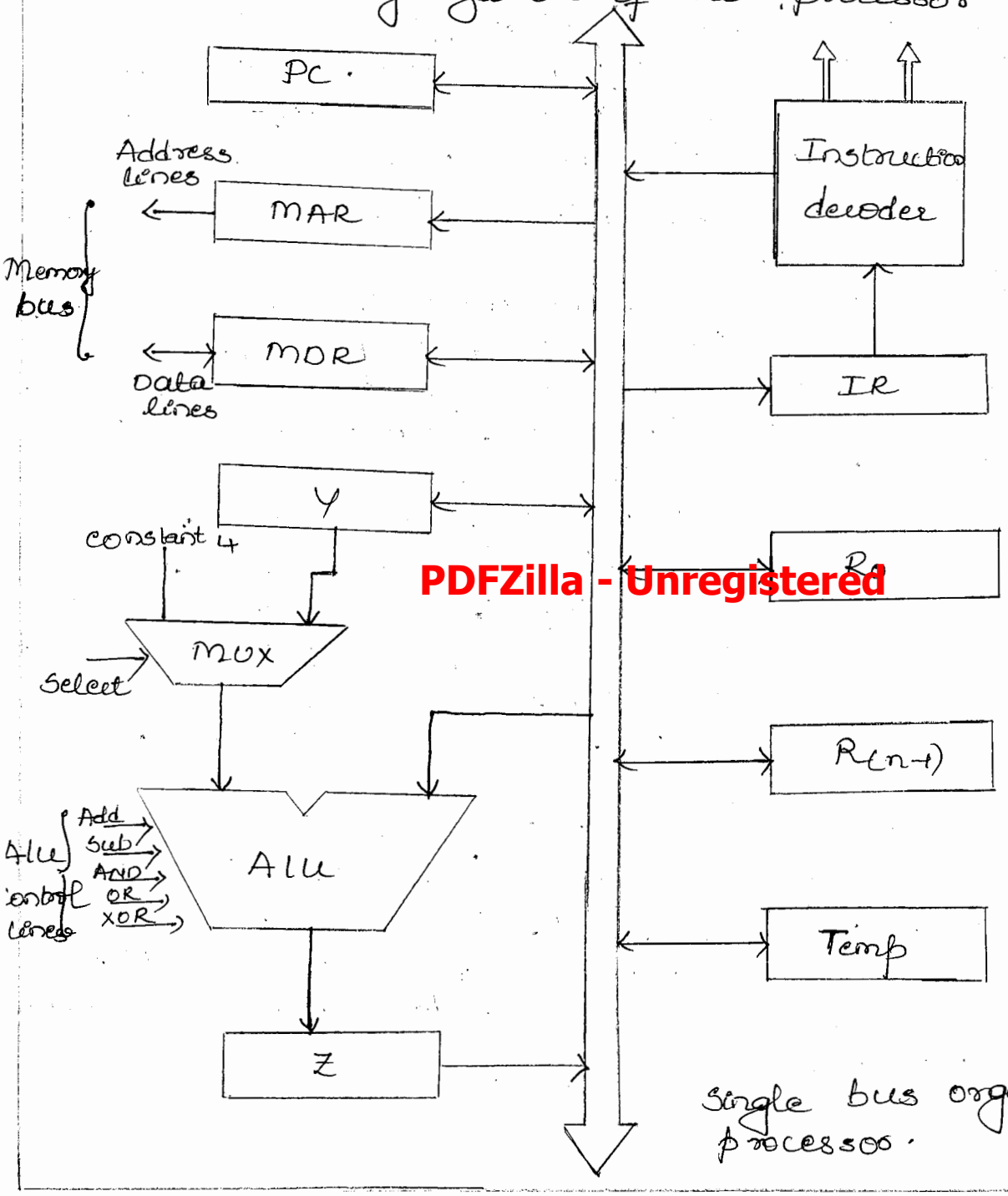
Some Fundamental Concepts

To execute a program processor fetches an instruction one after the other from memory sequentially unless a branch or jump instruction is encountered. To execute a instruction there are four stages, which are Fetch, Decode, Execute (Alu or logical operation), write Back. when the instruction is fetched from memory it gets decoded, to find the operands, meanwhile the content of Pc gets incremented to point to next instruction in memory. Once the operands are read it gets executed using Alu and the result is stored in the destination (write Back).

Internal organization of the processor



Single bus organization of processor.

In this organization all the registers, ALU and control unit is connected via a single bus. (This bus is the internal bus of the processor).

→ The external bus i.e data bus and address bus are connected to the processor bus via MAR and MDR registers.

→ MDR register is bidirectional. It has two inputs and two outputs. So data may be loaded into MDR by processor internal bus and by External memory bus & Data stored in MDR can be given to external memory bus and internal processor bus.

→ Register MAR has one input and one output. It gets the input from processor bus (address) and it sends the address to external address bus.

→ All the general purpose registers R0 to Rn-1 has one input and one output.

→ The three registers X, Y and Z are used by processor for temporary storage of data during execution. Register X recieves the data from bus but it cannot give the data to bus. Register Z gives the data to bus.

→ ALU has two inputs A & B. A is recieved from multiplexer, B is directly from bus.

→ It is a two input multiplexer. One input to a multiplexer is constant 4 another input is output of register Y. There are two select lines, Select 4 and Select Y. Select 4 is selected when content of PC has to be incremented. Select Y is selected during execution of instruction.

→ Output of ALU is directly gated to the register Z.

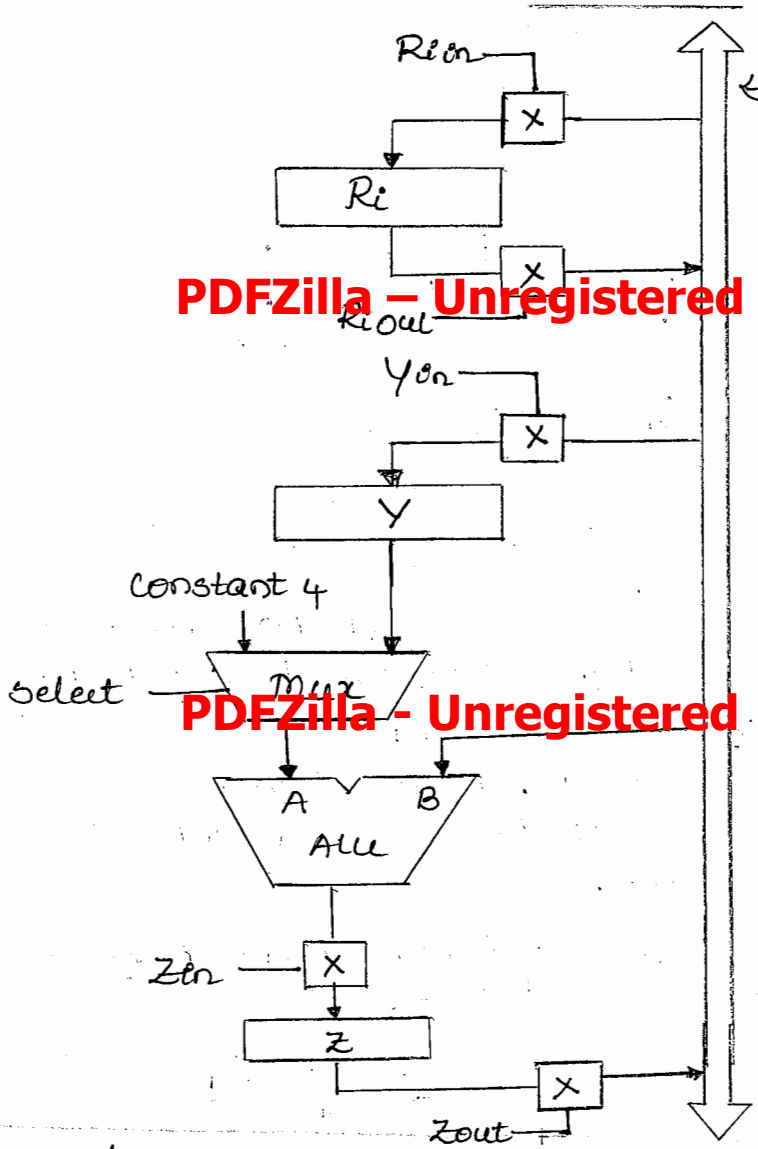
→ Instruction decoder and control unit is responsible for decoding the instruction which is present in the register IR. Hence the output of IR is directly connected to this block.

