UNIT-3 CONTROL UNIT

PRIYANKA GOEL
ASST PROF
DEPTT OF CSE

* To execute a program, the processors fetches one instruction at a time and performs the operations specified
* Instructions are fetched from successive memory locations until a branch or a 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 PC are updated to point to the next instruction in the sequence.

Note :- A branch instruction may load a different value into the PC.

* Suppose that each instruction comprises 4 bytes and that it is stored in one memory word.
* To execute an instruction, the processor has to perform the following steps :-

Fetch phase :- (1) Fetch the contents of the memory location pointed to by the PC. The contents of this location are interpreted as an instruction to be executed. So, they are loaded into the Instruction Register (IR). Symbolically, this can be written as :-

$$IR \leftarrow [[PC]]$$

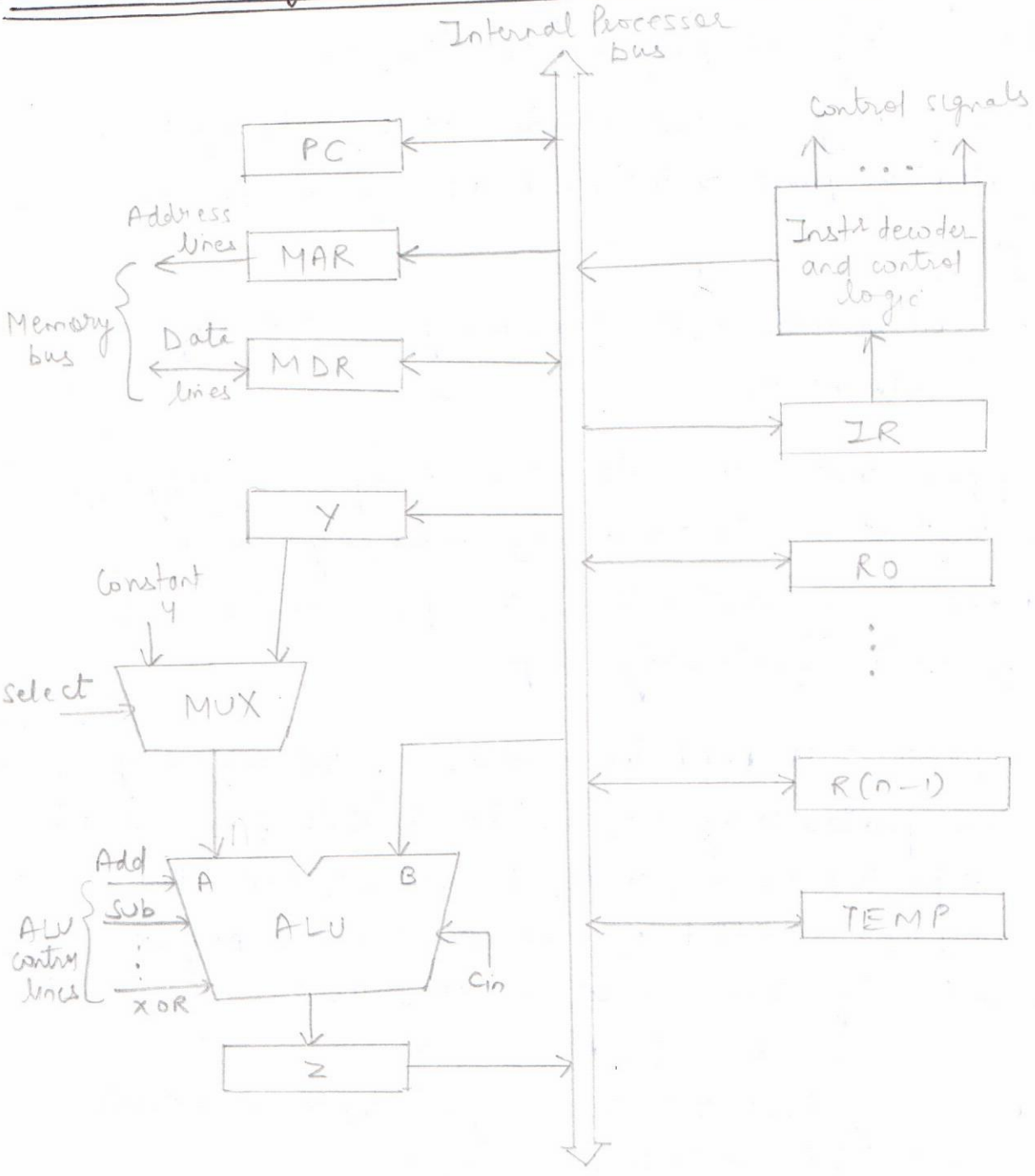
(2) Assuming that the memory is byte addressable, increment the contents of PC by 4

i.e. $PC \leftarrow [PC] + 4$

Execute phase :- Carry out the actions specified by the instructions in the IR.

* In case where an instruction occupies more than one word, fetch phase must be repeated as many times as necessary to fetch the complete instruction.

Internal single-bus organization of the processor



* In the above organization, the arithmetic and logic unit (ALU) and all the registers are interconnected via a single common bus.
* This bus is internal to the processor and should not be confused with the external bus that connects the 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 the Memory Data Register (MDR) and Memory Address Register (MAR) respectively.
* Register MDR has two inputs and two outputs.
* Data may be loaded into MDR either from the memory bus or from the internal processor bus.
* The data stored in MDR may be placed on either bus.
* The inputs of MAR is connected to the internal bus and its output is connected to the external bus.
* The control lines of the memory bus are connected to the instruction decoder and control logic block.
* 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.
* The no. of processor registers R0 through R(n-1) may vary depending from one processor to another.

* Some registers may be dedicated as special purpose registers such as index registers or stack pointers.
* Three registers Y, Z and TEMP are transparent to the programmer i.e. the programmer need not be concerned with them because they are never referenced explicitly by any instruction.
* They are used by the processor for temporary storage during execution of some instructions.
* These registers are never used for storing data generated by one instruction for later use by another instruction.
* The multiplexer MUX selects either the 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.
* These two values can be referred as Select4 and select Y for selecting constant 4 or register Y respectively.
* 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, ALU and the interconnecting bus are collectively referred to as the data path.

* An instruction can be executed by performing one or more of the following operations in some specified sequence :-

a) Transfer a word of data from one processor register to another or to the ALU.

b) Perform an arithmetic or a logic operation and store the result in a processor register.

c) Fetch the contents of a given memory location and load them into a processor register.

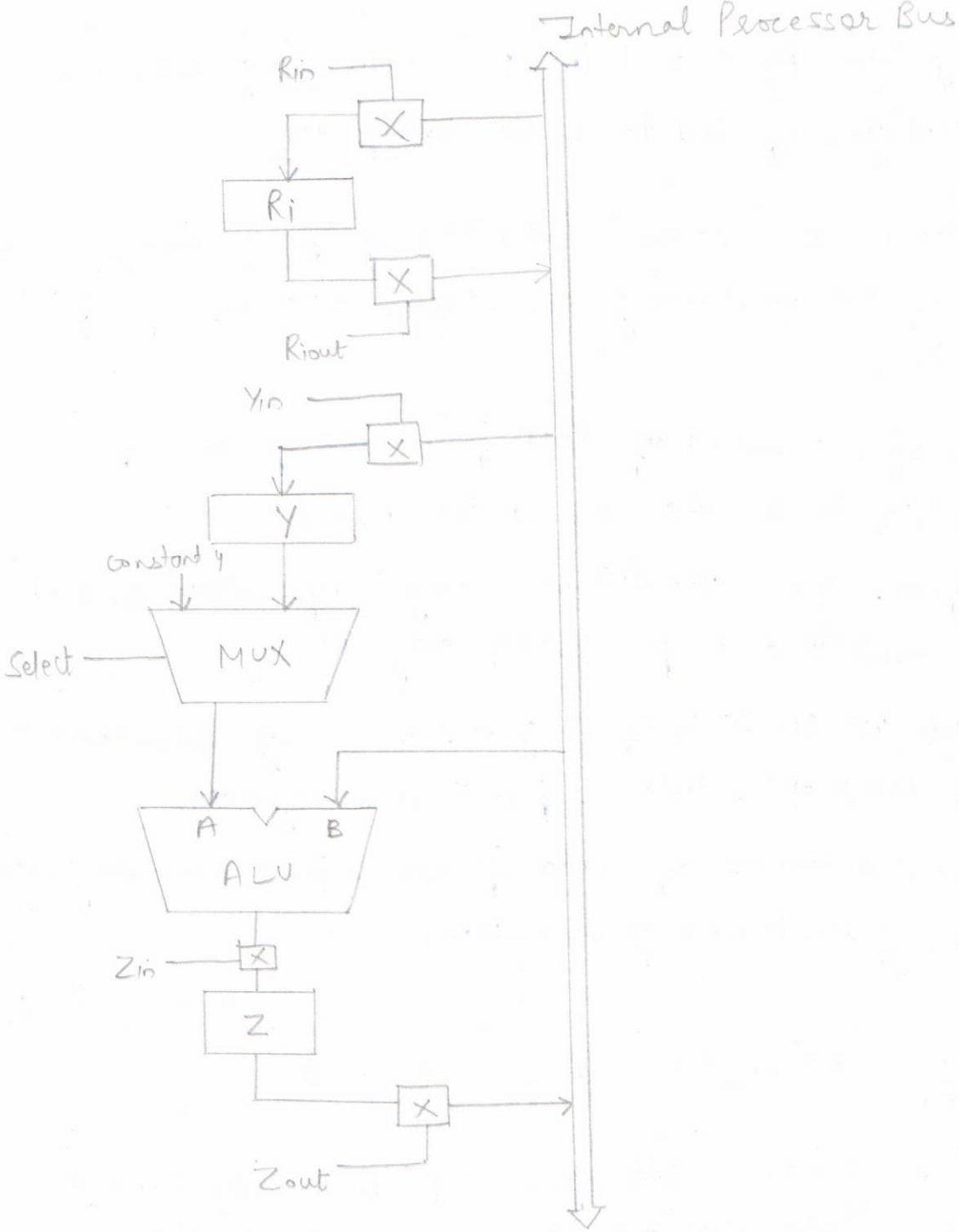
d) Store a word of data from a processor register into a given memory location.

