Unit - III

Fundamental concepts of Basic Processing Unit - Performing ALU operation - Execution of complete instruction, Branch instruction - Multiple Bus Organization - Hardwired control - Generation of control signals - Microprogrammed control - Microinstruction - Microprogram sequencing - Microinstruction with next address field - Basic concepts of pipelining - Pipeline performance - Pipeline Hazards - Data hazards Methods to overcome data hazards - Instruction hazards - Hazards on Conditional and Unconditional branching - Control hazards - Influence of hazards on Instruction Sets.

FUNDAMENTAL CONCEPTS OF BASIC PROCESSING UNIT:

* To execute a program, the processor fetches instructions from memory. It uses registers like Instruction Register (IR) and Program Counter (PC).

* To execute an instruction, the processor has to perform the following steps.

1. Fetch the contents of the memory location pointed to by the PC. The content is interpreted as an instruction to be executed. Hence, they are loaded into IR.

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

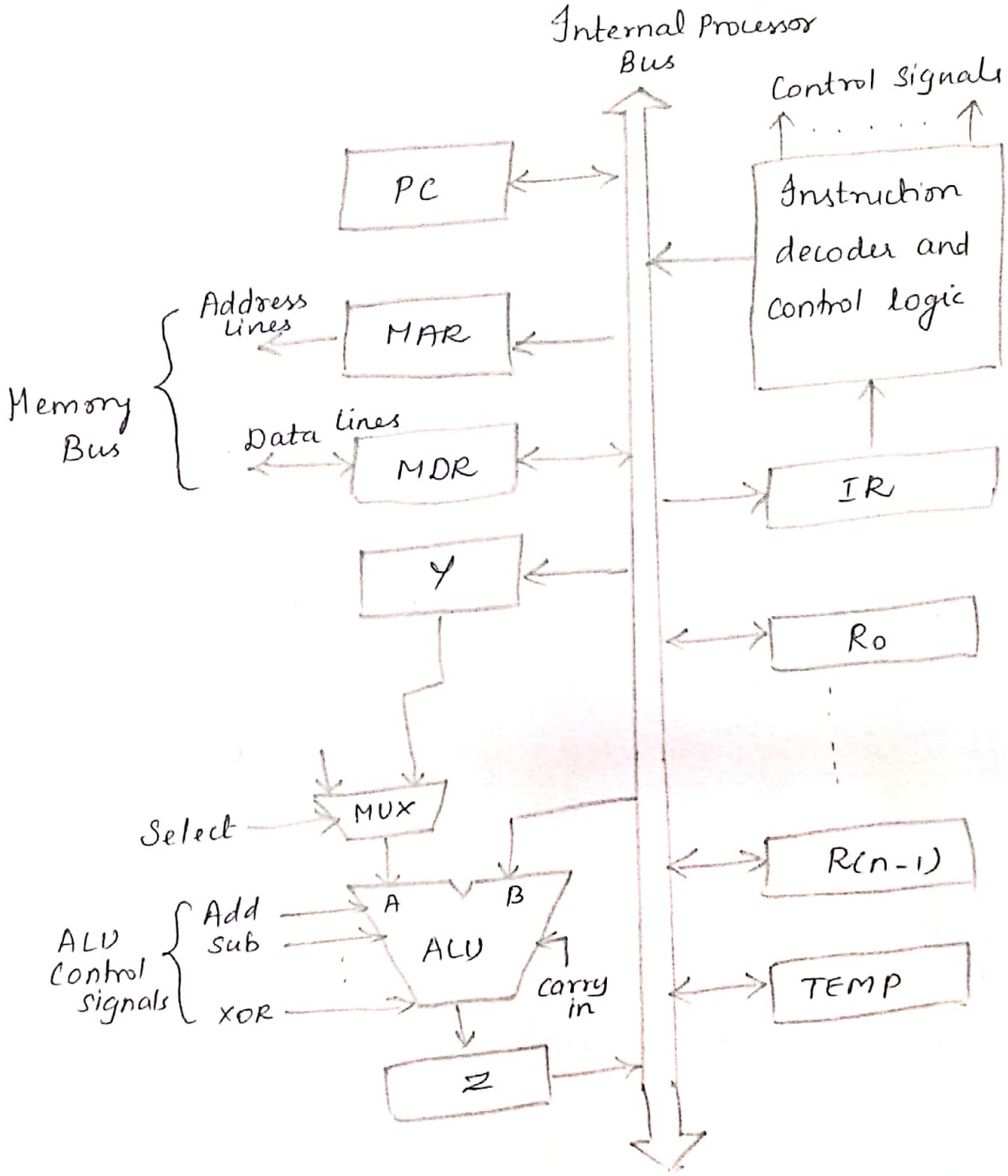
2. Assuming that the memory is byte addressable, increment the contents of PC by 4, that is,

$$PC \leftarrow [PC] + 4$$

3. Carry out the actions specified by the instruction in IR.

* The first two steps are referred to as the fetch phase and step 3 constitutes the execution phase

Single Bus Organization



* The figure shows an organization in which ALU and all registers are interconnected via single common bus.