→ Decoder generates the control signals needed to select the registers involved and direct the transfer of data.

Register Transfers

An instruction execution involves a series of steps where the data is transferred from one to another.

Every register has two control signals to place the content of register on to the bus or load the content from bus to register.



→ The input and output of register are connected to bus via switches. The two signals for input and output are Rin and Rout.

→ When Rin is set to 1, the data on the bus is loaded into R.

→ When Rout is set to 1, data from register R is loaded on to bus.

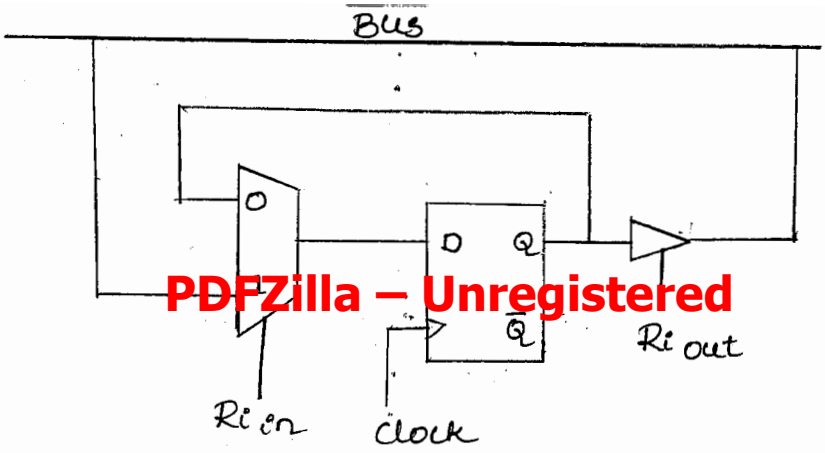
For example transfer the content of Register R_2 to R_4.

1) R_1 out is set to 1. This places the contents of R_1 onto bus.

2) R_4 in is set to 1. This loads the content from bus to R_4

All the operations and data transfers take place within the time periods defined by the processor clock. Registers are made of edge triggered flip flops.

Consider an implementation for one bit of the Register Ri.



Input and output for one register bit

A two input multiplexer is used to select the the data applied to the i/p of edge triggered D flip flop. The two select lines are R_{in0} and R_{in1}.

→ When Ri in is equal to 1, multiplexer selects the data on the bus. This data will be loaded into flip-flop at the rising edge of the clock.

→ When Ri in is zero, multiplexer feeds back the value currently stored in the flip flop.

→ Output of the flip-flop is connected to the bus via a tri-state gate.

→ When Ri out is equal to zero, its an open switch, the gate's output is in the high impedance state.

→ When Ri out = 1 the value (0 or 1) is placed on the bus. i.e the gate drives the bus to 0 or 1 depending on the value of Q.

Performing an Arithmetic operation

ex:- $R_3 = R_1 + R_2$ (we assume that instruction has been fetched)

→ Alu is a combinational circuit which has no internal storage
→ It performs logical and arithmetic operations on two operands applied to it A and B i/ps.
→ One of the operand is from o/p of multiplexer and other operand is directly obtained from bus.
→ Result of Alu operation is directly gated to the register Z.

The sequence for the instruction $R_3 = R_1 + R_2$ is as follows.

1) R_2 out, Y on
2) R_2 out, select Y, Add, Z en
3) Z out, R_3 in

These are the signals that gets generated. 3 steps occurs in 3 clock cycles.

In the first step content of R_1 is placed on the bus and it is loaded on to the register Y.

In the second step content of R_2 is placed on the bus and it is directly given to ALU as second input. Multiplexer selects the output of Y and gives it to ALU as first i/p. ALU add signal gets generated. o/p of ALU is directly stored on register Z.

In third step content of Z (result) will be placed on the bus and loaded onto the register R_3.

Fetching a word from Memory

Note:- It is not fetching an instruction. Instead it is to fetch an operand after decoding.

→ To fetch (read) a word from memory processor sends the address of memory location where the information is stored and then raises a Read request.

→ Address is sent to MAR register and control signal read gets generated. when the time period is complete data is stored in register MDR. From where it can be transfered to other registers inside the processor.

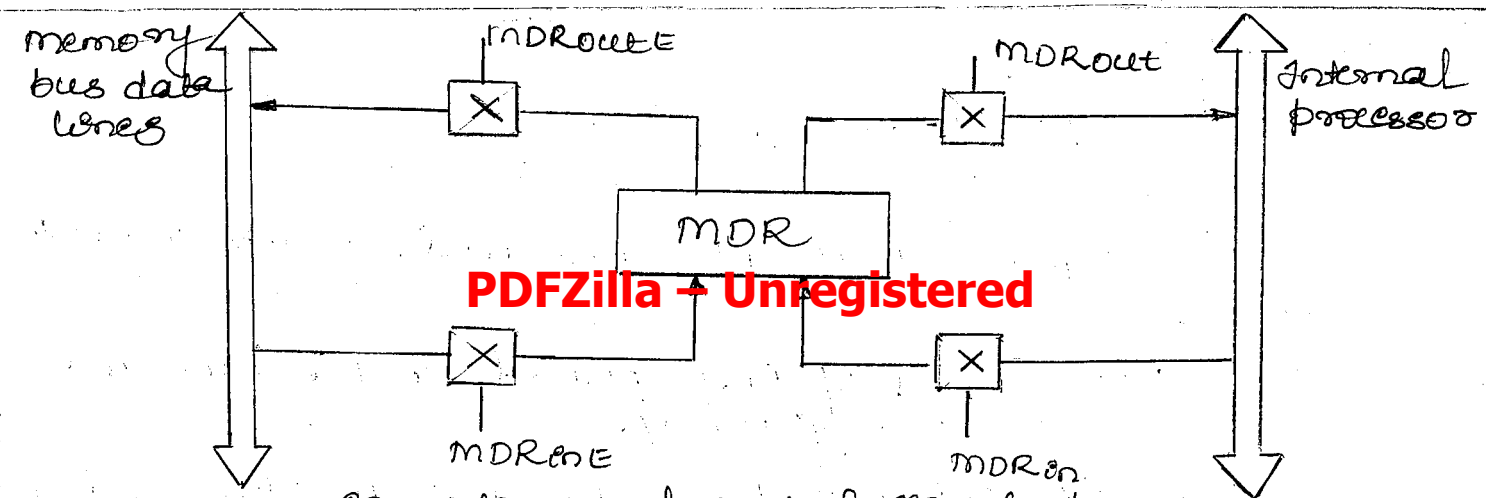
MDR has four control signals. MDRin and MDRout control the connection to internal bus. $MDRin_E$ and $MDROut_E$ control the connection to external bus.