2.1. Performing an Arithmetic or Logic Operation

* ALU has no internal storage. It performs arithmetic and logic operations on the two operands applied to its A and B inputs.

* It is assumed that one of the operands is the output of multiplexer MUX and other operand is obtained directly from the bus.

* The result produced by the ALU is stored temporarily in register Z.



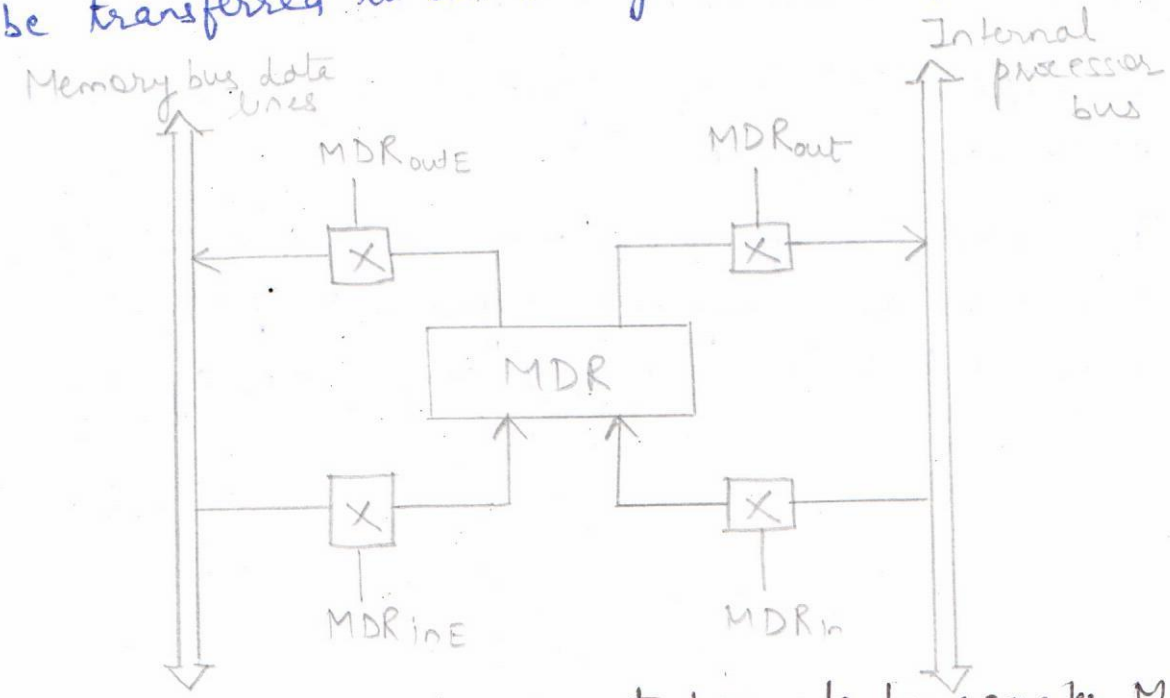
* A sequence of operations to add the contents of register R1 to those of register R2 and store the result in register R3 is :-

1. R1 out, Yin
2. R2 out, SelectY, Add, Zin
3. Zout, R3 in

* During each clock cycle, the signals used in any step are activated while the other signals are inactive.
* So, in step 1, the outputs of register R1 and inputs of register Y are enabled, causing the contents of R1 to be transferred over the bus to Y.
* In step 2, multiplexer's select signal is set to Select Y causing the multiplexer to send the contents of register Y to input A of ALU.
* At the same time, the contents of register R2 are send onto the bus and hence to input B.
* The functions performed by ALU depends on the signals applied to its control lines.
  * In our example, Add line is set to 1, causing the output of ALU to be sum of two numbers at inputs A and B.
  * This sum is loaded into register Z because its input control signal is activated.
  * In step 3, contents of register Z are transferred to destination register R3.
  * This last transfer cannot be carried out during step 2, because only one register output can be connected to the bus during any clock cycle.

Fetching a word from memory

* To fetch a word of information from memory, the processor has to specify the address of the memory location where this information is stored and request a Read operation.
* The information to be fetched may represent an instruction in a program or an operand specified by an instruction.
* The processor transfers the required address to the MAR, whose output is connected to the address lines of the memory bus.
* At the same time, the processor uses the control lines of the memory bus to indicate that a Read operation is needed.
* When the requested data are received from the memory, they are stored in register MDR, from where they can be transferred to other registers in the processor.



Connection & control signals for register MDR

* The connections for register MDR as shown in the figure.
* It has four control signals : MDR_{in} and MDR_{out} control the connection to the internal bus and MDR_{in_E} and MDR_{out_E} control the connection to the external bus.
* During memory Read and Write operations, the timing of internal processor operations must be coordinated with the response of addressed device on the memory bus.
* The processor completes one internal data transfer in one clock cycle.
* On the other hand, the speed of operation of the addressed device varies with the device.
* In some cases, cache memory is used to respond to a memory read operation in one clock cycle.
* However, when a cache miss occurs, the request is forwarded to the main memory which introduces a delay of several clock cycles.
* Moreover, memory-mapped I/O device registers are not cached so their accesses always take a number of clock cycles.
* To accommodate the variability in response time, the processor waits until it receives an indication that the requested Read operation has been completed.
* It is assumed that a control signal called Memory-Function-Completed (MFC) is used for this purpose.

The addressed device sets this signal to 1 to indicate that the contents of the specified location have been read and are available on the data lines of the memory bus.

Now, consider the instruction MOV (R1), R2. → Indirect addressing mode

Following actions are needed to execute this instruction :-

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← [MDR]

* Each of the above actions can be carried/completed in one clock cycle except action 3 which requires one or more clock cycles, depending on the speed of addressed device.
* The above actions can be combined in one or two steps depending on the fact that some actions can be completed in same clock cycle.
* Thus, the following actions can be written as :-

1. R1out, MARin, Read
2. MDRinE, WMFC
3. MDRout, R2in

Where WMFC is the control signal that causes the processor's control circuitry to wait for the arrival of MFC signal.

2.3. Storing a word in memory

* In writing a word into a memory location, the desired address is loaded into MAR.
* Then the data to be written are loaded into MDR, and a write command is issued.
* Hence executing the instruction MOV R2, (R1) requires following sequence:

1. $R1_{out}$, MAR_{in}
2. $R2_{out}$, MDR_{in}, Write
3. MDR_{outE}, WMFC

* As in the case of Read operation, the Write control signal causes the memory bus interface hardware to issue a Write command on the memory bus.
* The processor remains in step 3 until the memory operation is completed and an MFC response is received.

2.4. Execution of a complete instruction

Let us take an example:

Add (R3), R1

which adds the contents of memory location pointed to by R3 to register R1.

* Executing this instruction requires the following actions :-

1. Fetch the instruction
2. Fetch the first operand (contents of memory location pointed to by R3)
3. Perform the addition
4. Load the result into R1.

* Following is the sequence of control steps required to perform these operations for single bus architecture :

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, Select4, Add, Z in
7. Z out, R1 in, End

* In step 1, the instruction fetch operation is initiated by loading the contents of the PC into MAR and sending a Read request to the memory. The Select signal is set to Select4 which causes the multiplexer MUX to select the constant 4. This value is added to the operand at input B, which is the contents of PC and the result is stored in register Z.

* In step 2, the updated value is moved from register Z back into PC and the processor waits for the memory to respond.

* In step 3, the word fetched from the memory is loaded into IR.

* Step 1 through 3 constitute the fetch phase (same for all instructions)

* The instruction decoding circuit interprets the contents of IR at the beginning of step 4.

* This enables the control circuitry to activate the control signals for steps 4 through 7, which constitute the execution phase.

* The contents of register R3 are transferred to the MAR in step 4 and a memory read operation is initiated.

* Then the contents of register R1 are transferred to Y in step 5 to prepare for addition operation.

* When the Read operation is completed, the memory operand is available in register MDR and the addition operand is performed in step 6.

* The contents of MDR are sent to the bus and thus also to the B input of the ALU and register Y is selected as the second input to the ALU by choosing Select Y.

* The sum is stored in register Z, then transferred to R1 in step 7.

* The End signals causes a new instruction fetch cycle to begin by returning to step 1.

Here the contents of PC are copied into register Y when executing Add instructions. This is not needed in general cases. But in branch instruction, the updated value of PC is needed to compute Branch target address.
To speed up the execution of Branch instructions, this value is copied into register Y in step 2.