* The data and address lines of external memory are connected to the internal processor bus via memory data ~~bus~~ register (MDR) and memory address register (MAR).

* MDR has 2 inputs and 2 outputs. Data may be loaded into MDR either from memory bus or from internal processor bus. Data stored in MDR can be placed in either bus.

* The input of MAR is connected to internal bus and output is connected to external bus.

* The gpr processor registers R0 through R(n-1) is used for general-purpose by the programmer. Three registers Y, Z & TEMP are used by the processor for temporary storage during execution of some instructions.

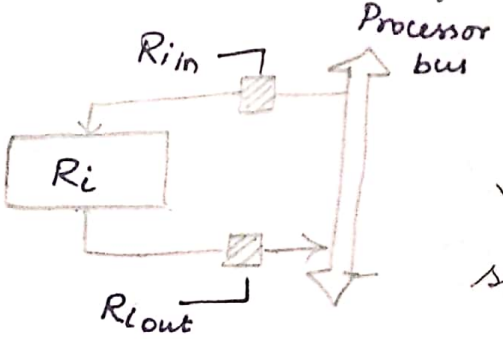
* The multiplexer MUX selects either the output of register Y (select Y) or a constant value 4 (select 4) to be provided as input A of ALU. Constant 4 is used to increment contents of program counter.

* The instruction decoder and control logic unit is responsible for implementing the actions specified by instruction loaded in IR register. The decoder generates control signals needed to select the registers involved and direct data transfer.

Register Transfers:

* Instruction execution involves a sequence of steps in which data are transfered from one register to another.

* For each register, 2 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 of register Ri are connected to bus via switches controlled by the signals Riin and Riout, respectively.

* When Riin is set to 1, the data on the bus are loaded into Ri. Similarly, when Riout is set to 1, contents of register Ri are placed on the bus.

* While Riout is equal to 0, the bus can be used for transferring data from other registers.

* To transfer the contents of register R_1 to R_4.

- Enable the output of register R_1 by setting R1 out to 1. This places the contents of R_1 on processor bus.

- Enable the input of register R_4 by setting R4in to 1. This loads data from the processor bus into register R_4.

* All operations and data transfers within the processor take place within time periods defined by the processor clock

PERFORMING AN ALU OPERATION:

* ALU is a combinational circuit that has no internal storage. It performs arithmetic and logic operations on the 2 operands applied to its A and B inputs.

* One of the operands is the output of multiplexer MUX and the other operand is obtained directly from the bus. The result produced by ALU is stored temporarily in register Z.

* Therefore a sequence of operations to add the contents of register R_1 to those of register R_2 and to store result in reg R_3

1) R1out , Yin

2) R2out , SelectY, Add, Zin

3) Zout , R3in

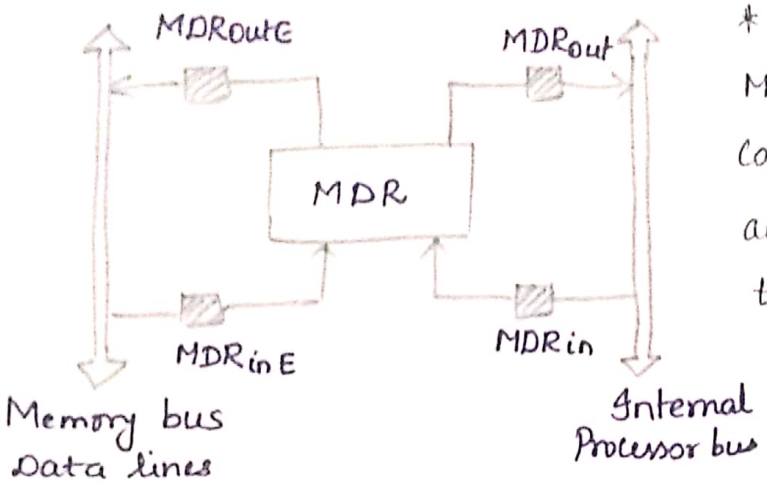
* The signals whose names are given in any step are activated for the duration of the clock cycle corresponding to that step

Fetching a Word from Memory

* To fetch a word of information (data/instruction) from memory, the processor has to specify the address of the memory location where this information is stored and request a Read operation.

* It involves registers MAR and MDR, where the address has to be stored in MAR and the read information is MDR

* The connections for register MDR are illustrated in figure



* It has 4 control signals: MDRin and MDRout control the connection to the internal bus, and MDRinE and MDRoutE control the connection to external bus.

* During 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.

* To accomodate the variability in response time, processor waits until it receives an indication that the requested Read operation has been completed. A control signal called Memory Function Completed (MFC) is used for this purpose.

* The addressed device sets MFC signal to 1 to indicate that requested Read/write is completed.

Eg: Move (R1), R2. The actions needed to execute this instruction:

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

* These actions may be carried out as seperate steps, but some can be combined into a single step.

* Each action can be completed in one clock cycle, except action 3 which requires one or more clock cycles, depending on the speed of the addressed device.