MDRin = 1. Data from internal processor bus is loaded onto MDR

MDRout = 1 Data from MDR is loaded onto internal processor bus.

$MDRin_E$ = 1 Data from external bus is loaded onto MDR

$MDROut_E$ = 1 Data from MDR is loaded onto external memory bus.



connection and control signals for register MDR

Example: Move (R1), R2

The actions to be performed

1) Content of R1 (address) should be given to MAR. i,e MAR ← [R1]
2) Initiate a read operation.
3) wait for MFC (memory function complete) signal from memory (Addressed device always sends MFC signal). The generation of MFC signal means the data is present on MDR register.
4) Load MDR from
5) R2 ← [MDR]

Sequence of signals are as follows.

1) R1out, MARin, Read (MAR out is always enabled)
2) MDRinE, WMFC
3) MDRout, R2in

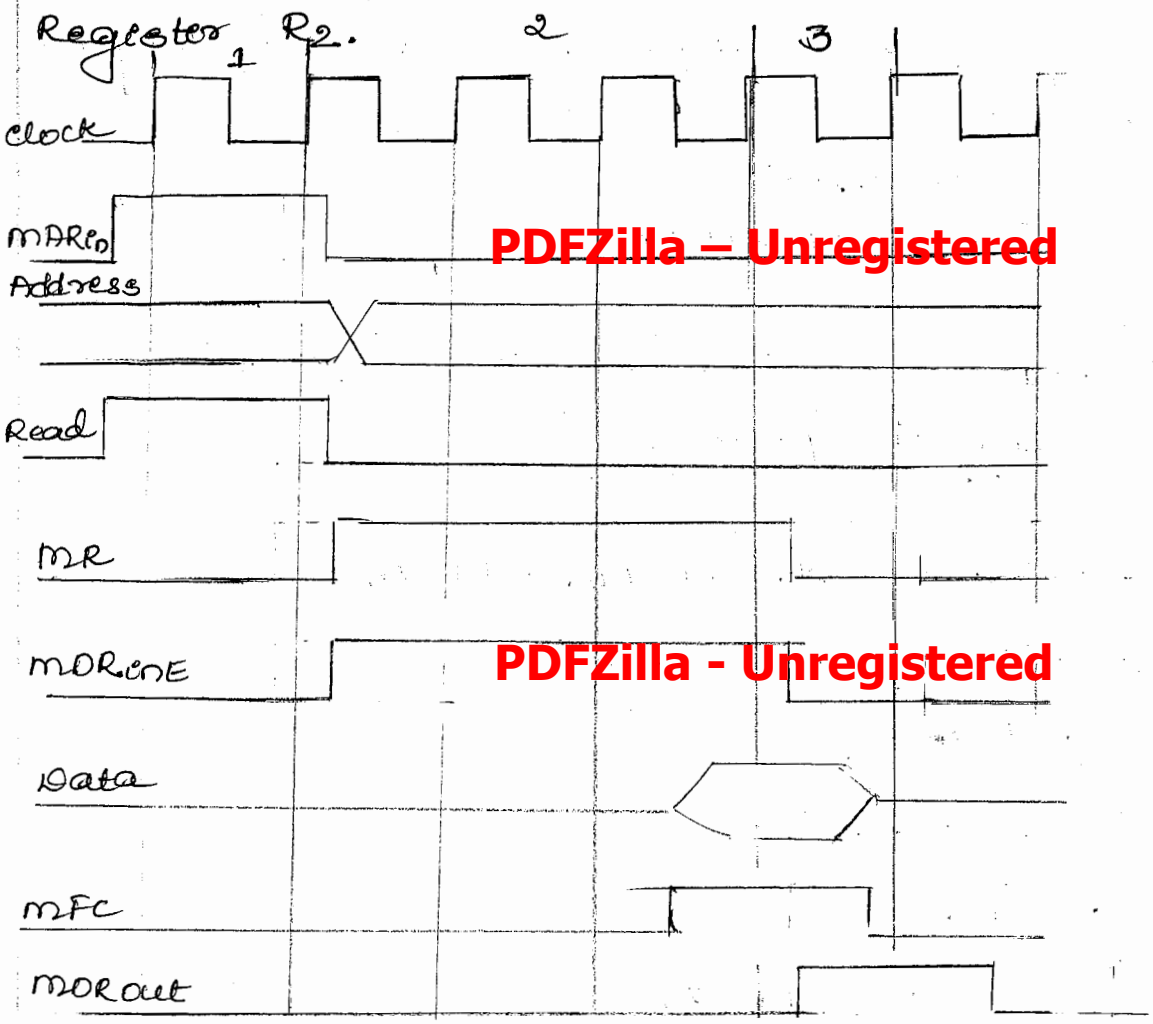
→ Content of R1 is placed on the bus which is recieved by MAR. Since MARout is always enabled o/p of MAR is recieved by external memory address bus. A read control signal is activated at the same time MAR is loaded.

→ This Read signal causes the bus interface circuit to generate an external command called as Memory Read i,e MR on the bus.

→ In the second step data is recieved from external bus into MDR and a wait MFC signal is activated

→ In third step, only after WMFC signal is generated (which means a valid data is present on MDR) content of MDR

is loaded on to internal processor bus. which is recieved by Register R_2.



Storing a word in Memory

Move $(R_2), (R_1)$

1) R_1out, MARin
2) R_2out, MDRin, Write
3) MDRoutE, WMFC

→ Content of R_1 (address) is loaded in MAR. The data to be written is loaded into MDR and write signal is initiated. In the third step the content from MDR is written to memory and processor waits until it recieves WMFC from memory (addressed device).

Execution of a complete Instruction

To execute an instruction 4 steps are required. They are

1) Fetch the instruction
2) Decode (read the operands)
3) Perform ALU operation
4) store the result.

Ex:- Add (R3), R1

Sequence of steps are as follows.

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

Note:- First three steps are for fetch phase which is common to all the instructions. Remaining steps are for Decoding, Execution and write back. One step occurs in one clock cycle.

1) In the first step, contents of PC (address of instruction in memory) are placed on bus, which goes onto MAR and Read signal gets activated. To increment content of PC, constant 4 is selected by mux, Add signal is generated. ALU performs addition on the I/Ps. one I/P is constant 4 and another I/P is content of PC which is available on Bus. Result is stored in Register Z.

2) In 2nd clock cycle updated value of PC which was in Z is placed on bus and recieved by PC and register Y. (It is saved in Y coz, this value is needed during Branch instruction) WMFC signal gets activated.

3) In third clock cycle content of MDR is placed in register IR

4) In fourth cycle content of R3 (address) is placed on MAR and read signal is activated. (Decoding step)

5) In fifth cycle content of R1 is placed on the bus, which is stored in Y and waits for MFC

6) In sixth cycle content of MDR is put on bus which becomes the 2nd i/p to ALU. First i/p is selected by mux using Select Y. Addition is performed & stored in Z. (Execution step)

7) In seventh cycle result is placed on bus, and stored in R. End signal causes a next instruction to be fetched.

Branch Instruction.

Branch instruction replaces the content of Pc with branch target address. The address is got by adding the offset to updated value of Pc. (-x+[PC])

Instruction fetch phase is same for all instructions. In fetch phase the updated contents of Pc is also loaded into register Y. (check step 2). Before the instruction is fetched Pc would have got the updated value. Then in the third step instruction will be loaded into IR.