5. Branch Instructions

* A branch instruction replaces the contents of PC with the branch target address.
* This address is obtained by adding an offset X which is given in the branch instruction, to the updated value of PC.
* Following is the control sequence that implements an unconditional branch instruction:

1. PCout, MARin, Read, SelectY, Add, Zin
2. Zout, PCin, Yin, WMFC
3. MDRout, IRin
4. Offset-field-of-IRout, SelectY, Add, Zin
5. Zout, PCin, End

* Step 1 starts with the fetch phase and in step 3, the instruction is loaded into IR.
* The offset value is extracted from IR by instruction decoding circuit which will also perform sign extension if required.

* Since the value of updated PC is already available in register Y, the offset X is send to the bus in step 4 and an addition operation is performed.

* The result (which is the branch target address) is loaded into the PC in step 5.

Note :- The offset X used in branch instruction is usually the difference between the branch target address and the address immediately following the branch instruction.

e.g. If the branch instruction is at location 2000 and if the branch target address is 2050, the value of X must be 46.

* Consider a conditional branch — In this case, we need to check the status of condition codes before loading a new value into PC.

e.g. for a Branch-on-negative (Branch <0) instruction, step 4 (in previous ~~instruct~~ control sequence) is replaced with :

Offset-field-of-IR$_{out}$, Add, Z$_{in}$, If N=0 then End

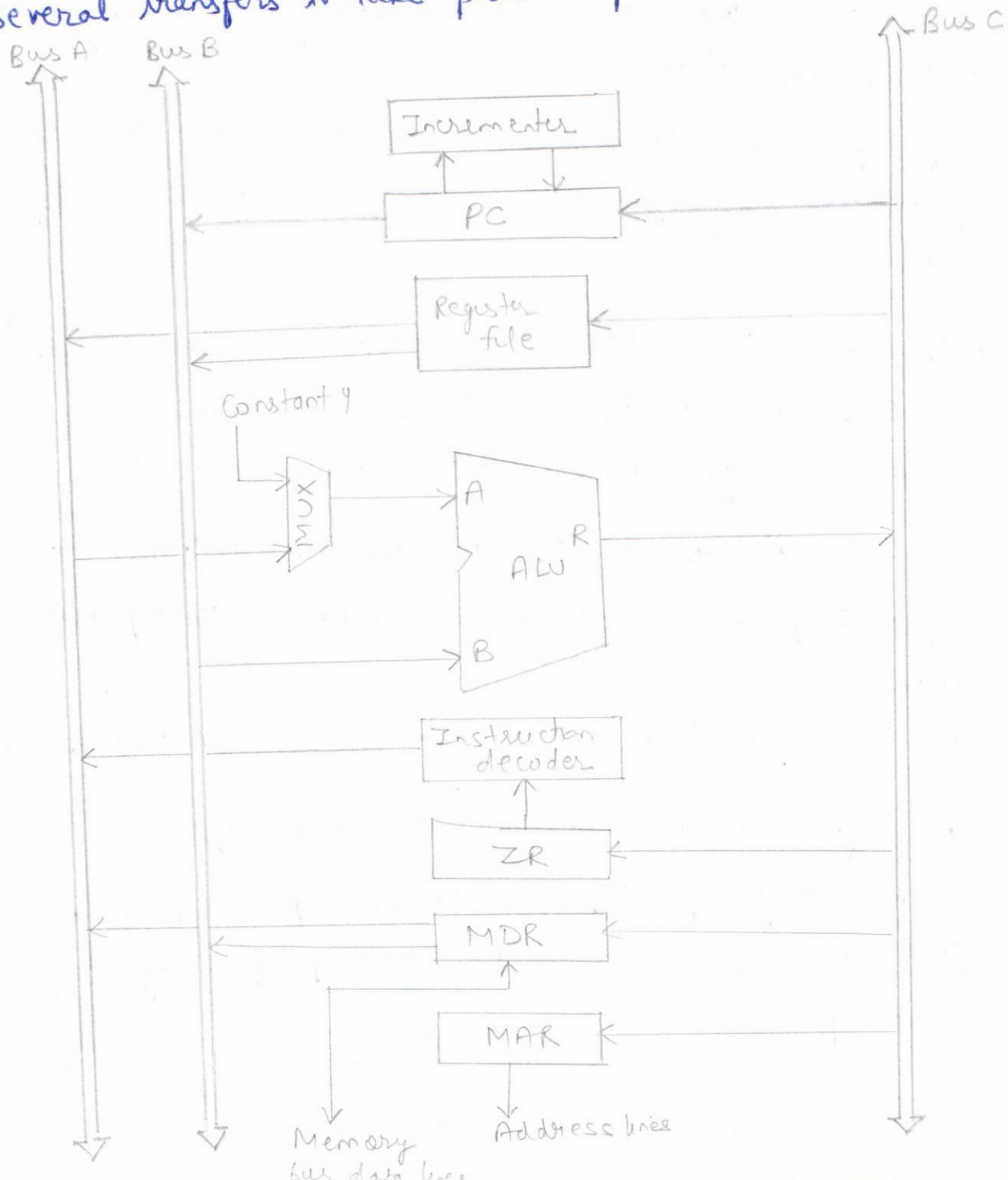
Thus if N=0, the processor returns to step 1 immediately after step 4. If N=1, step 5 is performed to load a new value into PC, thus performing the Branch operation.

N: Set to 1 if result is negative otherwise 0

Multiple Bus Organization

* In single bus organization, the resulting control sequences are quite long because only one data item can be transferred over the bus in a clock cycle.

* ~~The~~ To reduce the number of steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.



Three bus organization of datapath

* All general purpose registers are combined into a single block called register file.
* This is the most efficient way to implement a number of ~~random~~ registers, in the form of an array of memory cells.
* The register file has three ports.
* The two outputs allow the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.
* The third port allows the data on bus C to be loaded into third register during same clock cycle.
* Buses A and B are used to transfer the source operands to the A and B inputs of the ALU where an arithmetic or logic operation may be performed.
* The result is transferred to the destination over bus C.
* If needed, the ALU may simply pass one of its two input operands unmodified to bus C.
* For such operations, ALU control signals can be interpreted as R=A or R=B.
* This three-bus arrangement obviates the need for registers Y and Z.
* Another feature added is the Incrementer unit, which is used to increment the PC by 4.

* This eliminates the need to add 4 to the PC using main ALU.
* The source for the constant 4 at the ALU input multiplexer is still useful — It can be used to increment other addresses such as memory addresses in LoadMultiple and StoreMultiple instructions.

Now consider the three-operand instruction:

Add R4, R5, R6

Its control sequence is given as—

1. PCout, R=B, MARin, Read, IncPC
2. WMFC
3. MDRoutB, R=B, IRin
4. R4outA, R5outB, SelectA, Add, R6in, End

* Step 1: The contents of PC are passed through ALU, using R=B control signal and loaded into MAR to start a memory read operation. At the same time, PC is incremented by 4. The value loaded into MAR is the original contents of PC.
* Step 2: The processor waits for MFC and loads the data received into MDR, th.
* Step 3: This data is transferred to IR
* Step 4: The execution phase of the instruction requires only one control step to complete.

4. Status bit conditions / Condition codes

* The ALU circuit in the CPU is supplemented with a status register where status bit conditions can be stored for further analysis.
* These status bits are also called condition-code bits or flag bits.
* These bits are usually grouped together in a special processor register called condition code register or status register.
* Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.

Four commonly used flags are—

(1) C (Carry) : Sets to 1 if a carry-out results from the operation; otherwise, cleared to 0

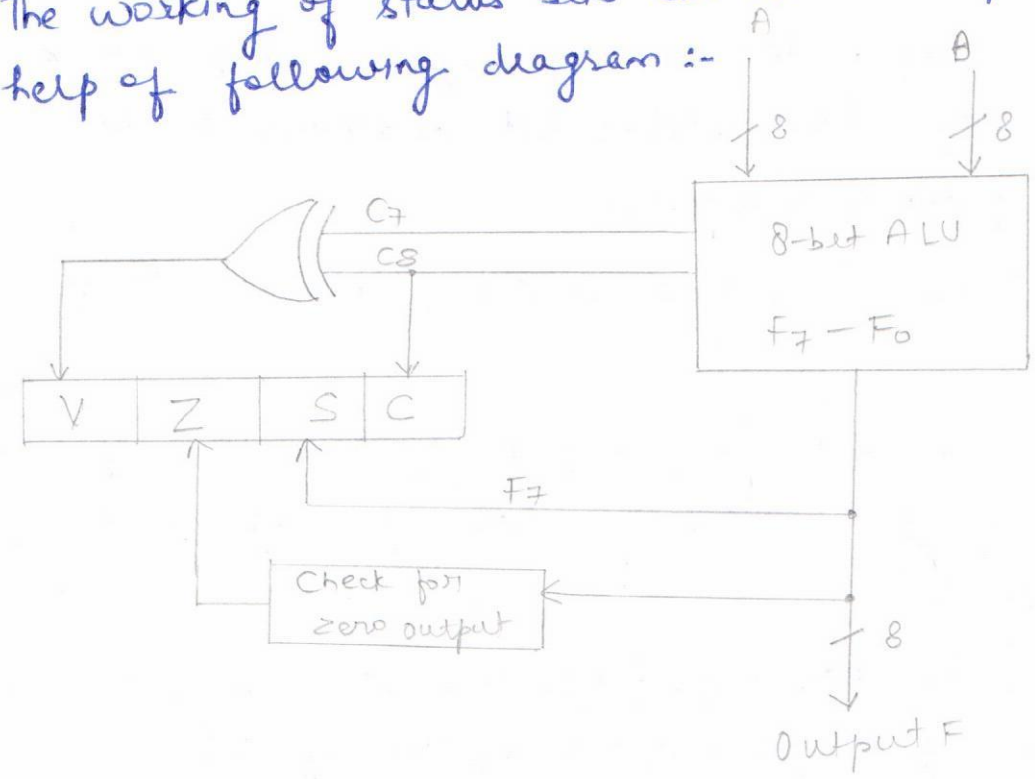
(2) S (sign) or N (negative) :- Set to 1 if the result is negative, otherwise cleared to 0. Alternatively, set to 1 if the highest-order bit is 1; set to 0 if the bit is 0.