* The memory read operation requires 3 steps, which can be described by the signals being activated as follows

1. $R1_{out}$, MARin, Read
2. MDR_{inE}, WMFC
3. MDR_{out}, $R2_{in}$

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

Storing a Word in Memory:

* The desired address is loaded into MAR. The data to be written are loaded into MDR and a Write command issued.

eg. Move R2, (R1)

1. $R1_{out}$, MARin
2. $R2_{out}$, MDRin, Write
3. MDR_{outE}, WMFC.

* The processor remains in step 3 until the memory operation is completed and an MFC response is received.

EXECUTION OF COMPLETE INSTRUCTION:

* Consider the instruction

Add (R3), R1

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

* Executing the instruction requires the following actions:

1. Fetch the instruction
2. Fetch the operand
3. Perform the addition.
4. Load the result into R1.

* The sequence of control steps required to perform those operations for single-bus organization.

Step	Action
1	PCout, MARin, Read, Select 4, Add, Zin
2	Zout, PCin, Yin, WMFC
3	MDRout, IRin
4	R3out, MARin, Read
5	R1out, Yin, WMFC
6	MDRout, Select Y, Add, Zin
7	Zout, R1in, End

Step 1:

* Instruction fetch operation is initiated by loading contents of PC into MAR and sending a Read signal request to the memory
* Select4 signal causes the MUX to select constant 4 which is added to operand in input B, which is contents of PC and result is stored in register Z.

Step 2:

* The updated value is moved from reg. Z to PC, while waiting for the memory to respond.

Step 3:

* The word fetched from memory is loaded in IR

Steps 1 through 3 constitue instruction fetch phase which is same for all the instructions.

* The instruction decoding circuit interprets the contents of IR and enables control circuitry to activate the control signals for step 4 through 7, which constitutes the execution phase

Step 4:

* Contents of register R3 are transferred to MAR and a memory read operation is initiated.

Step 5:

* The contents of R1 are transferred to reg Y

Step 6:

* When read operation is completed, the memory operand is

available in MDR. MDR is gated to bus, and thus to B input of ALU. Register Y is selected as second input to ALU by select Y

*The added value is stored in register Z.

Step 7:

*The contents of Z are transferred to R1.

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

BRANCH INSTRUCTIONS:

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

*The control signals to implement an unconditional branch.

Step	Action
1	PCout, MARin, Read, Select4, Add, Zin
2	Zout, PCin, Yin, WMR
3	MDR_{out}, IRin
4	offset-field-of-IR_{out}, Add, Zin
5	Zout, PCin, End

*Step 1 through 3 constitutes instruction fetch phase

*The offset value is extracted from IR by decoding circuit,

*Value of PC is available in Y register. The offset X is gated into bus in step 4. and an addition operation is performed.

*The result which is branch target address, is loaded into PC in step 5.

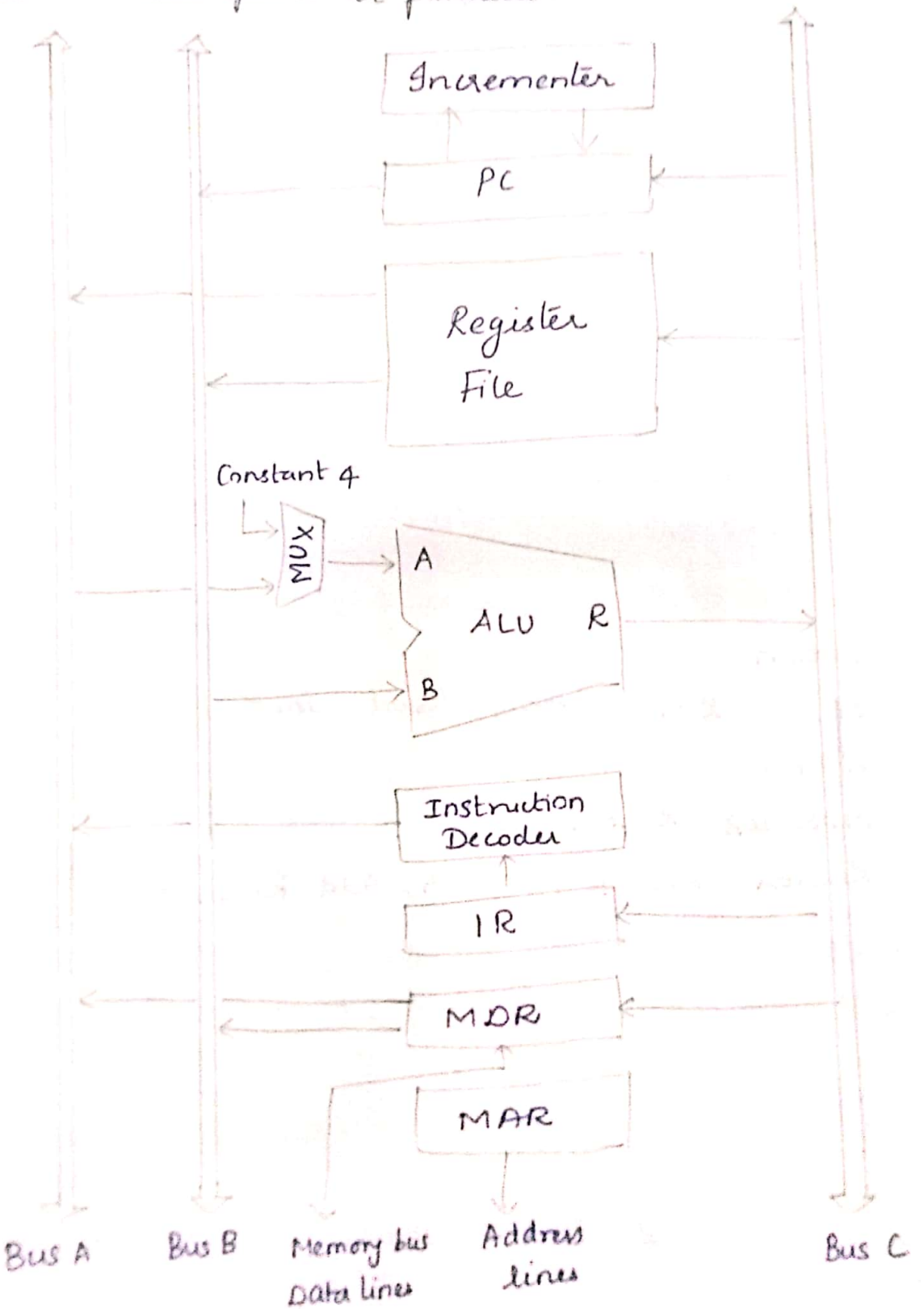
*Conditional branch: For a Branch_on_negative (Branch<0), step 4 is replaced with