If the instruction which was loaded into IR was a branch instruction then Pc should get the branch target address not the recently updated value. In this case the offset should be added with content of Pc. Content of Pc is on Y. First i/p to ALU is from multiplexer, it selects, Select Y and second i/p is from [illegible] which comes directly to ALU.

Offset is calculated by instruction decoder circuit and loaded on to bus.

ALU performs addition places the result on Z. Lastly content of Z is placed on the bus and recieved by Z.

1) Pcout, MARin, Read, Select4, Add, Zin

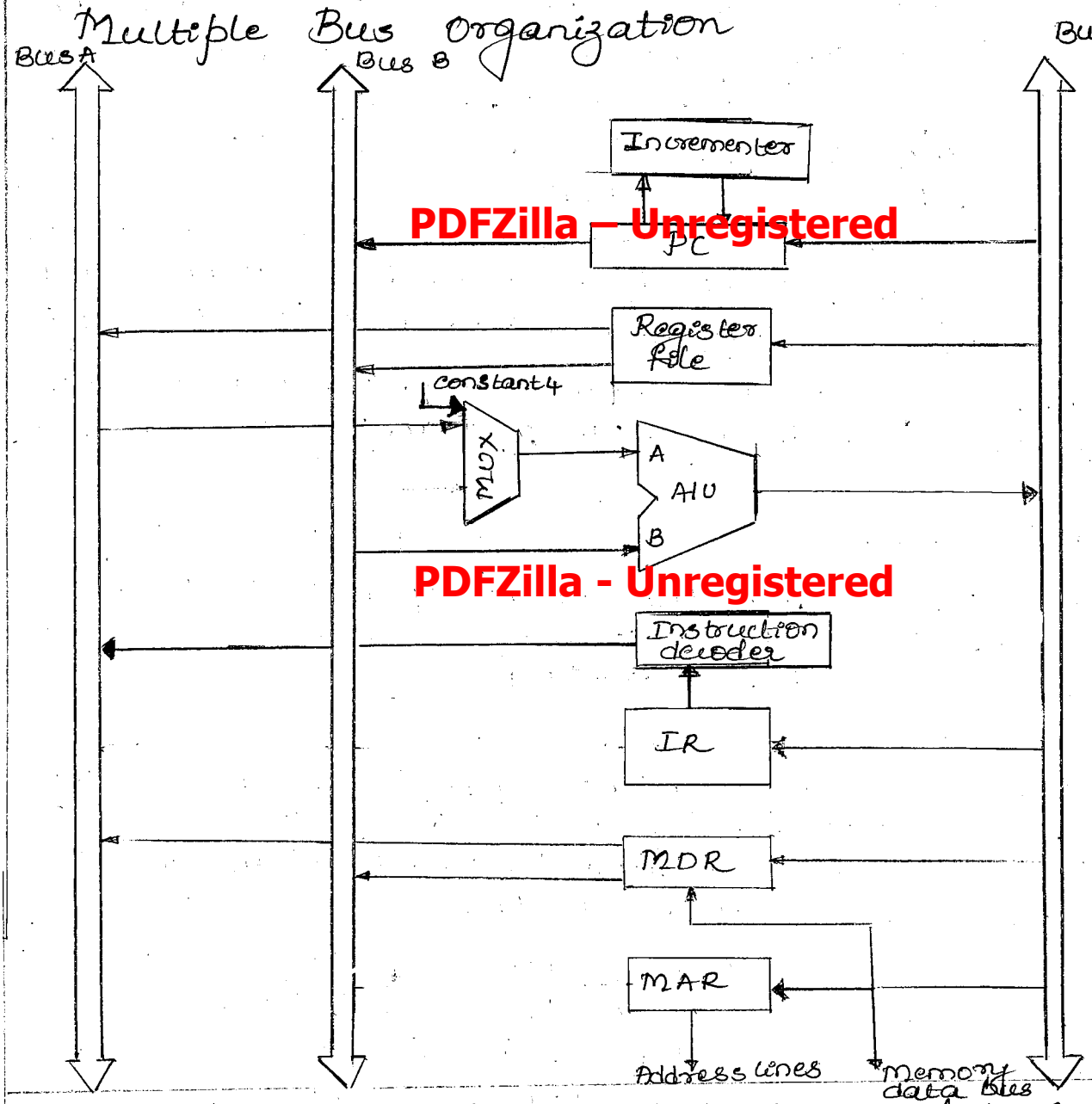
2) Zout, Pcin, Yin, WMFC

3) MDRout, IRin

4) Offset field of IRout, Add, Zin

5) Zout, Pcin, End

Multiple Bus Organization



In a single bus architecture only one data can be placed on a bus in one clock cycle which is time consuming. In modern processors a three bus architecture is used where more than one data can be transferred in 1 clock cycle. All the registers are grouped into a single register file. This file has two o/p port and one i/p port. Which means content of 2 registers can be placed on 2 bus (A and B). Third port allows the content from bus to be placed in regist Bus A and B are used to transfer source operands to A & B i/p of ALU. Result is transferred to destination using bus C.

Example:- Add R4, R5, R6

Steps	Action
1)	PCout, R=B, MARin, Read, IncPc
2)	WMFC
3)	MDRoutB, R=B, IRin
5)	R4outA, R5outB, selectA, Add, R6in, End

Note:- R=B means content on Bus B is given to Bus C
R=A means content on Bus A is given to Bus C

In this architecture an Incrementer unit is used which increments the contents of PC by 4. This unit eliminates the need to add 4 to PC by ALU. Constant 4 at ALU i/p mux is still useful to increment other address such as memory address on LoadMultiple and Store Multiple instructions.

In the Step1 to fetch an instruction Add R4, R5, R6, the content of PC is placed on Bus B. The signal R=B causes the content on B (which is content of PC) to be placed on Bus C. From Bus C, content is placed into MAR register. MAR is connected to external address bus. PC gets incremented.

In step 2 WMFC gets activated.

In step 3 content of MDR is placed on Bus B. This content is sent to Bus C by activating the signal R=B. From Bus C, content is placed into register IR.

In step 4 R4 content is placed on Bus A & R5 content placed on Bus B. R4 is given to ALU by mux, R5 is directly given to ALU. Result is stored on R6

Hardwired Control

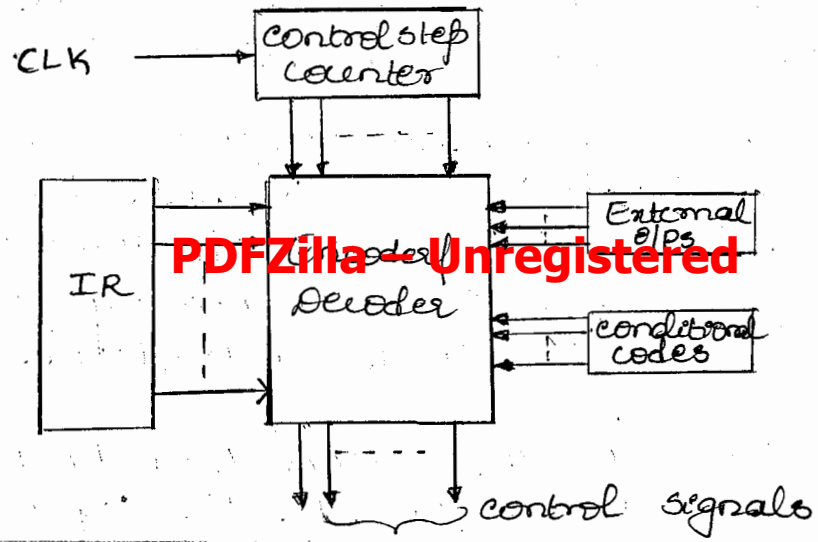
To execute instructions, processor must have some means of generating signals on a proper sequence. There are two approaches. 1) Hardwired control 2) Microprogrammed control.