(3) Z (zero) :- Set to 1 if the output of ALU contains all 0's. Otherwise, it is cleared to 0.

(4) V (overflow) :- Set to 1 if the XOR of the last two carries is equal to 1, otherwise cleared to 0. This is the condition for an overflow when negative

numbers are in 2's complement.

* The working of status bits can also be explained with the help of following diagram :-



5. Design of Control Unit

* To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
* The control signals are generated in the control unit and provide control unit inputs for the multiplexers in the common bus, control inputs in processor registers and micro-operations for the accumulator.
* There are two approaches for designing control unit :-
  (a) Hardwired control
  (b) Micro-programmed control

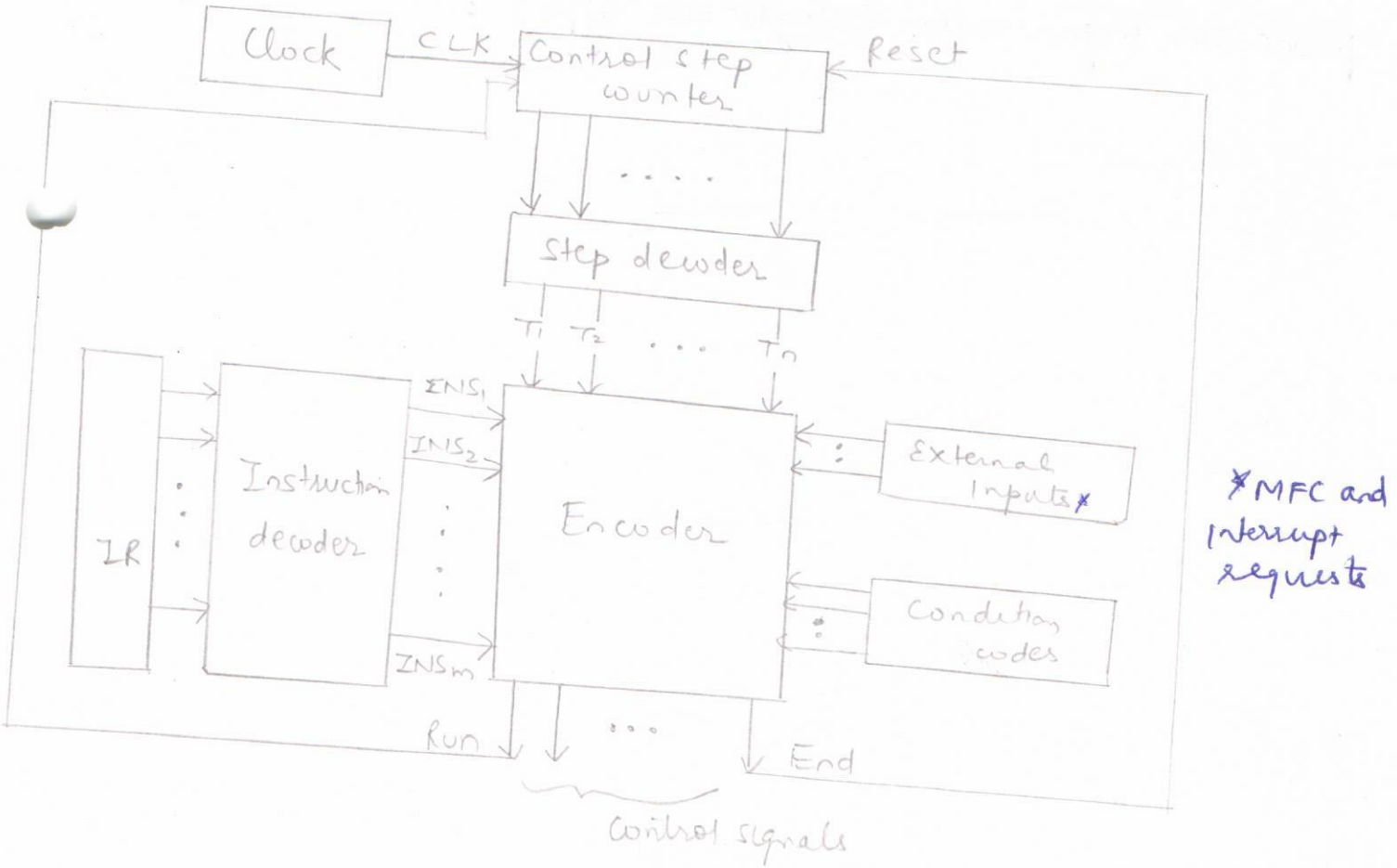
5.1. Hardwired control

* In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders and other digital circuits.

* The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register the condition codes 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 logic elements hence the name "hardwired".



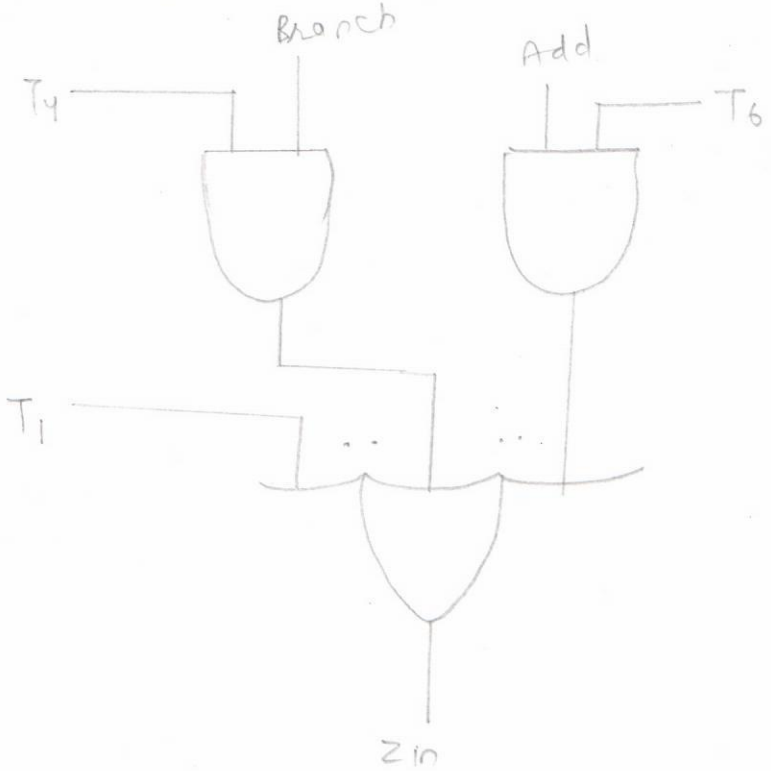
In the diagram, the step decoder provides a separate signal line for each step, or time slot, in the control sequence.

similarly, the output of the instruction decoder consists of a separate line for each machine instruction.

* For any instruction loaded in IR, one of the outputs lines INS$_1$ through INS$_m$ is set to 1, and all other lines are set to 0.

* The input signals to the encoder block are combined to generate individual control signals Yin, PCout, Add, End etc.

e.g How the encoder generates Zin control signal for processor organization (single-bus type) :



* This circuit implements the logic function:

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

* This signal is asserted during time slot T_1 for all instructions, during T_6 for an Add instruction, during T_4 for an unconditional branch instruction and so on.

* When RUN signal is set to 1, RUN causes the counter to be incremented by one at the end of every clock cycle.

* When RUN is equal to 0, the counter stops counting. This is needed whenever the WMFC signal is issued, to cause the processor to wait for the reply from the memory.