Offset-field-of-IR_{out}, Add, Zin, If N=0 then End

Thus, if N=0 the processor returns to step 1. If N=1, step 5 is performed to load PC with the branch target address.

MULTIPLE BUS ORGANIZATION :

* The control sequences 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 provide multiple internal paths that enable to several transfers to take place in parallel



THREE BUS ORGANIZATION OF THE DATAPATH

* All general-purpose registers are combined into a single block called the register file. The register file have three ports. There are 2 outputs, allowing contents of 2 different registers to be accessed simultaneously and place the contents on buses A and B.

* The third port allows the data on bus C to be loaded into a third register during the same clock cycle.

* Buses A and B are used to transfer the source operands to A and B inputs of the ALU, where ALU operation performed. Result is transferred to destination over bus C.

* If needed, ALU may simply pass one of its 2 input operands unmodified to bus C. ALU control signals for such operation R=A or R=B.

* The incrementer unit, which is used to increment PC by 4. The constant 4 at the ALU input multiplexer is used to increment other addresses such as memory addresses in LoadMultiple and StoreMultiple instructions.

Eg: Add R_4, R_5, R_6.

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

Step 1: Contents of PC passed through ALU, using R=B control signal and loaded into MAR. Read signal initiated. Content of PC incremented.

Step 2: Processor waits for MFC.

Step 3: Loads the data received into MDR and then transfers them to IR.

Step 4: The execution of instruction requires one clock cycle to complete.