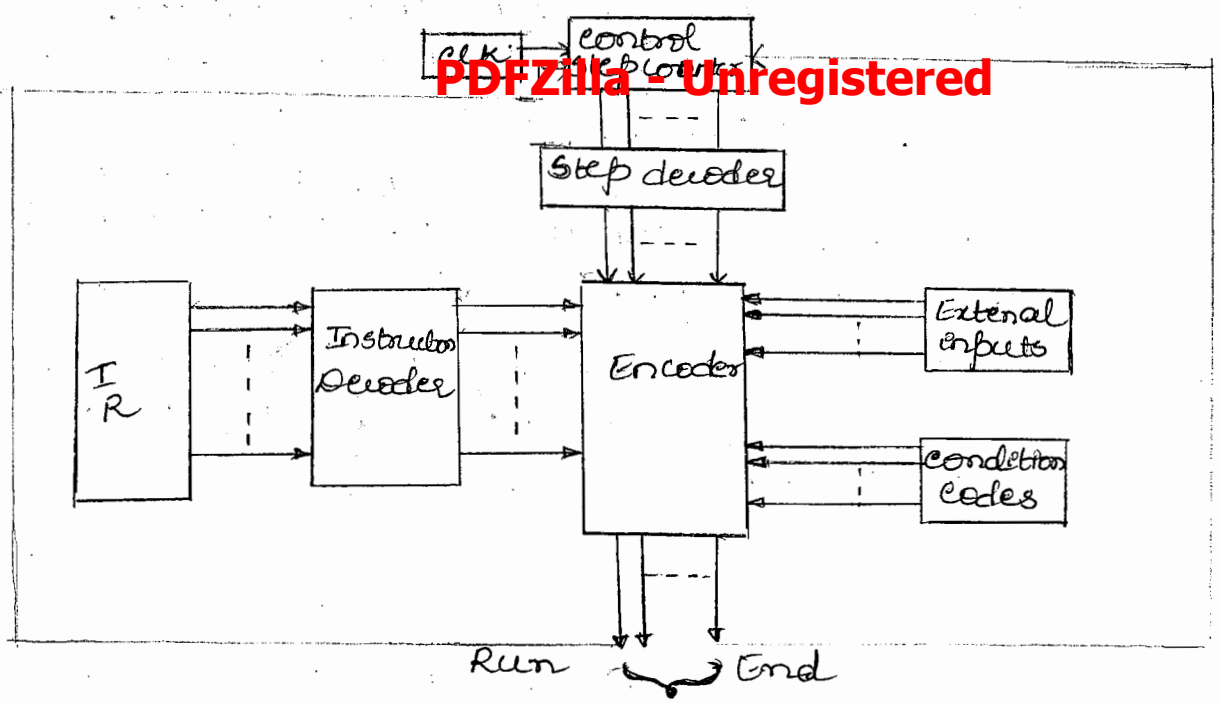
Control sequence consider the control sequence for the instruction Add (R3), R1 and Branch instruction.

Every step in the sequence is completed in one clock period.

The required control signals are determined by

1) contents of control step counter
2) contents of instruction register
3) contents of conditional code flags
4) External i/p signals, such as MFC and interrupt request.





The decoder/encoder block is a combinational circuits that generates the required control outputs depending on the state of all its inputs. Counter is to keep track of steps.

Step decoder provides a seperate signal line for each step, or time slot on the control sequence. i,e at time T_1, step1 gets generated, T_2 step2 gets generated and so on.

IR will have different instruction but one at a time. (Add, XOR, Move ...) Instruction decoder will generate seperate signal for each instruction. So if we assume in instruction set architecture there are 256 instructions then 256 different signals can be generated. (only one will be active at a time)

Encoder circuits combenes all the inputs and generate a individual control signal ;e Yin, Pcout ... .

For the control sequence Add (R3), R and Branch instruction when decoder generates Zin signal.

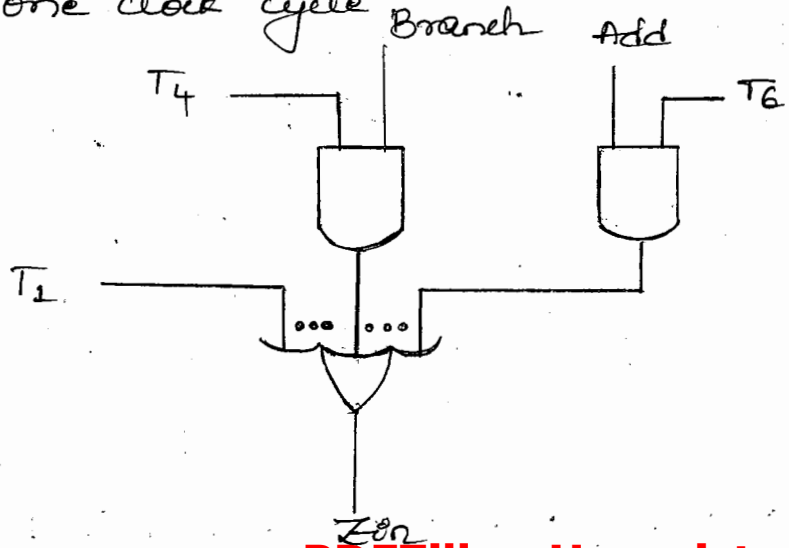
$$Zin = T_1 + T_6 \cdot Add + T_4 \cdot BR + \cdots$$

Zin is generated in time slot T_1 for all the instruction since fetch phase is same for all instructions. During T_6, Zin is generated for Add and during T_4, Zin is generated for Branch instruction.

$$End = T_7 \cdot Add + T_5 \cdot BR + \cdots$$

End signal resets the counter so that next instruction will be fetched.

Run signal is set to 1 causes the counter to be incremented at every clock cycle. When Run is 0 counter stops counting which is required when MFC signal is issued, which causes the processor to wait for the reply from memory for more than one clock cycle



Microprogrammed control

In hardwired, control signals are generated by encoder decoder circuit.

In microprogramed control, signals are generated and stored in memory. As and when required they are read from memory.

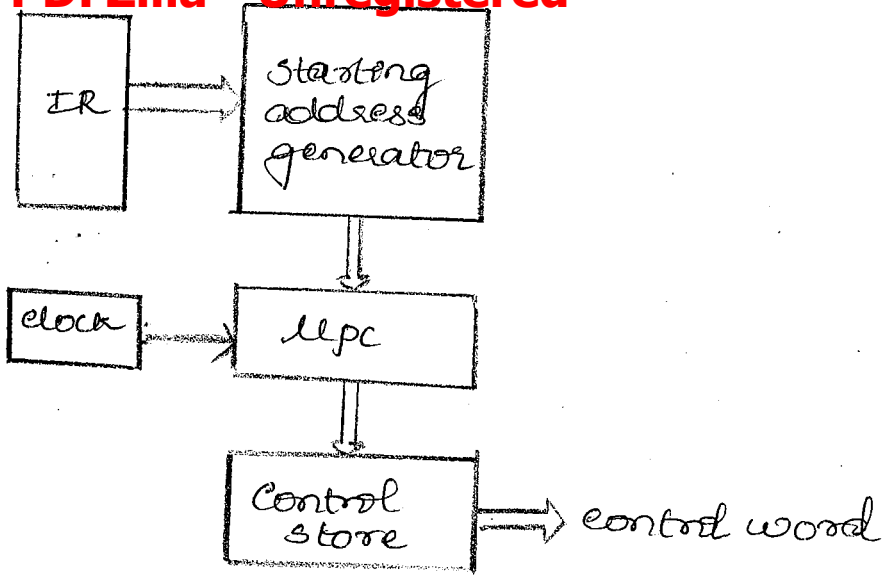
Control word is a word whose individual bits represents various control signals.

