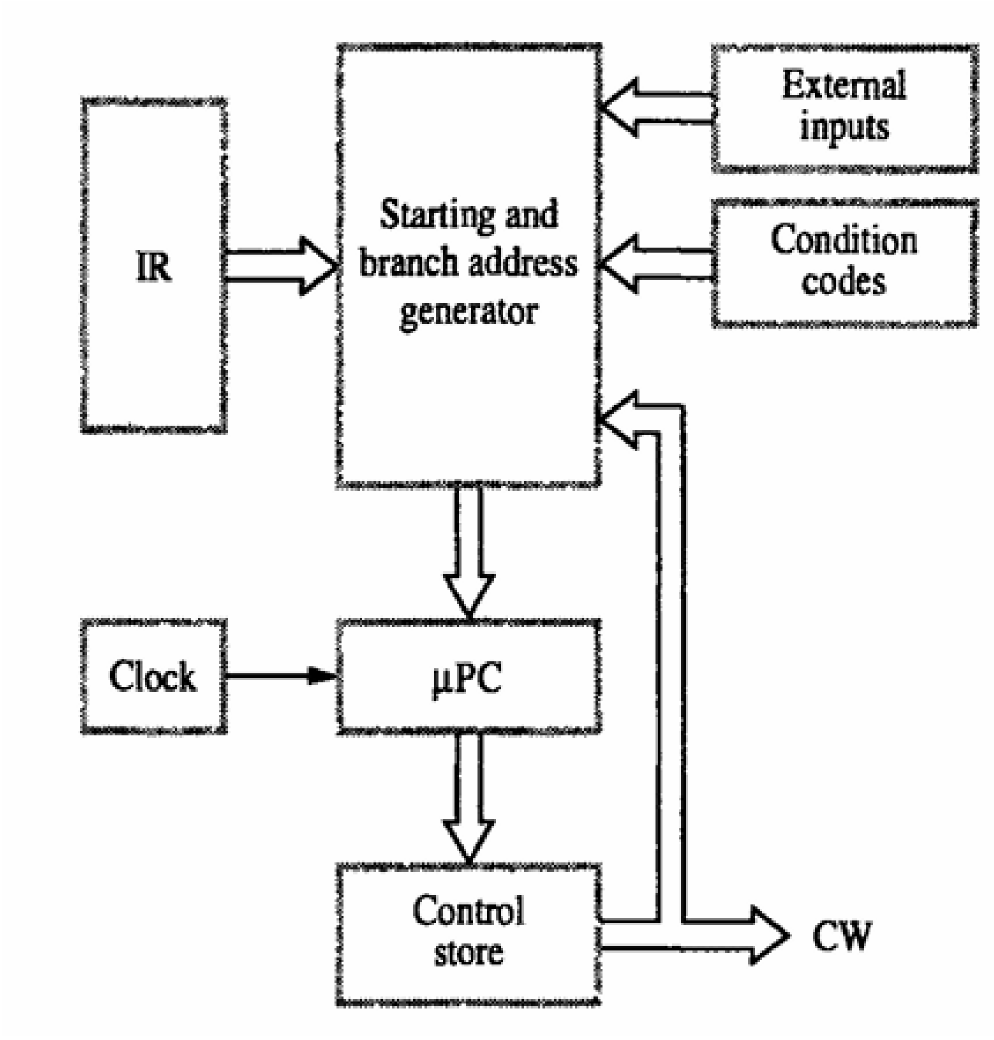
MICROPROGRAMMED CONTROL UNIT

- To support branch machine instruction, Microprogrammed control unit can be modified as shown now address generator will also generate the branch target for uPC.



Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

Figure 7.17 Microroutine for the instruction Branch < 0.