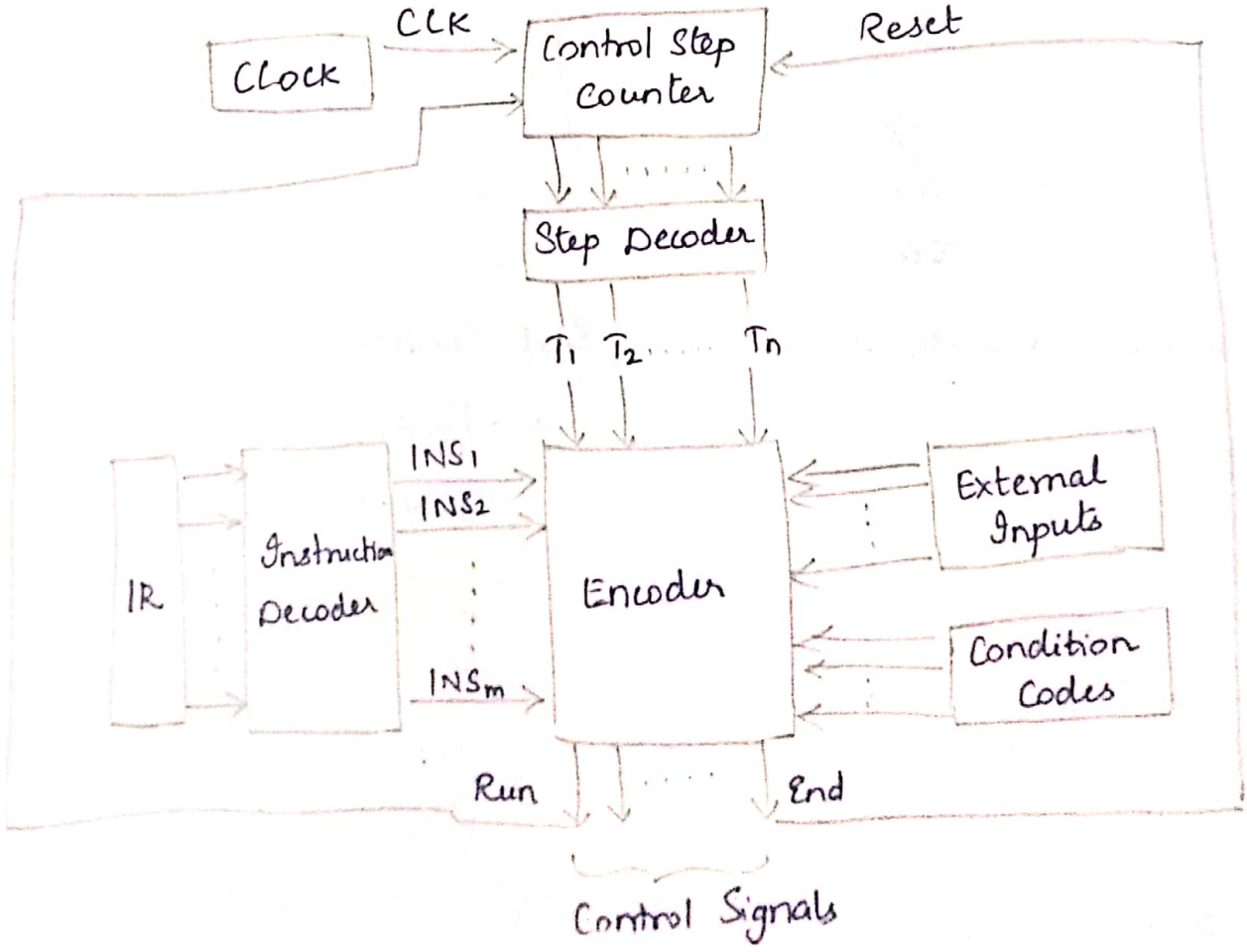
HARDWIRED CONTROL:

* To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.

* Approaches to generate control signals falls in 2 categories
  - → Hard wired control
  - → Microprogrammed control.

* Each step in the control sequence is completed in one clock cycle. A counter may be used to keep track of the control steps.

* The required control signals are determined by following information:
  - → Contents of control step counter
  - → Contents of instruction register
  - → Contents of condition code flags
  - → External input signals, such as MFC and interrupts requests



CONTROL UNIT ORGANIZATION

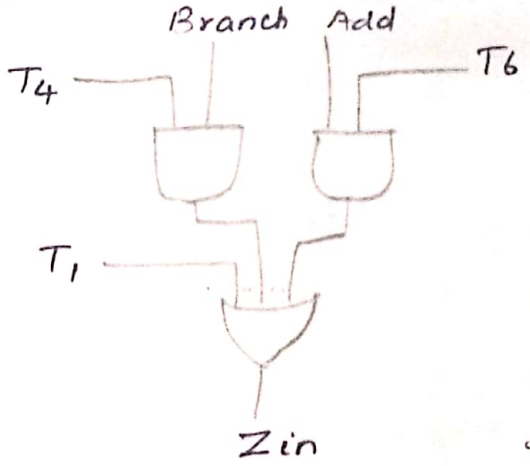
* The decoder/encoder block is a combinational circuit that generates the required control outputs, depending on the state of all its input.

* The step decoder provides a seperate signal line for each step or time slot in the control sequence.

* The output of Instruction decoder consists of a seperate line for each machine instruction. For any instruction in IR, one of the output lines INS1 through INSm is set to 1, and all other lines are set to 0.

* The input signals to the encoder block are combined to generate the individual control signals Y_{in}, PC_{out}, Add, ...

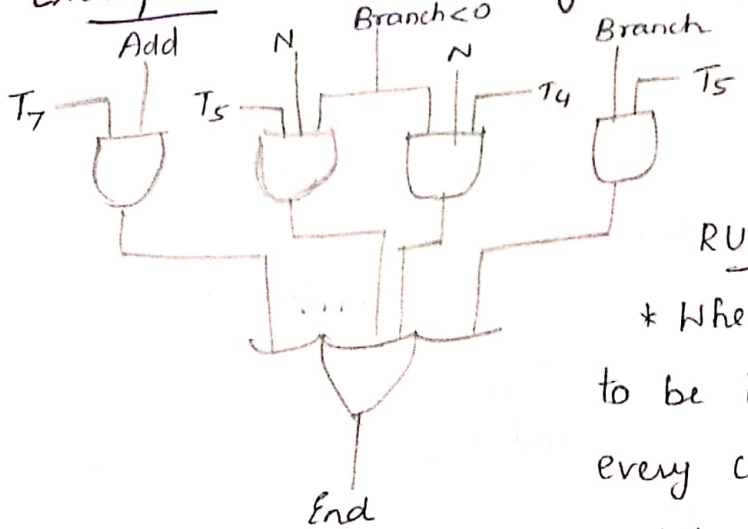
Example 1: Circuit to generate Z_{in} Signal.



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

The control signal Z_{in} 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.