Consider an example of control sequence written for Add (R3), R1.

Seven steps represents seven control word. A sequence of control words which is generated for a individual instruction is known as microroutine for that instruction. Individual control word in a microroutine represents microinstruction.

An example of microinstructions for the sequence generated for Add (R3), R1

Micro-instructions	..	PCin	PCout	MARin	Read	MDRout	IRin	Yin	Select	Add	Zin	Zout	R1out	R3out	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	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	1	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	



Basic organization of microprogrammed control unit

The microroutines for all instructions are stored in a special memory called as control store. The control unit can generate the control signals by sequentially reading the control word of the microroutine from the control store.

To read the control words sequentially from control store, a micro program counter (μpc) is used.
When a new instruction is loaded into IR, starting address generator, generates the starting address of the microroutine in control store.
The starting address is stored in μpc. On application of clock i.e every clock cycle the μpc gets incremented and control words are read sequentially from control store.

Microroutine for instruction Branch <0

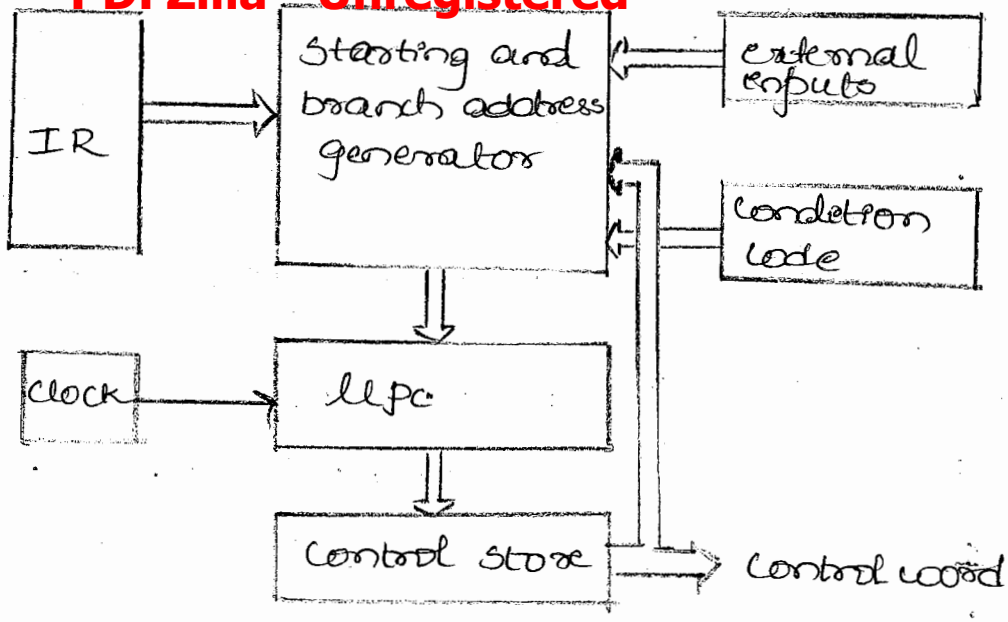
Address	Microinstruction
0	PC_{out}, MARin, Read, Select4, Add, Zin
1	Zout, PCin, Yin, WMFC
2	MDRout, IRin
3	Branch to starting address of appropriate microroutine.
. . .	. . . . .
25	If N=0, then branch to microinstruction 0
26	Offset field of IRout, Select Y, Add, Zin
27	Zout, PCin, End.

when the instruction Branch<0 is loaded into IR, in the above example we assumed that the appropriate routine for Branch<0 is in the address 25 of control store.

Since it is conditional branch, if N=0 (i,e result of arithmetic operation is -Ve then N=1 else N=0) then branch to microroutine 0. (Fetch the next instruction).

If not (N=1) then Branch target address should be added loaded into PC. So to calculate the branch target address by adding content of PC to offset (target address in the address 26,27). Store back in PC. (This happens

organization of the control unit to allow conditional branching in microprogram.



To support microprogram branching the control unit is modified. Starting address generator block becomes Starting and branch address generator.

This block loads a new address into μpc when the microinstruction instructs the starting address & branch address generator to do so.

To allow conditional branch, inputs to the block consist of external inputs and conditional codes and also Instruction register.

Microinstructions

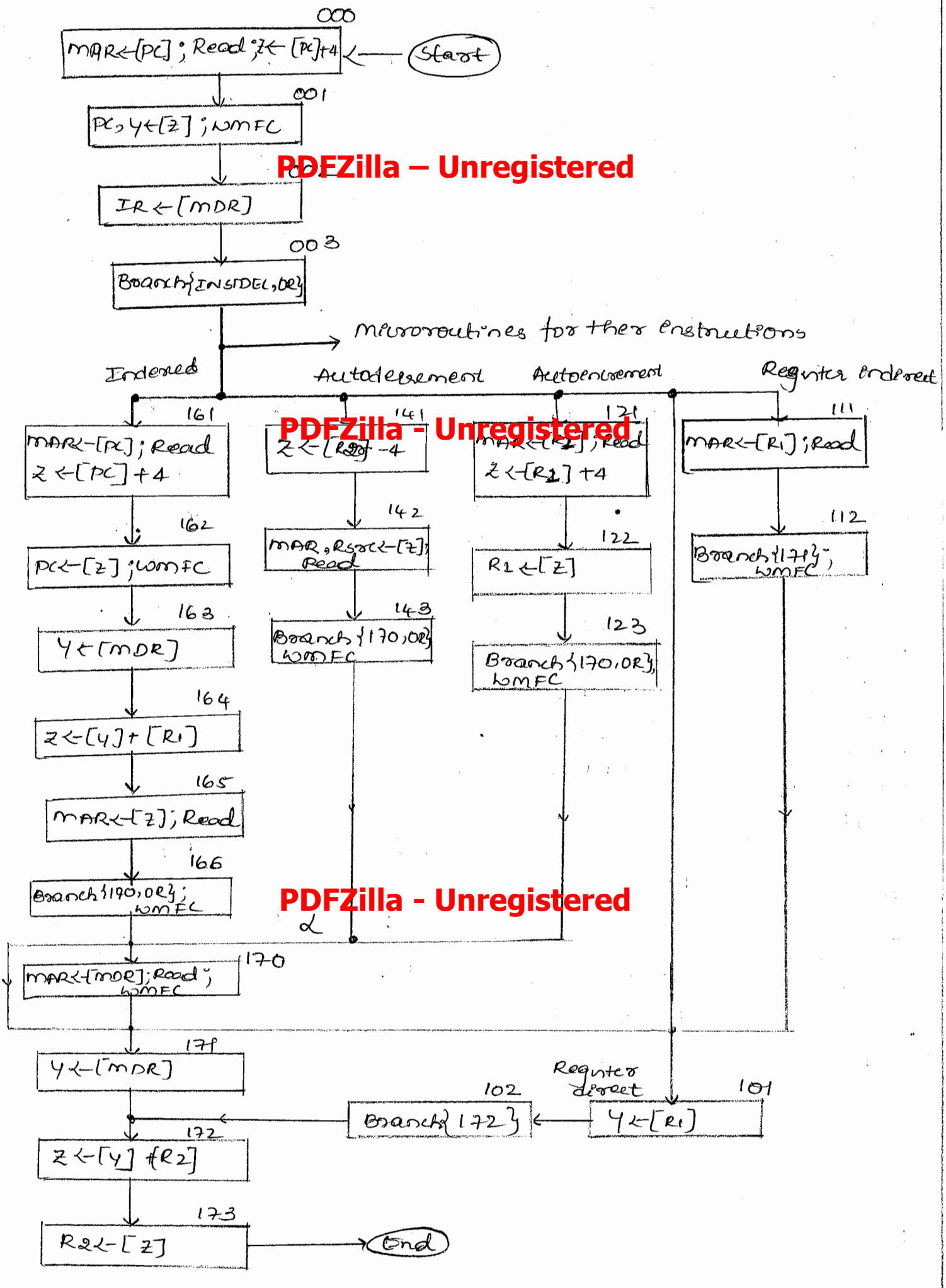
A simple way to structure microinstruction is to assign one bit position to each control signal.
Assigning individual bits to each control signal signal results in long microinstructions. Mar.
To avoid this the signals are grouped into different fields. Signals are grouped in such a way that the signals are mutually exclusive i,e only one signal in a field can be active at a time.

