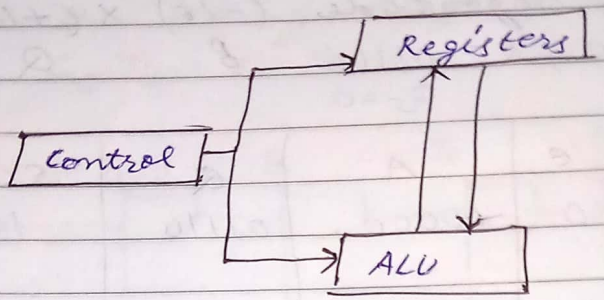
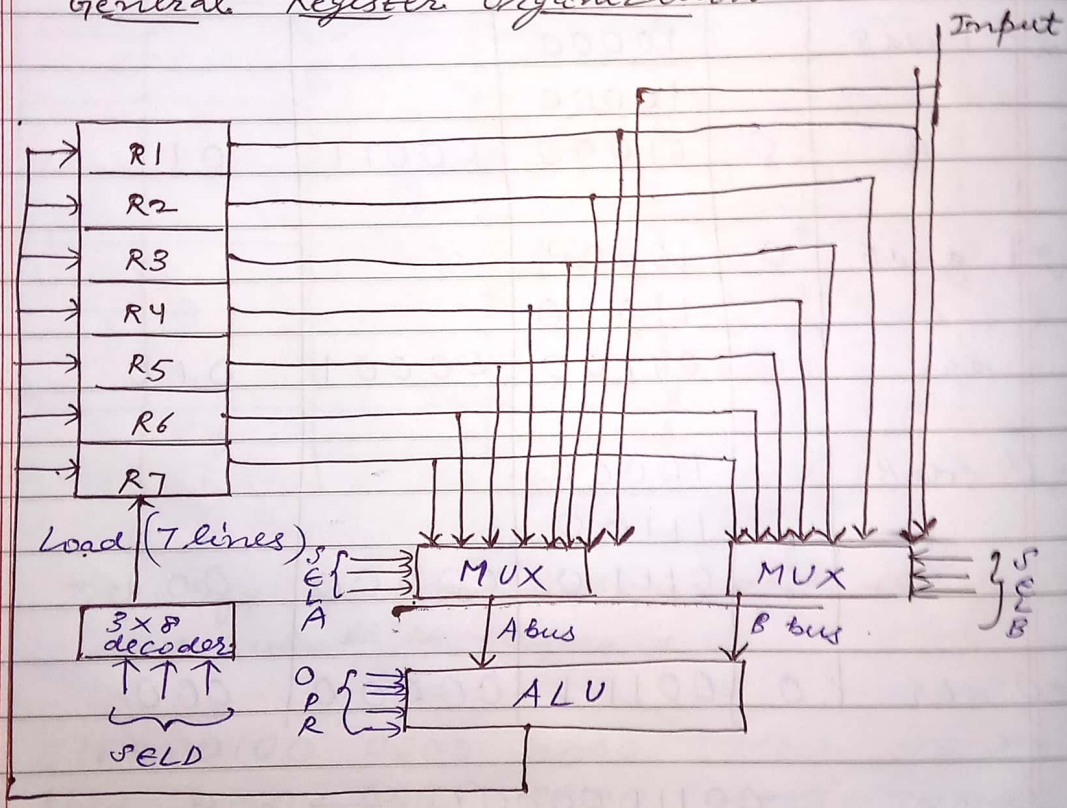
Central processing Unit:



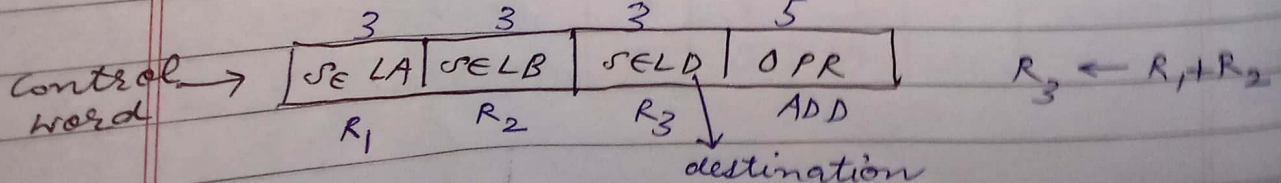
General Register Organization:



SEL → select lines

If data comes from input then value of SEL → 000

Total select lines = 14



$R_3 \leftarrow R_1 + R_2$

Ques

OPR → operation bits

OPR select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A+B	ADD
00101	Subtract A-B	SUB
00110	Decrement A	DECA
01000	AND A&B	AND
01010	OR A&B	OR
01100	XOR A&B	XOR
01110	Complement A	COMA
10000	shift right A	SHRA
11000	shift left A	SHLA

Ques

Microoperation	SELA	SELB	SELD	OPR	Control Word
$R_1 \leftarrow R_2 - R_3$	R_2	R_3	R_1	SUB	01001100101
$R_6 \leftarrow R_6 + 1$	R_6	—	R_6	INCA	11000011000001
Output ← Input	Input	—	output	TSFA	0000000 0000000
$R_5 \leftarrow 0$, $R_5 \leftarrow R_5 \oplus R_5$	R_5	R_5	R_5	XOR	101 10110 01100

→ The part of computer that performs the bulk of data-processing operations is called CPU.
Register set stores the intermediate data used during the execution of instructions.
ALU performs required microoperations.
Control unit supervises the transfer of

information among registers and instructs ALU as to which operation to perform.

General Register organization:

Used for performing all arithmetic, logic, shift microoperations.

A bus organization for 7 CPU registers.

Output of each register is connected to 2 multiplexers to form buses A & B. selection lines in each multiplexer select one register or input data for particular bus. Buses A & B forms input to common ALU. The operation selected in ALU determines microoperation to be performed. The register that receives the information from output bus is selected by a decoder. Decoder activates one register for providing a transfer between data in output bus and inputs of selected destination register.

$$R_1 \leftarrow R_2 + R_3$$

(1) MUX A selector (SELA) → place content of R2 into bus A

(2) MUX B selector (SELB) → place content of R_3 into bus B.

(3) ALU operation selector (OPR) → provide arithmetic addition A+B.

(4) Decoder destination register (SELD) → transfer content of output bus into R1.

Ques A bus organized CPU has 16 registers with 32 bits in each and alu and a destination decoder

(1) how many multiplexers are there in A bus and what is size of each multiplexer

(2) how many selection inputs are needed for MUX A and MUX B.

(3) how many inputs and outputs are there in the decoder.

(4) how many inputs and outputs are there in the ALU for data include in input and output carries.

(5) Indu formulate a control word for the system assuming that ALU has 35 operations.

→ (2) as there are 16 registers means 4 bits are required i.e. 4 select lines

(3) 4 x 16 decoder

(4) 32 bit input from MUX A and 32 bit input from MUX B and output is of 32 bit

Input = 32+32+1 = 65

output = 32+1 = 33

(5)

SELA	SELB	SELD	OPR
4	4	4	6

35 operations ↓ 2^6

4+4+4+6 = 18 bits

(1) 32 multiplexer (as no. of bits = 32)