Example 2: Circuit to generate End Control Signal.



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

RUN control Signal:

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

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

MICROPROGRAMMED CONTROL:

* In microprogrammed control, the control signals are generated by a program similar to machine language programs

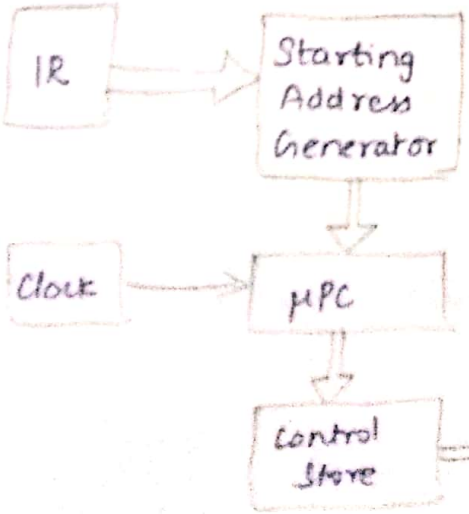
Micro Inst.	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	
1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	Z_{out}, PC_{in}, Y_{in}, WMFC
3	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	MDR_{out}, IR_{in}
4	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	$R3_{out}$, MAR_{in}, Read
5	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	$R1_{out}$, Y_{in}, WMFC
6	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	MDR_{out}, Select Y, Add, Z_{in}
7	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	Z_{out}, $R1_{in}$, End

* A control word is a word whose individual bits represent the various control signals. Each of the control steps in control sequence of an instruction defines a unique combination of 1's & 0's in the control word (CW).

* Select Y is represented by Select = 0 and Select 4 by Select = 1

* A sequence of control words corresponding to the control sequence of a machine instruction constitutes the microroutine for that instruction, and the individual control words in this microroutine are referred to as microinstructions.

* The microroutines for all the instructions in the instruction set of a computer are stored in a special memory called the control store



* The control unit can generate control signals for any instruction by sequentially reading the CWs of the corresponding microroutine from control store

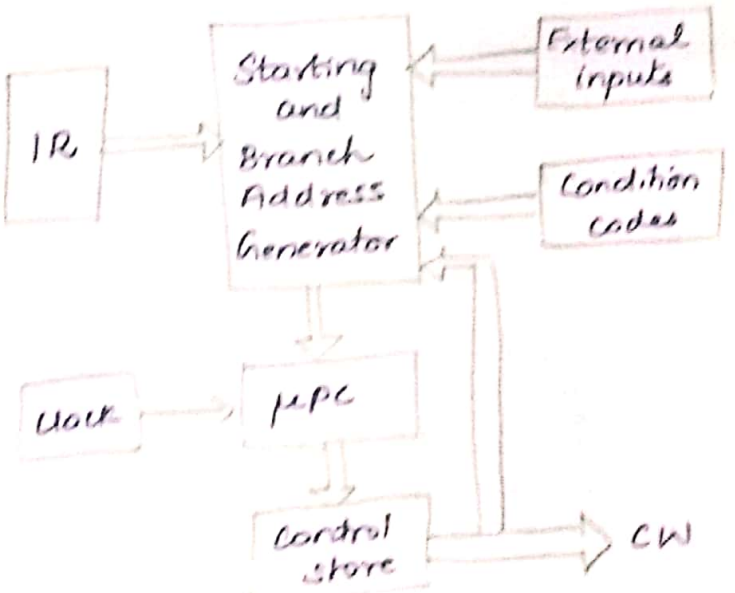
* The control unit is organized as shown in figure

* To read the control words from the control store, a micro PC (μPC) is used.

* Every time a new instruction is loaded into the IR, the output of block 'starting address generator' is loaded into μPC
* The μPC is then automatically incremented by the clock, causing successive microinstructions to be read from control store

Control Unit with Branching:

* There arises situation when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action An approach is to use conditional branch microinstructions
* To support microprogram branching, the organization of the control unit should be modified as shown in figure



* The starting address generator becomes the starting and branch address generator This block loads a new address into the μPC when a microinstruction instructs it to do so
* To allow implementation of a conditional branch, inputs to the block consists of the external inputs and condition codes as well as the contents of IR
* In this control unit, the μPC is incremented every time a new microinstruction is fetched from the microprogram memory, except in the following situations

1 When a new instruction is loaded into IR, the μPC is loaded

with the starting address of the microroutine of that instruction.

2. When a Branch microinstruction is encountered and the branch condition is satisfied, the μPC is loaded with the branch address.

3. When an End microinstruction is encountered, the μPC is loaded with the address of first CW in the microroutine for the instruction fetch cycle.

MICROINSTRUCTION:

* A straight forward way to structure microinstructions is to assign one bit position to each control signal. However, it has a drawback - assigning individual bits to each control signal results in long microinstructions because the number of required signals is usually large.

* Consider a simple processor (Single-bus) and assume it contains only 4 registers $R0$, $R1$, $R2$ and $R3$. Assume ALU supports 16 functions. So control signals are required to gate the register connections and ALU functions.