Microinstruction

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)
0000: No transfer	000: No transfer	000: No transfer
0001: PC out	001: PC in	001: MAR in
0010: MDR out	010: IR in	010: MDR in
0011: Z out	011: Z in	011: TEMP in
0100: R0 out	100: R0 in	100: Y in
0101: R1 out	101: R1 in	
0110: R2 out	110: R2 in	
0111: R3 out	111: R3 in	
1010: TEMP out		
1011: Offset out		

F4 (4 bits)	F5 (2 bits)	F6 (1 bit)	F7 (1 bit)
0000: Add	00: No action	0: select y	0: No action
0001: Sub	01: Read	1: select 4	1: WMFC
.	10: write		
.			
1111: XOR			

F8 (1 bit)
0: Continue
1: End



Flow chart for microprogram Add R1, R2 instruction

The flow chart is for add instruction, which represents the microroutines for different addressing mode of add instruction.

we will consider the microroutine for add instruction in autoincrement mode

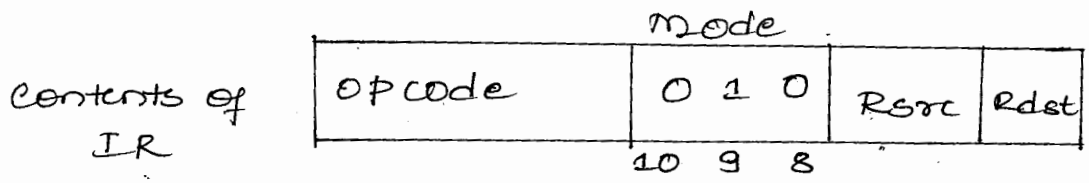
i,e Add (R1)+, R2

The address 000, 001, 002 are for fetch, which is same for all the instruction.

The address 003 is a branch instruction. which decides the jump to the appropriate routine.

Instruction Decoder determines the address of the appropriate microroutine in control store. For the above example, since it is a Add instruction a jump is taken to the address 101.

Now there is a need to determine the microroutine for the autoincrement addressing mode. So for that 10th bit and 9th bit of IR register is added to 5th bit of 4th bit of uPC.



In IR register three bits i,e 10, 9 & 8 are used to specify modes.

10th	9th bit	
1	1	→ this indicates indexed addressing mode
1	0	→ autodecrement addressing mode
0	1	→ autoincrement addressing mode
0	0	→ register addressing mode.

If 8th bit is 1 then it is indirect form of the corresponding addressing mode.

Ex:- 111, 101, 011, 001 → indirect indexed, autodecrement autoincrement and register mode.

Ex:- 110, 100, 010, 000 → direct indexed, autodecrement autoincrement and register mode.

· $\mu pc = 101$ (octal)

8 7 6 5 4 3 2 1 0

001 000 001 ← μpc (101)

10 ← (10th & 9th bit of IR)

BitOR

001 100 001 ← μpc

Set the 3rd bit of μpc by 8th bit of IR i.e

$\mu pc_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$

$0 \cdot 1 \cdot 0 = 0$

$\mu pc =$ 001 100 001

1 4 1

Take a branch to 141 address which is the routine for autoincrement addressing mode.

→ At the address 143, once again there is a branch instruction to decide whether to execute the instruction at address 170 are to skip.

→ Micro instruction at address 170 gets executed only if it is indirect form of addressing mode. In our example it is direct form of autoincrement mode hence 170 has to be bypassed.

μpc contents = 170 i.e 001 111 000 ← μpc (170)

$\mu PC_0 = [\overline{IR_8}]$ = bit OR 1

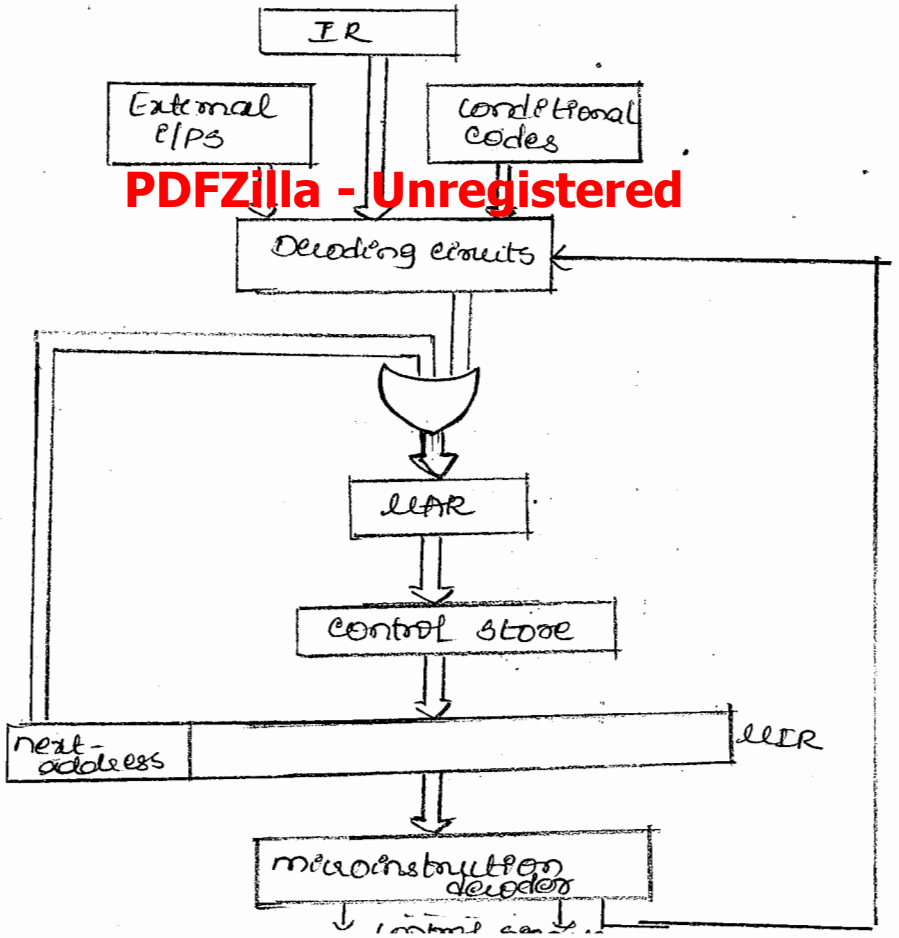
$\overline{IR_8} = 1$

001 111 001 (171)

(Octal) Address	Microinstruction
000	PCout, MARin, Read, Select4, Add, Zin
001	Zout, PCin, Yin, WMFC
002	MDRout, IRin
003	uBranch { uPC ← 101 (from Instruction decoder); $uPC_{5,4} \leftarrow [IR_{10,9}]$; $uPC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$
121	R1out, MARin, Read, Select4, Add, Zin
122	Zout, R1in
123	uBranch { uPC ← 170; $uPC_0 \leftarrow [\overline{IR_8}]$ }; WMFC
170	MDRout, MARin, Read, WMFC
171	MDRout, Yin
172	R2out, SelectY, Add, Zin
173	Zout R2in, End

For our example 170 will not execute.