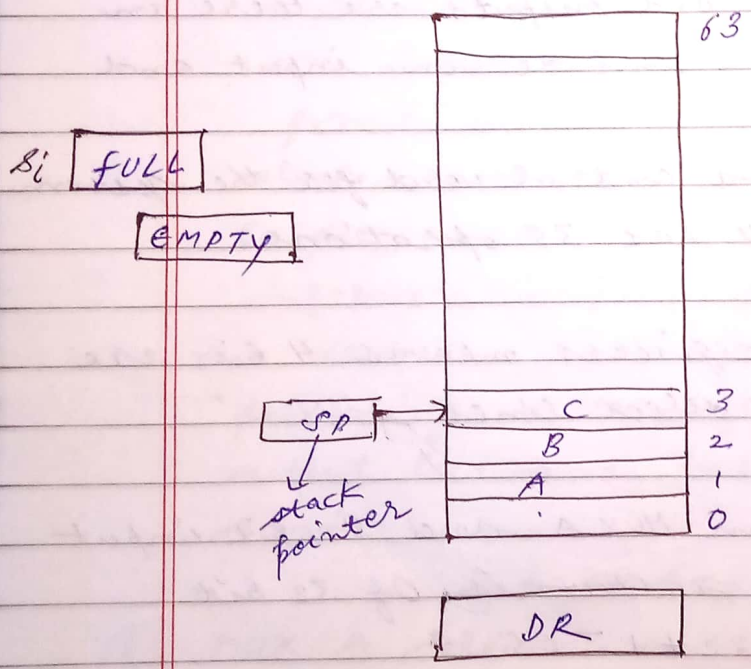
16X1 → size of multiplexer

Ques Determine the microoperations that will be executed in the processor when the following 14 bit control words are applied.

		SELA	SELB	SELD	OPR	micro-operation
a)	00101001100101	R1	R2	R3	SUB	$R_3 \leftarrow R_1 - R_2$, $R_2 \leftarrow 0$
b)	01001001001100	R2	R2	R2	XOR	$R_2 \leftarrow R_2 \oplus R_2$
c)	11110001110000	R7	R4	R3	SHRA	$R_3 \leftarrow R_7$ shr

Stack Organization:

Register stack



Push: $SP \leftarrow SP + 1$ Increment stack pointer

$M[SP] \leftarrow DR$ write item on top of stack

if $(SP = 0)$ then $FULL \leftarrow 1$ check if stack is full

$EMPTY \leftarrow 0$ Mark the stack not empty

pop: DR ← M[SP] read item from top of stack

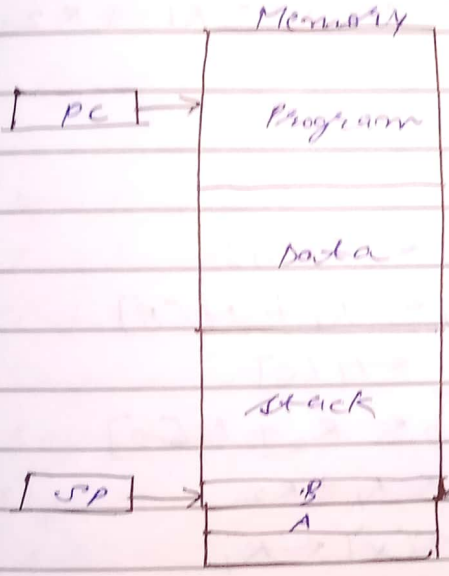
SP ← SP − 1 Decrement stack pointer

If (SP = 0) then EMPTY ← 1 check if stack is empty

FULL ← 0 Mark the stack not full

Last element will be inserted at 0 location

Memory organization:



push

SP ← SP − 1 Decrement stack pointer

M[SP] ← DR

pop

DR ← M[SP]

SP ← SP + 1 Increment stack pointer

stack organization:

Register stack

stack follows LIFO (Last-in, first-out)

The register that holds the address of the stack is called stack pointer because its value always points at the top item in stack.

Operation of insertion is called push

Operation of deletion is called pop.

Types of CPU Organization:

1. Single Accumulator organization $AC \leftarrow AC + M[X]$
2. General Register organization $R_1 \leftarrow R2 + R3$
3. stack Organization push and pop

Three-Address Instructions: $X = (A+B) * (C+D)$ ($A+B$ → $R1$, $C+D$ → R_2)

ADD	R1	A	B	$R_1 \leftarrow M[A] + M[B]$
ADD	R2	C	D	$R_2 \leftarrow M[C] + M[D]$
MUL	X	R1	R2	$M[X] \leftarrow R1 * R2$

Two-Address Instructions:

MOV	R1	A	$R_1 \leftarrow M[A]$
ADD	R1	B	$R_1 \leftarrow R_1 + M[B]$
MOV	R2	C	$R_2 \leftarrow M[C]$
ADD	R2	D	$R_2 \leftarrow R_2 + M[D]$
MUL	R1	R2	$R_1 \leftarrow R_1 * R_2$
MOV	X	R1	$M[X] \leftarrow R_1$