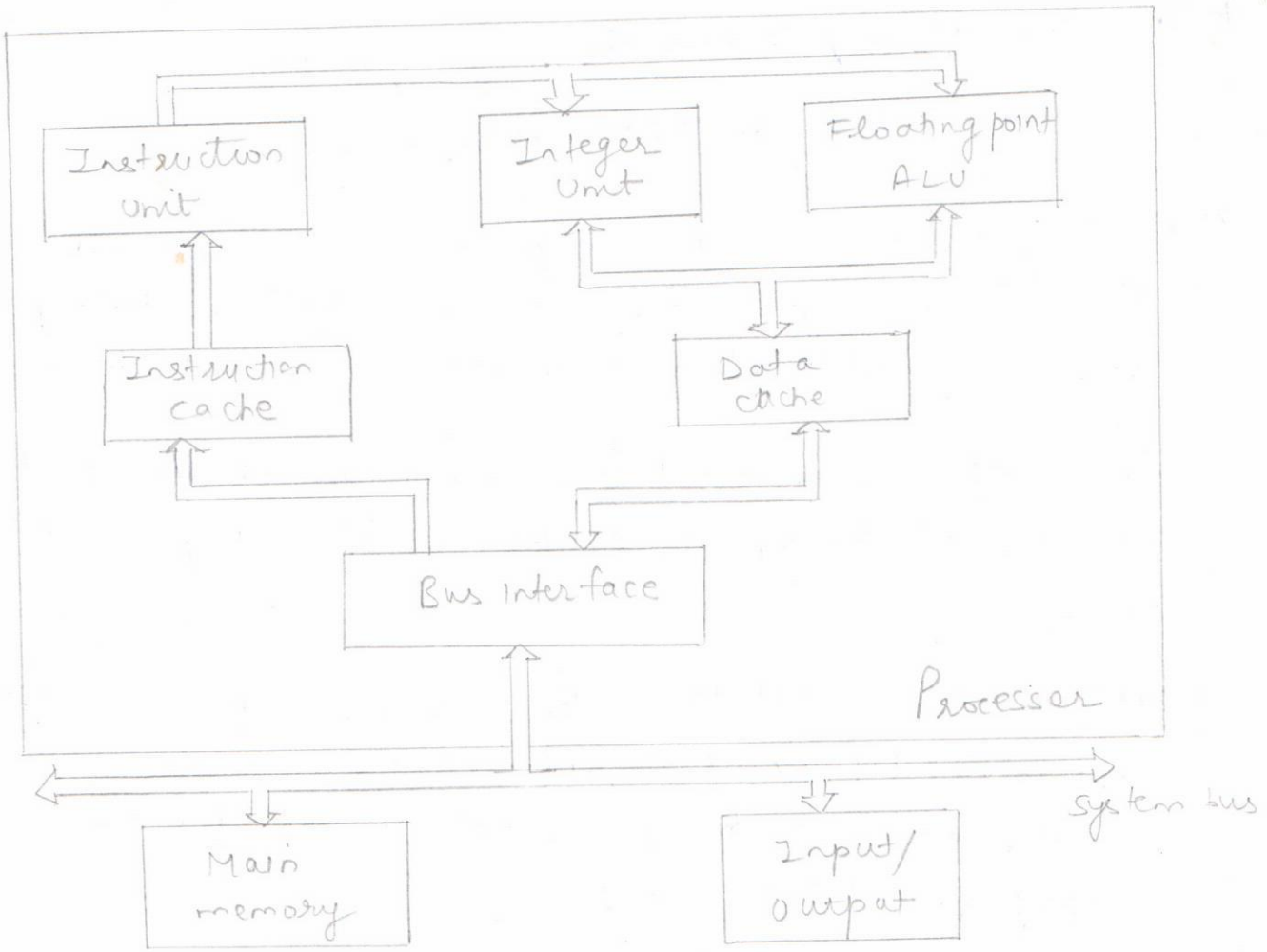
Pros and Cons

* A controller that uses this approach can operate at high speed.

* But it has little flexibility and the complexity of the instruction set it can implement is limited.

5.1.1. A Complete Processor using Hardwired control

* It has an instruction unit that fetches instructions from an instruction cache or from the main memory when the desired instructions are not already in the cache.



* It has separate processing units to deal with integer data and floating point data.
* A data cache is inserted between these units and the main memory.
* The processor is connected to the system bus and to the rest of the computer, by means of bus interface.
* A processor may include several units of each — integer data and floating point data — to increase the potential for concurrent operations.

5.2. Microprogrammed control (control signals are generated by a program similar to m/c language program)

Few terms used :-

(a) **Control Word** :- It is a word whose individual bits represent various control signals. Each of the control steps in the control sequence of an instruction defines a unique combination of 1s and 0s in the control word.

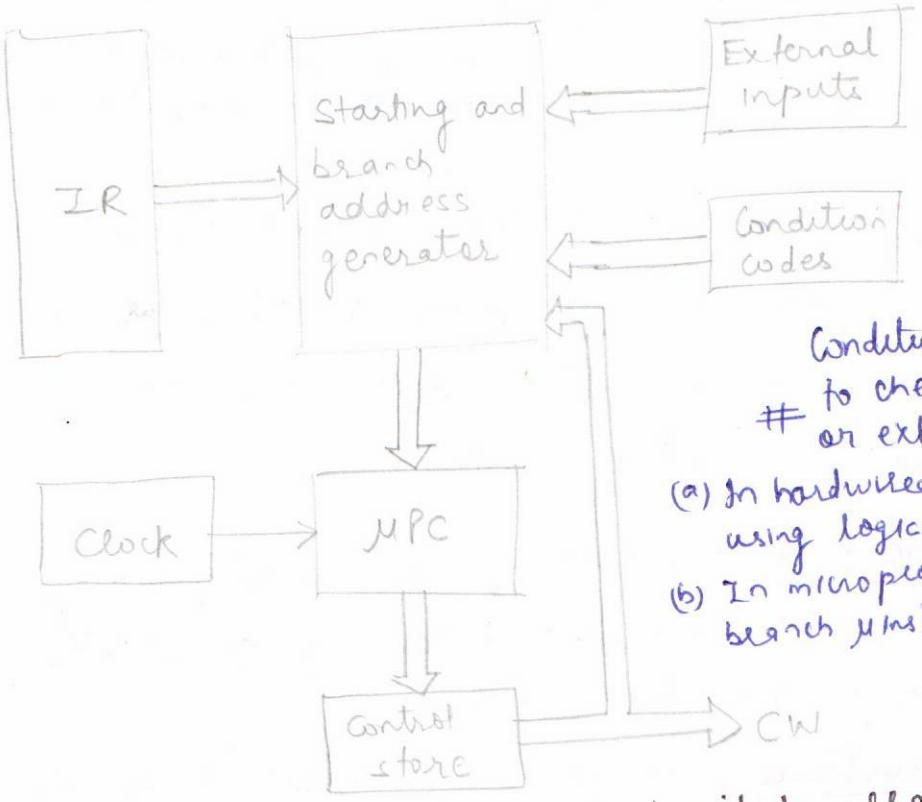
(b) **Micro-routine** :- A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.

(c) **Micro-instructions** :- Individual control words in the micro-routine are referred to as micro-instructions.

(d) **Control store** :- The micro-routines for all instructions in the instruction set of a computer are stored in a special memory called control store.

The control unit can generate the control signals for any instruction by sequentially reading the CWs of the corresponding micro-routine from the control store.

* To read the control words sequentially from the control store, a micro-program counter (μPC) is used.
* Every time a new instruction is loaded into the IR, the output of the block labelled 'starting address generator' (and branch) is loaded into the μPC.
* The μPC is then automatically incremented by the clock, causing successive micro-instructions to be read from control store.



Condition when CU is required to check status of condition bits or external I/Ps to choose b/w :-

(a) In hardwired, this could be handled using logic fun^c.

(b) In microprogrammed, conditional branch μinstr are used.

Organization of control unit ~~to allow~~

* Thus, the control signals are delivered to various parts of the processor in the correct sequence. #

* To support micro-program branching, inputs to the 'starting and branch address generator' consists of the external inputs and condition codes as well as the contents of the IR.

* In this control unit, the μPC is incremented every time a new micro-instruction is fetched from the micro-program memory, except in following situations :-

(a) When new instruction is loaded into IR, the μPC is loaded with starting address of micro-routine for that instruction.

(b) When a Branch micro-instruction is encountered and the branch condition is satisfied, the μPC is loaded with branch address.

(c) When an End micro-instruction is encountered, μPC is loaded with address of first CW in micro-routine for instruction fetch cycle.

In case of micro-programmed control, the instruction Branch<0 can be implemented by a micro-routine as :-

Address	Micro-instruction
0	PCout, MARin, Read, Select, Add, Zin
1	Zout, PCin, Yin, WMFC
2	MDRout, IRin
3	Branch to starting address of appropriate micro-routine
. . . . .	. . . . .
25	If N=0, then branch to micro-instruction 0
26	Offset-field-of-IRout, SelectY, Add, Zin
27	Zout, PCin, End

Format of micro-instructions

There are two approaches for designing micro-instructions :-
(a) Vertical organization
(b) Horizontal organization

Vertical Organization

* Highly encoded schemes that use compact codes to specify only a small number of control functions in each micro-instruction are referred to as vertical organization.
* This approach results in slower operating speeds because more micro-instructions are needed to perform desired control functions.

Micro-instruction

F1	F2	F3	F4	F5	F6	F7	F8	...

F1 (4 bits)

0000 : No transfer
0001 : PCout
0010 : MDRout
0011 : Zout
0100 : R0 out
0101 : R1 out
0110 : R2 out
0111 : R3 out
1010 : TEMPout
1011 : offsetout

F2 (3 bits)

000 : No transfer
001 : PCin
010 : IRin
011 : Zin
100 : R0 in
101 : R1 in
110 : R2 in
111 : R3 in

F3 (3 bits)

000 : No transfer
001 : MARin
010 : MDRin
011 : TEMPin
100 : Yin

F4 (4 bits)

0000 : Add
0001 : Sub
⋮
1111 : XOR

16 ALU functions

F5 (2 bits)

00 : No action
01 : Read
10 : Write

F6 (1 bit)

0 : select y
1 : select 4

F7 (1 bit)

0 : No action
1 : WMFC

F8 (1 bit)

0 : Continue
1 : End

* In vertical organization, signals can be grouped so that all mutually exclusive signals are placed in same group.
* Thus, almost one micro-operation per group is specified in any microinstructions.
* e.g. Register output control signals can be placed in a group consisting of PCout, MDRout, Zout, Offset out, R0 out, R1 out, TEMPout.

* In given example —

  F1 : includes binary codes for all register output control signals.
  0000 signifies that the contents need to be placed on bus.

  F2 : Includes all register input signals

  F3 : Includes specific registers

  F4 : binary code for functions.