Microinstructions with Next-Address Field



In the previous flowchart there are several branch microinstructions.
These microinstructions are needed just to find the address of next microinstructions. So to avoid this extra branch routine an alternate approach is used where the address field is included as a part of every microinstruction. This address specifies the from where the next microinstruction has to be fetched. (so every instruction becomes a branch instruction also). Only problem in this approach is extra bits are needed to specify address.

Since every microinstruction holds the address of next instruction μpc is not needed to hold the starting address (sequential) of microinstructions.

μpc is replaced by micro address register which is loaded from next address field of every microinstruct

Next-address is fed into μAR by OR-gate, so that address can be modified based on external I/P, conditional codes.

Decoding circuits will generate the starting address of microroutine based on opcode in IR.

For this approach several extra signals are needed.
Ex:- OR mode to set of bit-ORing is used.
OR indsrc → is set to 1 if indirect addressing mode is used.

Microinstruction

F0 (8 bits)	F1 (3 bits)	F2 (3 bits)
Address of next instruction	000 : NO transfer 001 : PCout 010 : MDRout 011 : Zout 100 : Rsrcout 101 : Rdstout 110 : Tempout	000 : NO transfer 001 : PCin 010 : IRin 011 : Zin 100 : Rsrcin 101 : Rdstin

F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
000 : No transfer	0000 : Add	00 : No action
001 : MARin	0001 : Sub	01 : Read
010 : MDRin	:	10 : write
011 : Tempin	:	
100 : Yin	:	

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0 : Select Y	0 : No action	0 : Next Addr
1 : Select 4	1 : WMFC	1 : Instruc

F9 1bit	F10 1bit
0 : No action	0 : No [illegible]
1 : ORmode	1 : ORindirec.

Implementation of microroutine for sequence Add (R1)+, R2 using encoded signals

Octal address	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
000	00000001	001	011	001	0000	01	1	0	0	0	0
001	00000010	011	001	100	0000	00	0	1	0	0	0
002	00000011	010	010	000	0000	00	0	0	0	0	0
003	00000000	000	000	000	0000	00	0	0	1	1	0
121	01010010	100	011	000	0000	01	1	0	0	0	0
122	01111000	011	100	000	0000	00	0	1	0	0	1
170	01111001	010	000	001	0000	01	0	1	0	0	0
171	01111010	010	000	100	0000	00	0	0	0	0	0
172	01111011	101	011	000	0000	00	0	0	0	0	0
173	00000000	011	100	000	0000	00	0	0	0	0	0

QUESTION BANK - III (Module – 4 & 5)

1. Convert the following pairs of decimal numbers to 5 bit, signed, 2's complement binary numbers and add them. State whether or not overflow occurs in each case.

i) 7 and 13
ii) –5 and 7
iii) –10 and –13
iv) –14 and 11
v) -3 and -8

2. Draw a figure to illustrate and explain a 16-bit Carry-Look Ahead Adder using 4-bit adder blocks.
   Show that the carry and sum are generated in 5 and 8 gate delay respectively.

3. Using a block diagram which shows the register configuration, perform sequential circuit binary multiplication of Multiplicand=1010 and Multiplier = 1101.

4. Explain Booth's algorithm. Multiply 01110 (+14) and 11011 (-5) using Booth's multiplication, and explain.

5. Explain the design of 4-bit Ripple Carry Adder with a neat diagram.

6. Explain the design of a 4-bit Carry-Look Ahead Adder , with a neat diagram.

7. Show the multiplication of (+13) and (-6) using bit-pair recoding technique.

8. Explain Binary addition/subtraction logic network to perform the subtraction operation X-Y on 2's complement numbers X and Y.

9. Differentiate between restoring and non-restoring division.

10. Illustrate the steps for non restoring division algorithm on the following data:
    Dividend=1011, divisor=0101.

11. Explain Two methods of Fast Multiplication.

12. Explain Carry-Save Addition of Summands.

13. Write IEEE standard floating–point formats for 32-bit representation and explain

14. Illustrate with an example the algorithm for Restoring binary division with block diagram.

15. Illustrate with an example the algorithm for Non-Restoring binary division with block diagram.

16. Explain 4 bit carry–look ahead adder and use it to build 16 bit carry–look ahead adder.

17. Perform 56-78 using 1's complement and 2's complement methods.

18. Using Booth algorithm multiply (-13) and (+ 107).

19. Draw a circuit diagram for binary division and explain its operation.

20. Explain how Booths algorithm is suitable for signed numbered multiplication in comparison of conventional shift and add method.

21. Write a short note on look ahead carry generator.

22. How do you design FAST ADDERS? Explain 4 bit carry look ahead adder?

23. Explain the sequential binary multiplier with the use of a block diagram.

24. Explain the computational details of multiplying two 4 bit numbers 1 0 1 1 and 0 1 0 1 using Booths algorithm. Verify the result obtained.

25. With a neat diagram explain the floating-point addition-subtraction unit.

26. Multiply 10011 and 01001 using Booth's algorithm.

27. Let Multiplicand A = 110101 and Multiplier B= 011011. Multiply the given signed 2's complement numbers using Booth's algorithm. Verify the result using bit-pairing of multiplier.

28. Draw a neat sketch of single bus organization of the data path inside a processor , explain the three steps to be performed by the processor to execute an instruction.

29. Write and explain the control sequences for execution of following instruction with respect to single bus organization: Add R2, (R4) .

30. Write and explain the control sequences for execution of following instruction with respect to single bus organization: Add (R3), R1 .

31. List the actions needed to execute the instruction Add R1, (R3). Write the sequence of control steps to perform the actions for a single bus structure. Explain steps.

32. With a neat Block diagram , explain three bus organization and write control sequence for the instruction: Add R1, R2, R3 .

33. Compare hardwired control unit with micro-programmed control unit.

34. Write the control sequences for the instruction Move (R1), R2.

35. With a neat block diagram , explain hardwired control unit ,which shows separation of the decoding and encoding function.

36. Explain with a neat block diagram , the basic organization of a micro-programmed control unit.

37. Write and explain the control sequences for execution of an unconditional branch instruction.

38. Write and explain the control sequences for the execution of following instruction: Add (R3), R1

39. What are the modifications required in the basic organization of a micro programmed control unit to support conditional branching in the micro program.

40. Explain micro instruction sequencing with next address field.

41. Give the Micro instruction for Branch < 0.

42. Draw a block diagram of a complete processor and identify the units