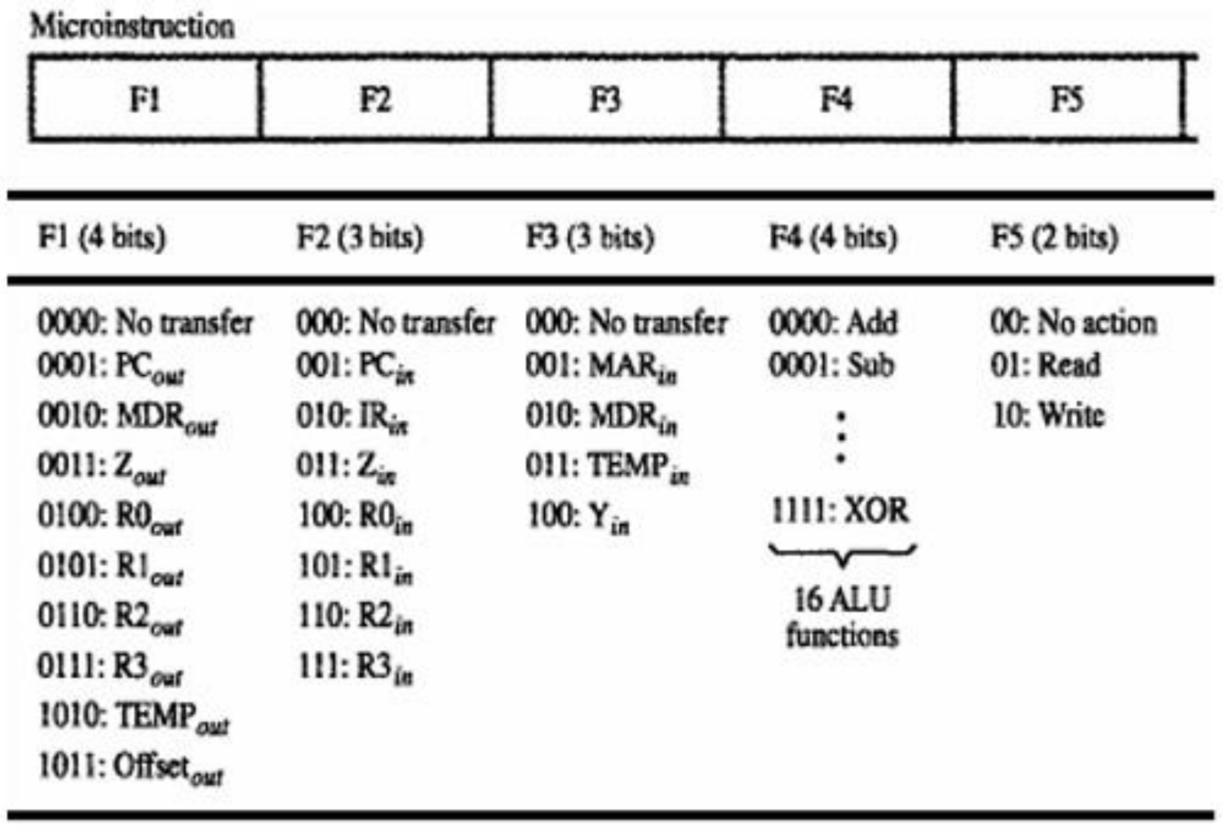
MICROPROGRAMMED CONTROL UNIT: DRAWBACK

- A simple way to structure microinstructions is to assign one bit position to each control-signal required in the CPU. So if there are 42 signals then 42-bits long microinstruction will be required i.e. 42-bit long control word.
- Assigning individual bits to each control-signal results in long microinstructions because the number of required signals is usually large.
- Available bit-space is poorly used because only a few bits are set to 1 in any given microinstruction.
- How can we solve this problem?

MICROPROGRAMMED CONTROL UNIT: SIGNAL GROUPING

- Signals can be grouped because –
  - Most signals are not needed simultaneously.
  - Many signals are mutually exclusive. For example only 1 function of ALU can be activated at a time.
- Grouping control-signals into fields requires a little more hardware because decoding-circuit must be used to decode bit pattern of each field into individual control-signals.

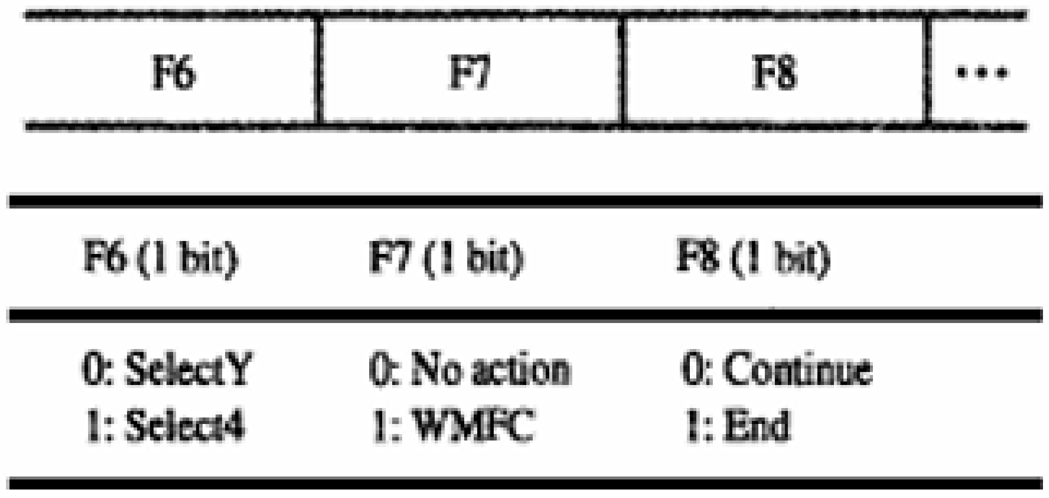
MICROPROGRAMMED CONTROL UNIT: SIGNAL GROUPING



Microinstruction

F1	F2	F3	F4	F5

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PC_{out}	001: PC_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$		
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	1111: XOR	
0101: $R1_{out}$	101: $R1_{in}$		16 ALU functions	
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				



F6	F7	F8	•••

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY	0: No action	0: Continue
1: Select4	1: WMFC	1: End

SIGNAL GROUPING TECHNIQUES

- There are two types of grouping techniques which are followed in microprogrammed control unit –
  - Horizontal technique
  - Vertical technique
- Based on what kind of grouping technique is being used, microprogrammed control unit can be classified into two types –
  - Horizontal Micro-programmed Control Unit
  - Vertical Micro-programmed Control Unit

HORIZONAL MICRO-PROGRAMMED CONTROL UNIT

- In this type, each control signal uses 1 bit in control word.
- It supports longer control words.
- It is used in parallel processing applications.
- It requires no additional hardware (decoders) which means it is faster than vertical micro-programmed control unit.

VERTICAL MICRO-PROGRAMMED CONTROL UNIT

- In this type, control signals are represented using bit patterns.
- It supports shorter control words.
- It allows a low degree of parallelism.
- It requires additional hardware (decoders) to identify control signal from bit pattern. Which means it is slower compared to horizontal micro-programmed control unit.