  F5 : memory operation

  F6 : Selection of MUX input

  F7 : wait for m/m operation

  F8 : termination of current program

* Grouping control signals into fields requires a little more hardware because decoding circuits must be used to decode the bit patterns of each field into individual control signals.

* The cost of this additional hardware is more than offset by the reduced number of bits in each micro-instruction, which results in a smaller control store.
In given example, only 20 bits are needed to store the patterns for 42 signals.

Horizontal Organization

* Minimally encoded scheme in which many resources can be controlled with single micro-instruction is called horizontal organization.

* This approach is useful when higher operating speed is desired and when the machine structure allows parallel use of resources.

e.g. When we are given control sequence as :

1. PCout, MARin, Read, Select4, Add, Zin
2. Zout, PCin, Yin, WMFC
3. MDRout, IRin
4. R3out, MARin, Read
5. R1out, Yin, WMFC
6. MDRout, Select4, Add, Zin
7. Zout, R1in, End

Corresponding micro-instructions :-

* Select4 = 1
  Select4 = 0

Micro-instruction	PCin	PCout	MARin	Read	MDRout	IRin	Yin	Select	Add	Zin	Zout	R1out	R1in	R3out	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	1	0	1	0	0	1		

- In this approach, each control signal is assigned one bit position.
- But this results in long micro-instructions because the no. of required signals is usually large.
- Since few bits are set to 1 (to be used for active gating) in any given micro-instruction, the available bit space is poorly used.

Difference between horizontal and vertical organization

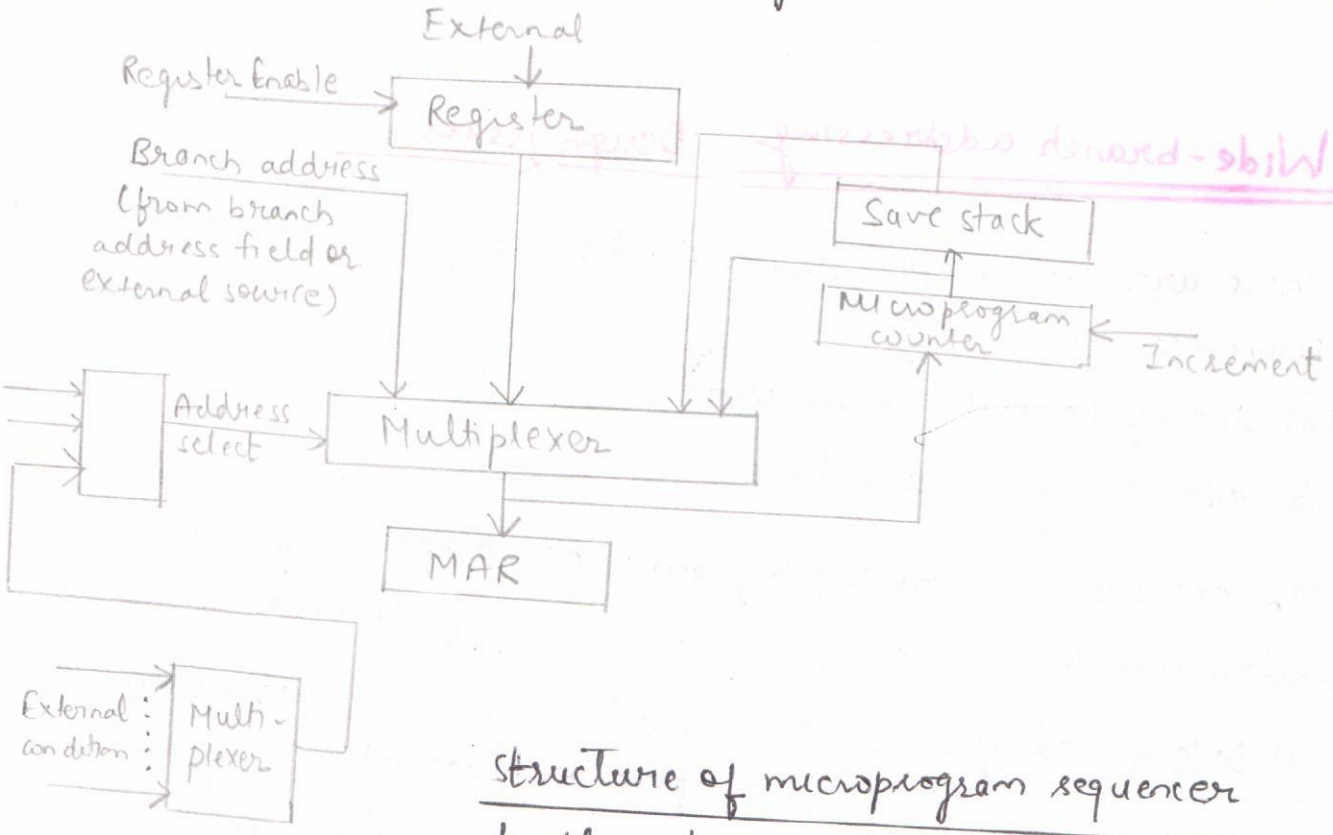
S.No.	Horizontal	Vertical
1	Long formats	short formats
2	Ability to express a high degree of parallelism	Limited ability to express parallel micro-operations
3	Little encoding of the control information	Considerable encoding of the control information
4	Useful when higher operating speed is desired.	Slower operating speed

Microprogram Sequencing

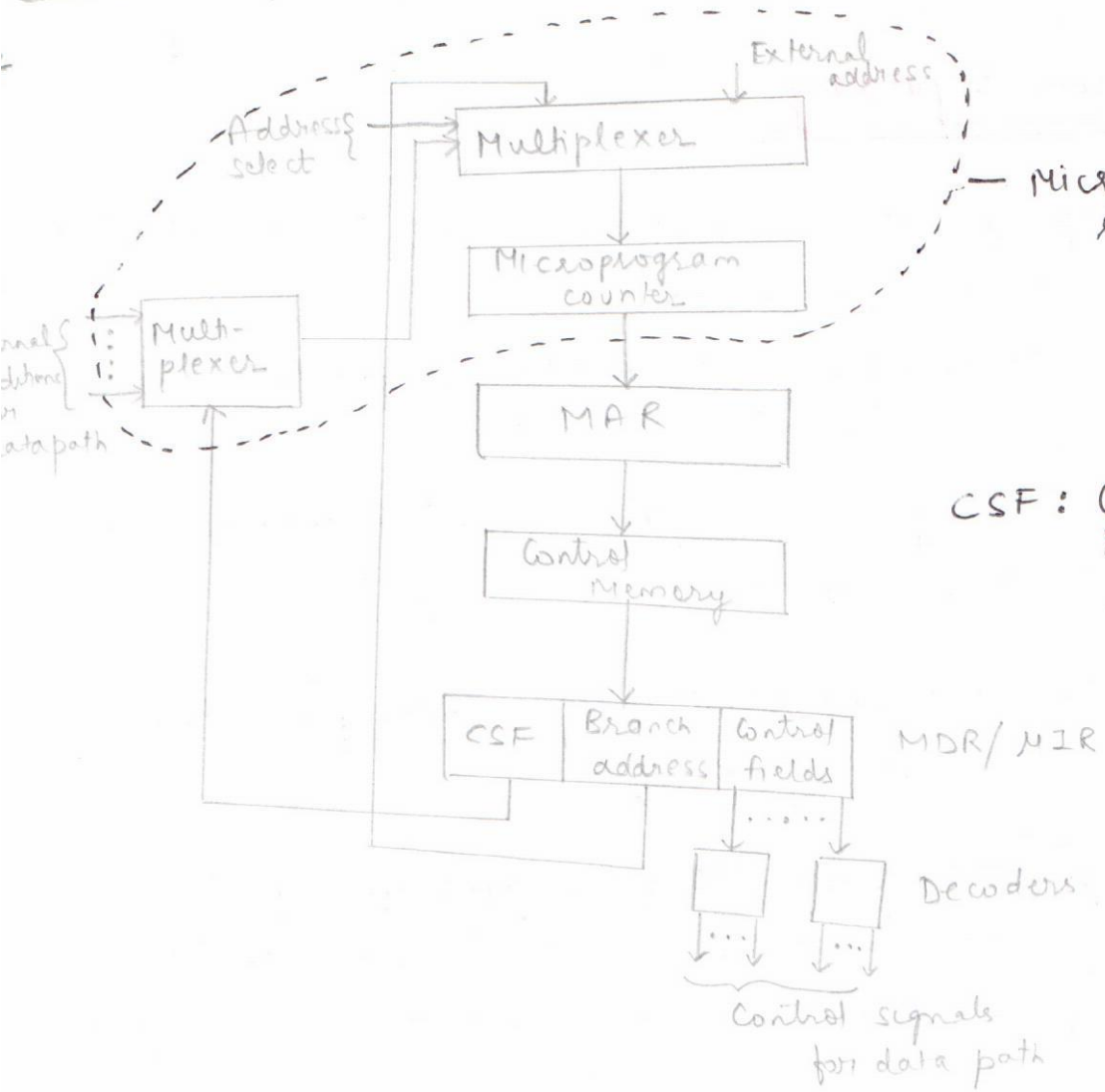
* The microprogrammed control structure can be divided into two distinct parts —
  (a) next-address generation hardware for the control memory
  (b) control memory along with associated decoders to generate control signals.

* Microprogram sequencer is responsible for controlling micro-instruction sequencing.