One-Address Instruction:

LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$
STORE	T	$M[T] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	T	~~$AC \leftarrow AC * M[T]$~~ $M[T] \leftarrow AC * AC$ ~~$AC \leftarrow AC * M[T]$~~
STORE	X	$M[X] \leftarrow AC$

Zero-Address Instructions:

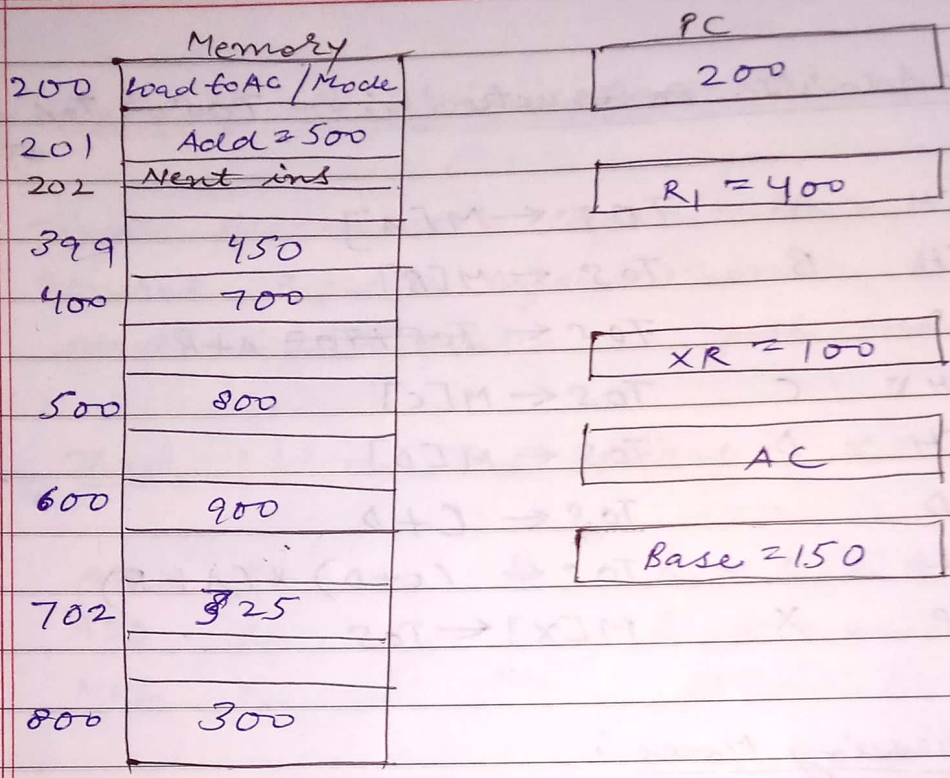
TOS → Top of stack

PUSH	A	TOS ← M[A]
PUSH	B	TOS ← M[B]
ADD		TOS ← ~~TOS+TOS~~ A+B
PUSH	C	TOS ← M[C]
PUSH	D	TOS ← M[D]
ADD		TOS ← C+D
MUL		TOS ← (C+D)*(A+B)
POP	X	M[X] ← TOS

Addressing Modes:

1. Direct
2. Indirect
3. Immediate
4. Implied
5. Register
6. Register Indirect
7. Auto-Increment
8. Auto-Decrement
9. Relative
10. Indexed Addressing
11. Base Register Addressing

← EA = AP + Content of Register
(address part)
500 + 202
= 702



	Effective address	Content of AC
1.	500	800
2.	800	300
3.	201	500
4.	—	—
5.	—	400
6.	400	700
7.	400	700
8.	399	450
9.	702	325
10.	600	900
11.	650	—

Implied Mode → operands are specified implicitly. eg. complement accumulator zero address instructions are implied mode

Immediate Mode → operand is specified in instruction itself. Operand field contains actual operands to be used in conjunction with operation specified. Useful for initializing registers to a constant value.