* Additional control signals are needed, including Read, Write, Select, WMFC and End signals. Totally, 42 signals are needed.

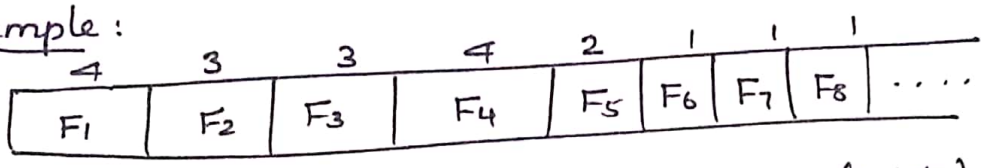
* By using simple encoding scheme, the length of microinstruction can be reduced.

* Most signals are not needed simultaneously and many signals are mutually exclusive. For eg, only one function of ALU can be activated at a time. The source for a data transfer must be unique because it is not possible to gate the contents of 2 different registers onto bus at same time.

* This suggests that signals can be grouped so that all mutually exclusive signals are placed in the same group. Thus, atmost one microoperation per group is specified in any microinstruction. Hence, a binary coding scheme to represent signals within a group can be used.

* For eg, 4 bits suffice to represent the 16 available functions in ALU. Register output control signals can be placed in a group consisting of PCout, MDRout, Zout, Offsetout, R0out, R1out, R2out, R3out and TEMPout. Any one can be selected by unique 4-bit code.

Example:



F1 (4 bits)
0000 : No transfer
0001 : PCout
0010 : MDRout
0011 : Zout
0100 : R0out
0101 : R1out
0110 : R2out
0111 : R3out
1010 : TEMPout
1011 : Offsetout

F2 (3 bits)
000 : No transfer
001 : PCin
010 : IRin
011 : Zin
100 : R0in
101 : R1in
110 : R2in
111 : R3in