* It may need to handle four types of input addresses — microprogram counter, stack, external address from instruction register and branch address.



Structure of microprogram sequencer handling four input addresses



CSF : Condition Select Field

Wide-branch addressing Design Issues

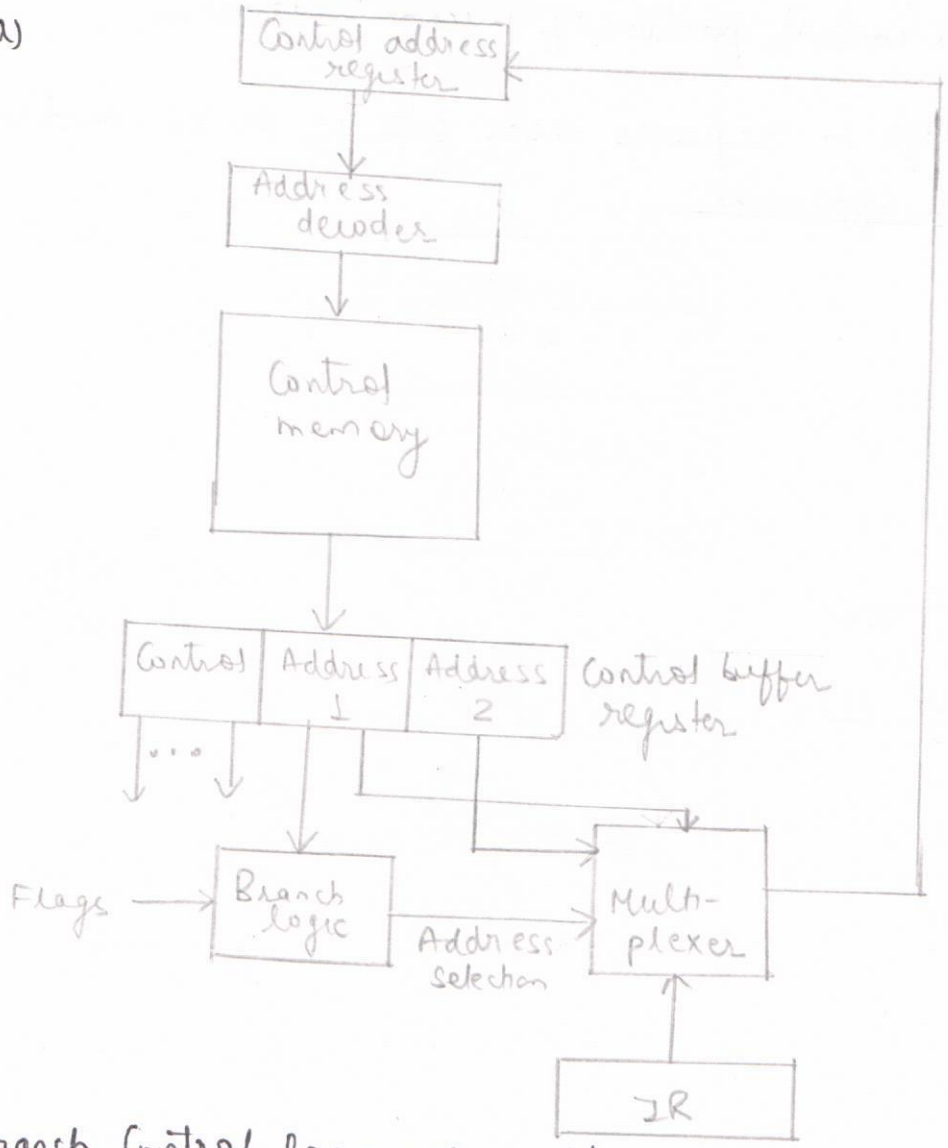
* There are two mayer issues in the design of microinstruction sequencer —
  (a) size of the microinstruction
  (b) address generation time

* In executing a microprogram, the address of next micro-instruction to be executed is in one of following categories—
  (a) Determined by IR :- occurs only once per instruction cycle, just after an instruction is fetched
  (b) Next sequential address :- occurs in most microprograms
  (c) Branch :- both conditional and unconditional

Sequencing techniques

* Based on current micro-instruction, condition flags and contents of instruction register, a control memory address must be generated for next micro-instruction.
* There are three categories based on format of address information in the micro-instruction :
  (a) Two address fields
  (b) Single address field
  (c) Variable format

(a)

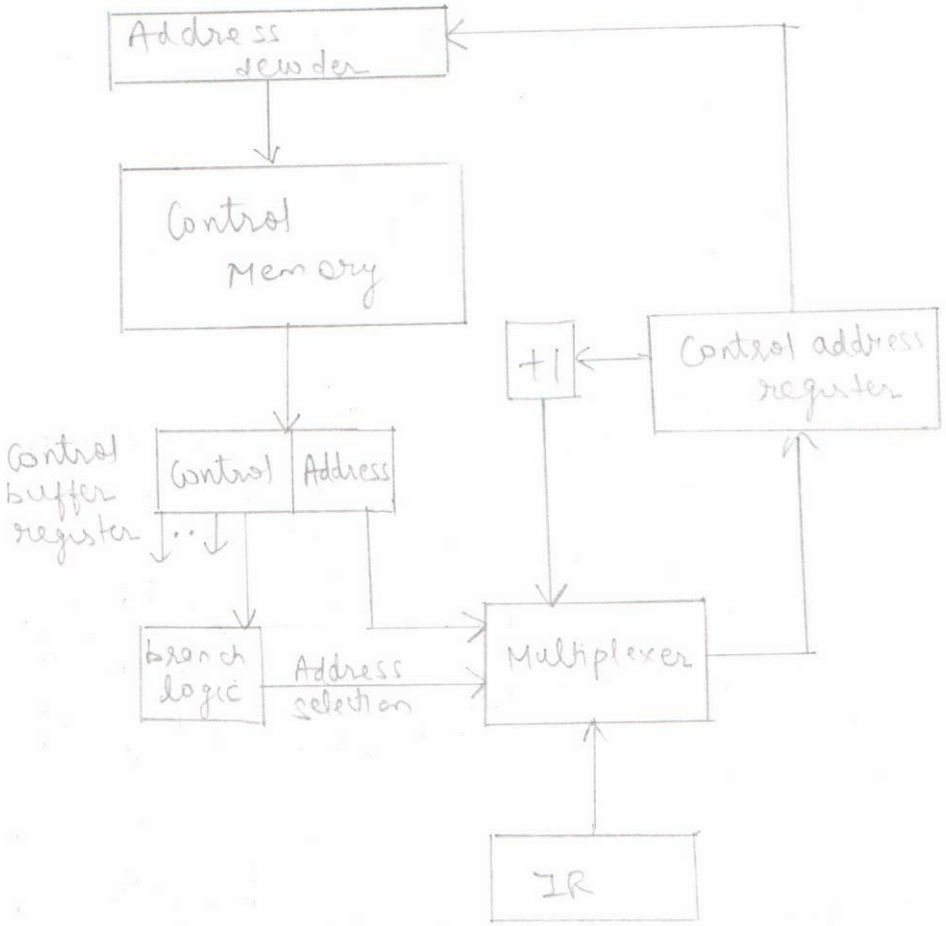


Branch Control logic : Two address fields

* In this approach, multiplexer is provided that serves as a destination for both address fields plus the instruction register.
* Based on an address-selection input, the multiplexer transmits either the opcode or one of two addresses to control address register (CAR).
* The CAR is decoded to produce next micro-instruction address.
* The address-selection signals are provided by branch logic module whose input consists of control unit flags plus bits from control portion of micro-instruction.

Disadvantage :- requires more bits in micro-instruction than other approaches.