Register Mode → Operands in register reside within CPU. A k-bit field specify any one of 2^k registers.

Register Indirect Mode → selected register contains address of operand. Before using a indirect mode, programmer must ensure that the memory address of operand is placed in processor register with previous instruction. Advantage is that the address field of instruction uses fewer bits to select a register.

Autoincrement or Autodecrement Mode → Used to access memory. In this, in case of auto-increment first the effective address is fetch and then it is incremented but in case of autodecrement, effective address is decremented first.

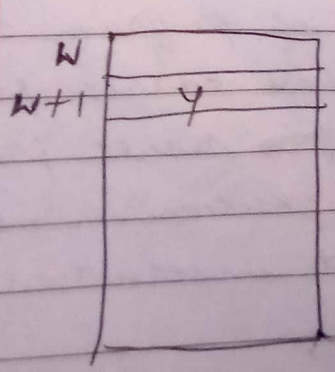
Relative address Mode → Content of program counter is added to address part. When address part is added to content of PC, the result produces an effective address whose position in memory is relative to address of next instruction.

Indexed Addressing Mode → content of an index register is added to address part of instruction to obtain effective address, Index register contains index value. Index register can be incremented to facilitate access to consecutive operands.

Base Register Addressing Mode → content of base register is added to address part of instruction to obtain effective address, Used to facilitate the relocation of programs in memory.

Ques A 2 word instruction is stored in memory at address designated by symbol W. The address field of instruction stored at W+1 is designated by symbol Y. The operand used during the execution of instruction is stored at an address symbolized by Z and index register contains the value at X. State how Z is calculated from other addresses if the addressing mode of instruction is

a) Direct
b) Indirect
c) Relative
d) Indexed



XR = X

direct → Z = Y

indirect → Z = M[Y]

relative → Y + W + 2

indexed → Y + X

Ques An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 200. Evaluate effective address if the addressing mode of instruction is

a) Direct
b) Immediate
c) Relative
d) Register Indirect
e) Indent with R1 as inden register

300	
301	400

$R_1 = 200$
direct → 400
register indirect → 200
relative → 400+302 = 702
immediate → 301
inden → 600
indirect → M[400]

Ques find effective address & content

Memory

300	Load to AC1
301	Add =850
700	950
850	1050
900	
1050	125
1102	880

XR = 100
R1 = 700 (processor register)