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

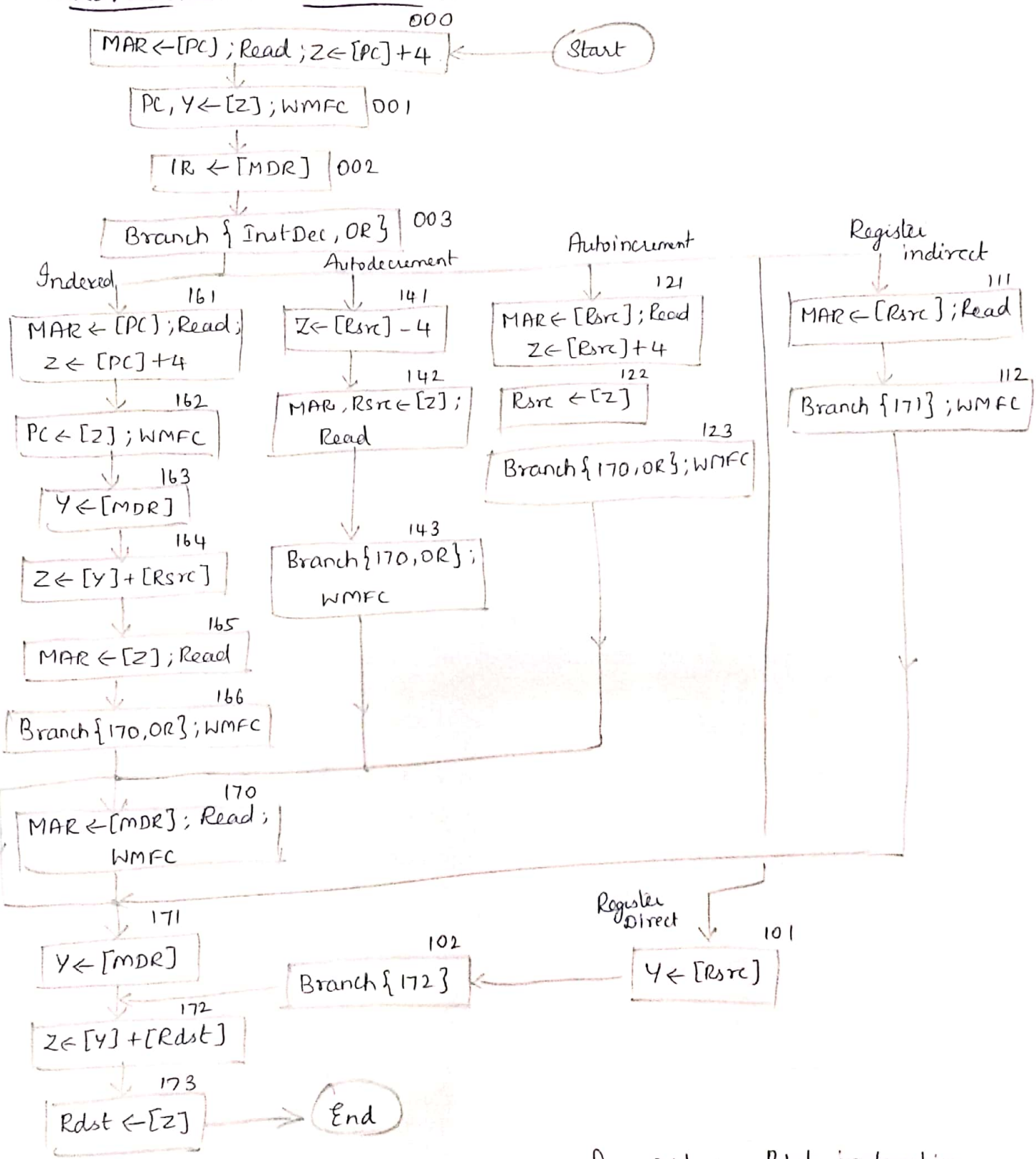
F8 (1 bit)
0 : Continue
1 : End

* Highly encoded schemes that use compact codes to specify only small number of control functions in each microinstruction are referred to as vertical organization.

* The minimally encoded scheme, in which many resources can be controlled with a single microinstruction, is called horizontal organization.

* The horizontal approach is useful when higher operating speed is desired and when machine structure allows parallel use of resources. The vertical approach results in considerably slower operating speeds because more microinstructions are needed to perform the desired control functions.

MICROPROGRAM SEQUENCING:



Flowchart of a microprogram for Add src, Rdst instruction

* The task of microprogram sequencing is done by microprogram sequencer. Two important factors must be considered while designing the microprogram sequencer.

1) The Size of the microinstruction
2) The address generation time.

* The size of the microinstruction should be minimum so that the size of control memory required to store microinstructions is also less. This reduces cost of control memory.

* With less address generation time, microinstruction can be executed in less time resulting better throughput.

* During execution of a microprogram the address of the next microinstruction to be executed has 3 sources.

1) Determined by instruction register
2) Next sequential address
3) Branch.

Disadvantage of Microprogrammed branching:

1) Having a seperate microroutine for each machine instruction results in large total number of microinstructions and a large control store.
2) Execution time is longer because it takes more time to carry out the required branches.

* Consider the instruction Add src, Rdst; which adds the source operand to the contents of Rdst and places the sum in Rdst. Let source-operand can be specified in following addressing modes

a) Indexed b) Autoincrement c) Autodecrement d) Register indirect e) Register direct

* Each box in the chart corresponds to a microinstruction that controls the transfers and operations indicated within the box. The microinstruction is located at the address indicated by the octal number (001,002).

Branch Address Modification Using Bit-Oring:

* The branch address is determined by ORing particular bit or bits with the current address of microinstruction. Eg. If the current address is 170 and branch address is 171 then the branch address can be generated by ORing 01 (bit 1), with the current address.

* It is necessary to choose between direct and indirect addressing modes. If indirect mode is specified in the instruction, then the

microinstruction in location 170 is performed to fetch the operand from the memory.

* If direct mode is specified, this fetch must be bypassed by branching immediately to location 171. The most efficient way to bypass microinstruction 170 is to have bit-ORing of current address 170 & branch address 171.

Wide Branch Addressing:

* The instruction-decoder (InstDec) generates the starting address of the microroutine that implements the instruction that has been loaded into IR.

* The register IR contains the Add instruction, for which the InstDec generates the microinstruction address 101.

* The source-operand can be specified in any of addressing mode. The bit-ORing technique can be used to modify the starting-address generated by the instruction-decoder to reach the appropriate path.

Use of WMFC:

* WMFC signal is issued at location 112 which causes a branch to the microinstruction in location 171. WMFC may take several clock cycles to complete.

Add (Rsrc)+, Rdst:

* It adds Rsrc content to Rdst content, then stores the sum in Rdst and finally increments Rsrc by 4.

* In bit 10 and 9, bit-patterns 11, 01, 10 and 00 denote Indexed, auto-increment, auto-decrement and register modes respectively. Bit 8 is used to specify the indirect version.

* There are 2 stages of decoding:

1) The microinstruction field must be decoded to determine that an Rsrc or Rdst register is involved.

2) The decoded output is then used to gate the contents of Rsrc or Rdst fields in IR into second decoder, which produces gating signals for registers R0 to R15.

MICROINSTRUCTION WITH NEXT ADDRESS FIELD:

* The microprogram requires several branch microinstruction which perform no useful operation. They detract operating speed of computer

* The solution is to include an address-field as a part of every microinstruction to indicate the location of next microinstruction to be fetched. Thus, every μinst becomes a branch μinstruction.

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

Adv: Seperate branch microinstructions are virtually eliminated.

Disadv: Additional bits for the address field.

* There is no need for a counter to keep track of sequential address. Hence, μPC is replaced with μAR.

* The next address fields are fed through OR gate to μAR, so that address can be modified on the basis of data in IR, external inputs and condition-codes.

* The main function of microprogrammed control is to provide a means for simple, flexible and relatively inexpensive execution of machine instruction. Its flexibility in using machine's resources allows diverse classes of instructions to be executed.

* Emulation allows to replace obsolete equipment with more up-to-date machines.

* If the replacement computer fully emulates the original one, then no software changes have to be made to run existing programs. Emulation is easiest when the machines involved have similar architectures.

—x—