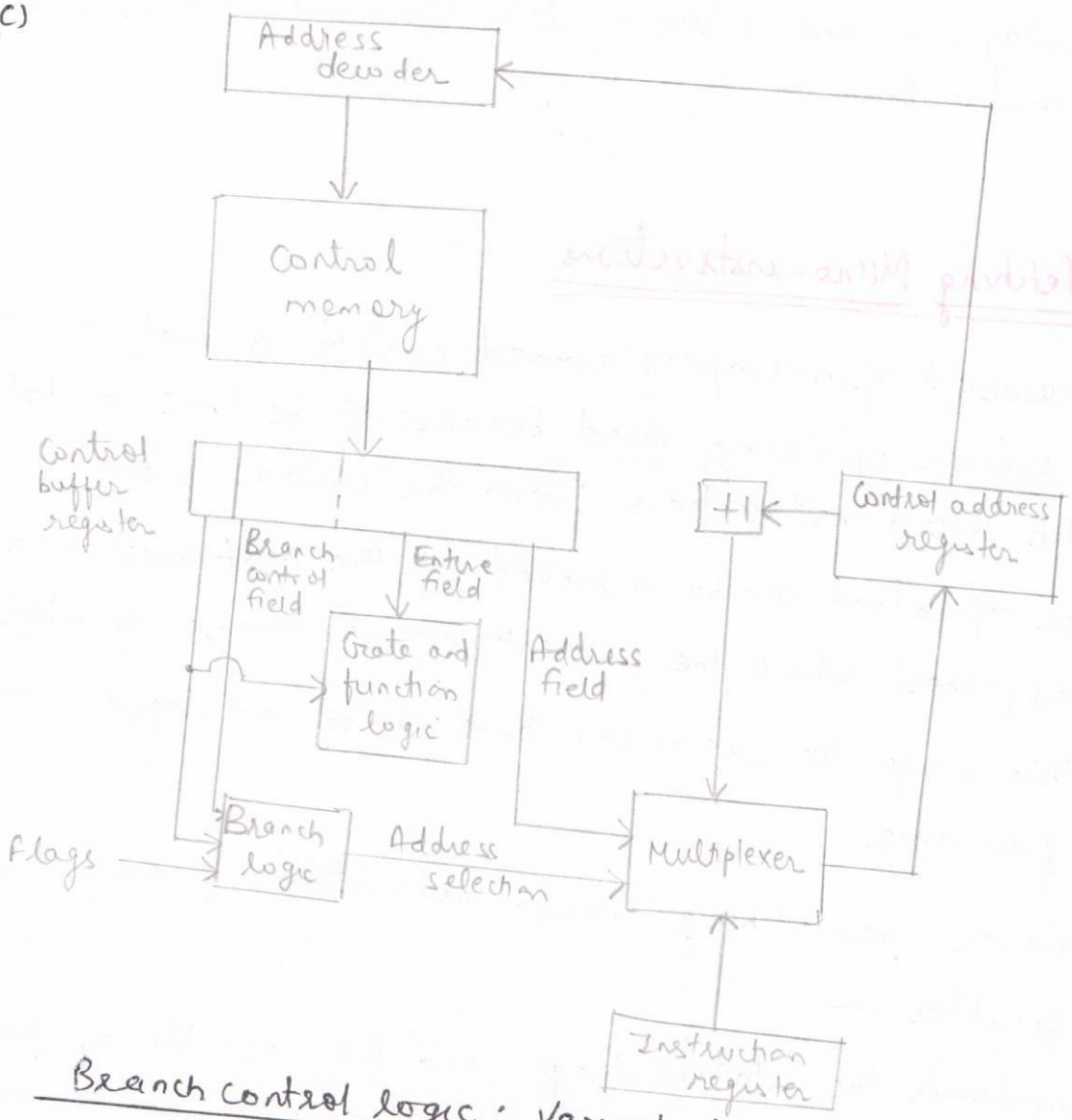
(b)



Branch Control logic : Single address field

* In this approach, the options for next address are:
  (i) Address field
  (ii) Instruction register code
  (iii) Next sequential address

(c)



Branch control logic: Variable format

* In this approach, one bit designates which format is being used.
* In one format, the remaining bits are used to activate control signals.
* In other format, some bits drive the branch logic module, and the remaining bits provide the address.

- With the first format, the next address is either the next sequential address or an address derived from instruction register.
- With second format, either a conditional or unconditional branch is being specified.

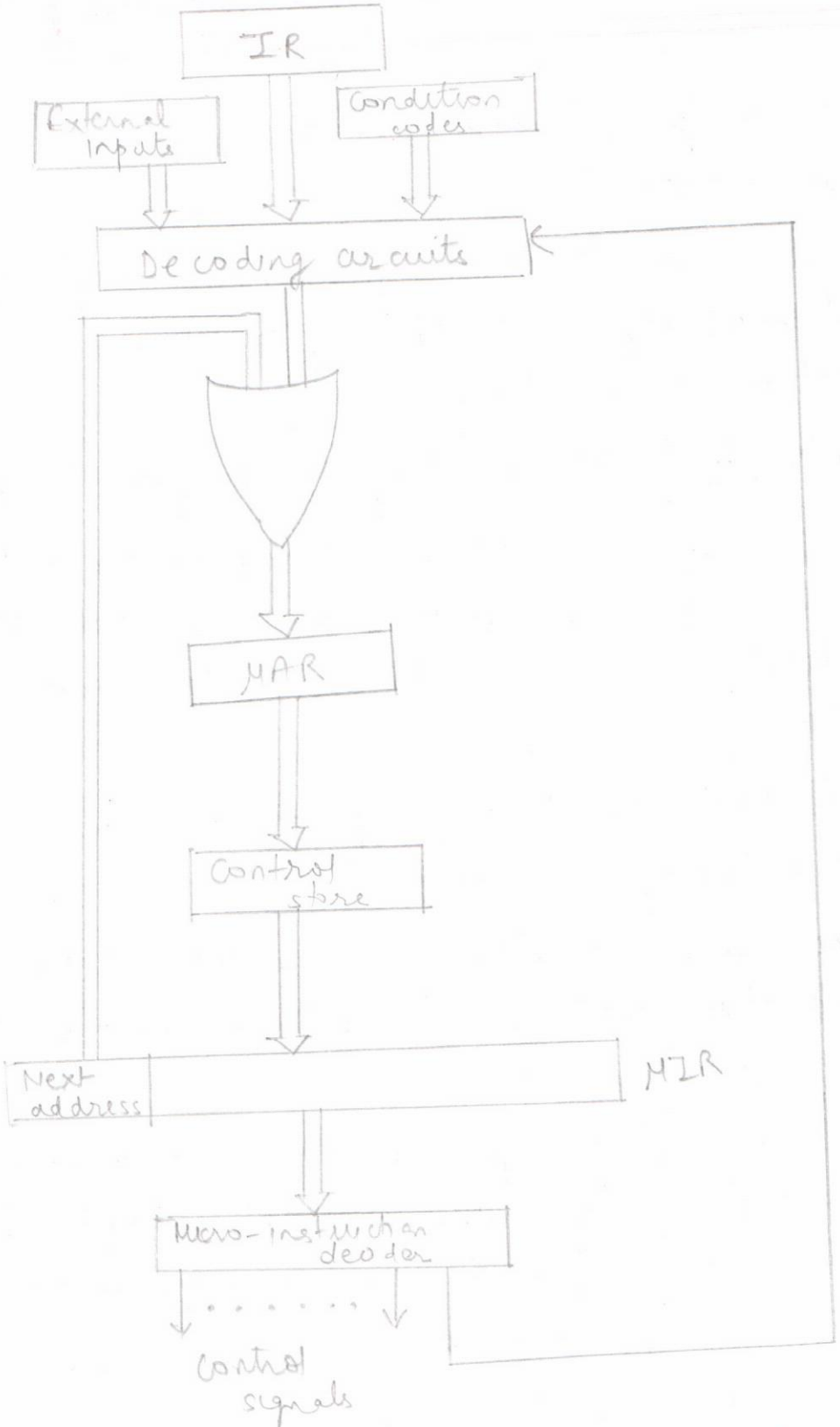
Disadvantage :- one entire cycle is consumed with each branch micro-instruction.

Pre-fetching Micro-instructions

* One drawback of microprogrammed control is that it leads to a slower operating speed because of the time it takes to fetch micro-instructions from the control store.
* Faster operation can be achieved if the next micro-instruction is prefetched while the current one is being executed.
* In this way, the execution time can be overlapped with the fetch time.
* However, prefetching micro-instructions presents some difficulties —
  Sometimes the status flags and the results of the currently executed microinstruction are needed to determine the address of next micro-instruction. Thus, straight forward prefetching occasionally prefetches a wrong micro instruction.
* In these cases, fetch must be repeated with the correct address, which requires more complex hardware.
* However, disadvantages are minor, and the prefetching technique is often used.

Micro-instructions with next-address fields

* Generally, micro-programs require several branch micro-instructions.
* These micro-instructions perform no useful operation in the datapath; they are needed only to determine address of next micro-instruction.
* This affects the operating speed of the computer.
* A powerful alternative to this problem is to include an address field as a part of every micro-instruction to indicate the location of next micro-instruction to be fetched.
* Its advantage is that separate branch micro-instructions are virtually eliminated.
* Since each instruction contains the address of the next instruction, there is no need for a counter to keep track of sequential addresses.
* So, the μPC is replaced with a micro-instruction address register (μAR) is which is loaded from the next-address field in each micro-instruction.
* Moreover, the next address bits can be fed through the OR gates to the μAR, so that the address can be modified on the basis of data in IR, external inputs and condition codes.
* The decoding circuits generate the starting address of a given micro-routine on the basis of OP code in IR.

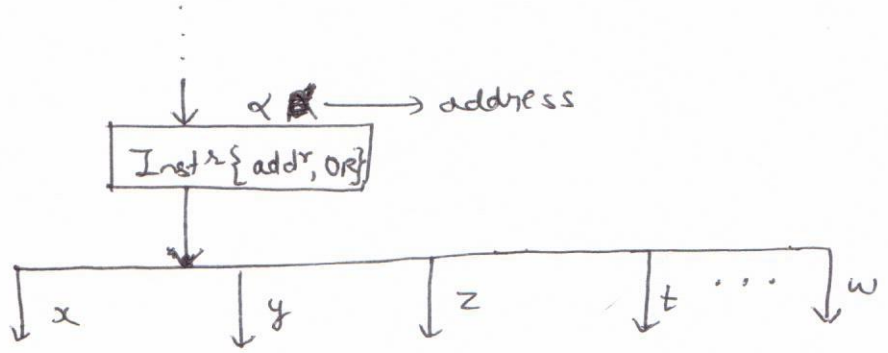


* However this approach comes at the expense of additional bits for the address field.

Wide Branch Addressing

* In general, the branches are not always made to a single branch address.
* This happens when simple micro-routines are combined by sharing common parts.
* At some point α, it may be needed to choose between actions specified by different addressing points.

e.g.



* The most efficient way to generate required branch address is using bit-ORing technique.
* Thus using bit-ORing technique, α address can be changed to one of the five possible address values x, y, z, t or w depending on the addressing mode used in the instruction.