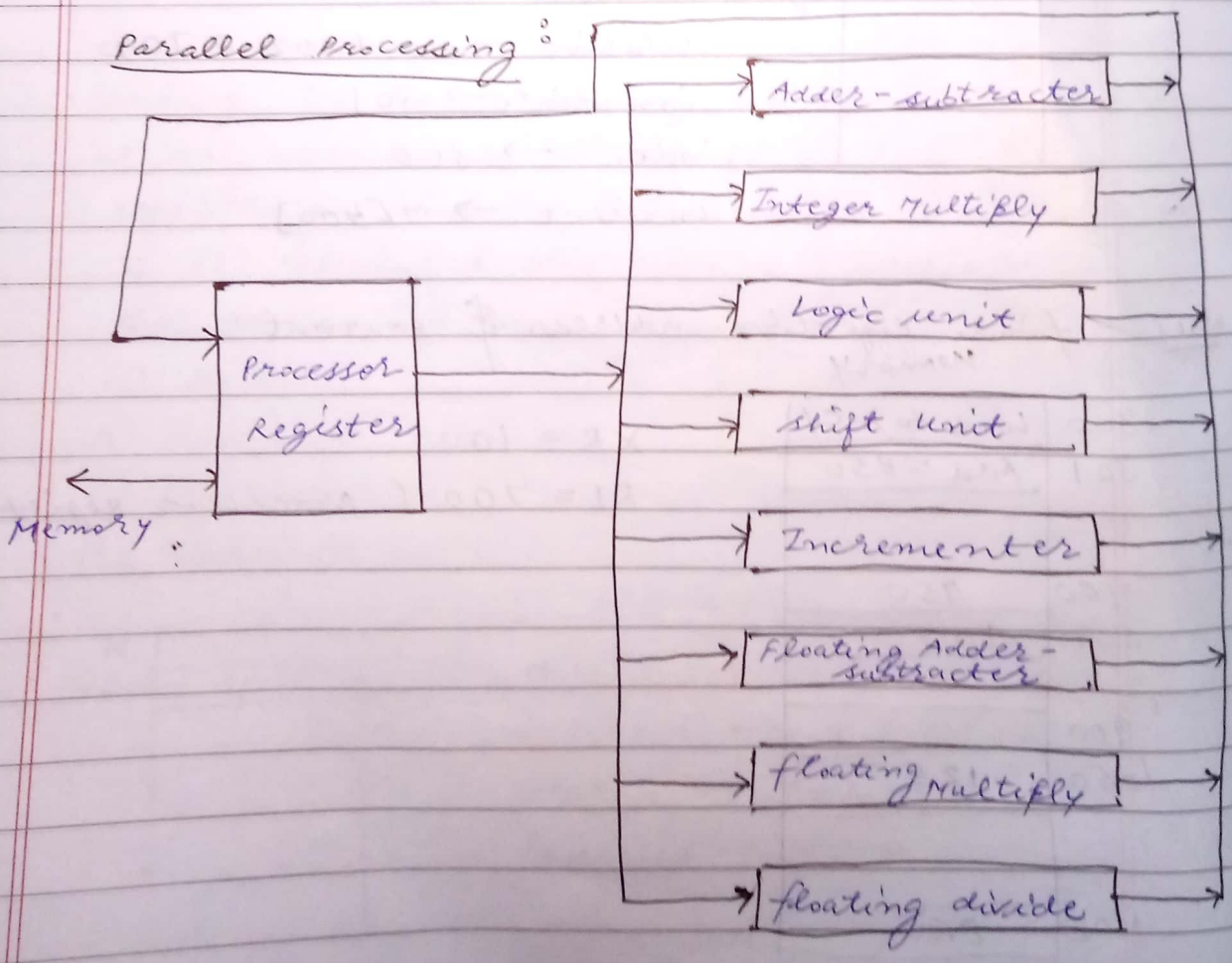
Program Control (Theory)

	EA	EA	Content of AC
1.	Direct	850	1050
2.	Indirect	1050	125
3.	Immediate	301	850
4.	Register (700)	.	950
5.	Register Indirect	700	950
6.	Auto-Increment	700	950 M[699]
7.	Auto-decrement	699	850 M[950]
8.	Relative (850+302)	1152	M[1550]
9.	Indexed	950	
10.	Base (850+700)	1550	

Relative (Add Part + PC)
Indexed (AP + IR)
Base (AP + Base)

chapter-9

Parallel Processing :



Pipelining:

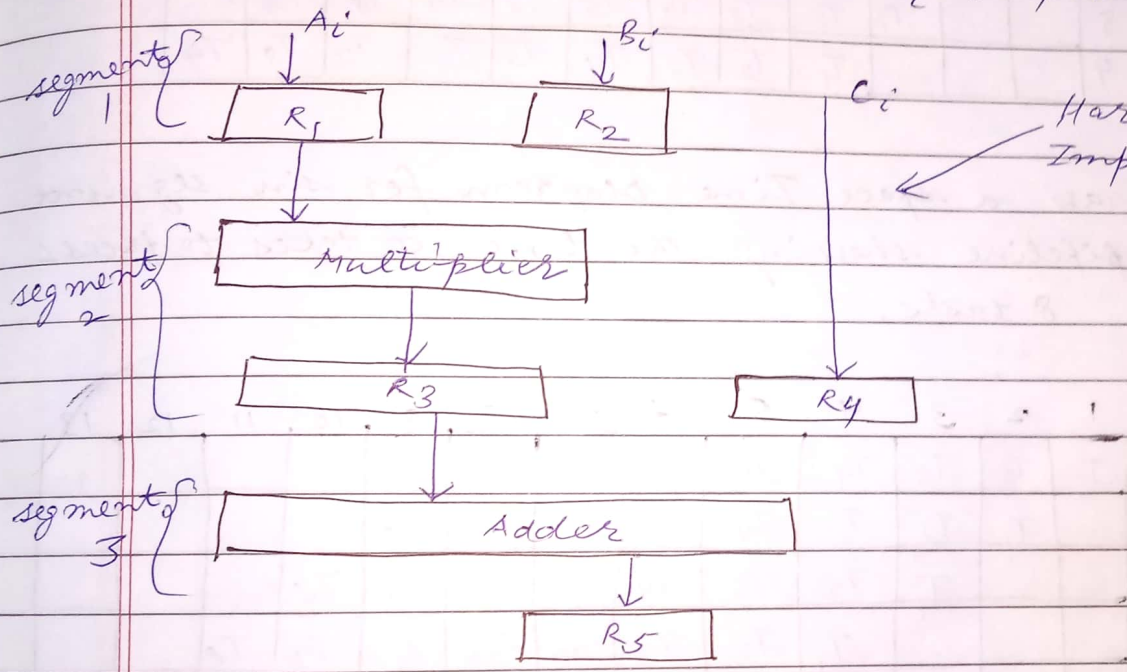
$A_i * B_i + C_i$ for $i = 1, 2, 3 \ldots 7$

segments:

$R_1 \leftarrow A_i$, $R_2 \leftarrow B_i$ Input A_i and B_i

$R_3 \leftarrow R_1 * R_2$, $R_4 \leftarrow C_i$ Multiply and input C_i

$R_5 \leftarrow R_3 + R_4$ Add C_i to product



clock pulse	segment 1		segment 2		segment 3
	R_1	R_2	R_3	R_4	R_5
1					
2	A_1	B_1	~~B_1~~	—	—
3	A_2	B_2	$A_1 * B_1$	C_1	—
4	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
5	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
6	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
7	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
8	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
9			$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
10					$A_7 * B_7 + C_7$

9 clock pulses as we exclude the registers

$n = 9 \rightarrow$ Tasks
$k = 4 \rightarrow$ segments

Space-Time-Diagram :

clock pulses →	1	2	3	4	5	6	7	8	9	10	11	12
segment 1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9			
segment 2		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9		
segment 3			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	
segment 4				T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9

← 4 segment pipeline

Ques Draw a space Time diagram for six segment pipeline showing the time it takes to process 8 tasks.

	1	2	3	4	5	6	7	8	9	10	11	12	13
segment 1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8					
segment 2		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8				
segment 3			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8			
segment 4				T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8		
segment 5					T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
segment 6						T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8

$6+8-1$

ques →

k-segment pipeline : n tasks

clock pulse = t_p

time for 1st task = kt_p

$(n-1)$ task = $(n-1)\,t_p$

Total time = $(k+n-1)\,t_p$

or

$(k+n-1)$ clock pulses

1 task time = t_n (without pipeline)

Total time = nt_n

$$\text{speedup} = \frac{\text{without pipeline}}{\text{pipeline}} = \frac{nt_n}{(k+(n-1))\,t_p}$$

If in speedup, no. of tasks (n) becomes more than it approaches to n and treats k as negligible

In that case $= \frac{nt_n}{nt_p} = \frac{t_n}{t_p}$

$$S_{max} = \frac{t_n}{t_p}$$

$t_n = kt_p$

$= \frac{kt_p}{t_p} = k$ → S_{max} can't be maximum than k greater

Ques The non-pipelining system takes 50ns to process a task. The same task can be processed in a 6 segment pipeline with a clock cycle of 10ns. Determine speed up ratio of pipeline of 100 tasks. What is the maximum speedup?

→ speedup = ? $\qquad$ speedup $= \frac{nt_n}{(k+(n-1))t_p}$

$t_n = 50$, segments $= k = 6$, $n = 100$

$t_p = 10$, Total time $= (k+n-1)$

$= (6+100-1)$

$= (106-1) = 105\,t_p$

$= 105 \times 10$

$= 1050$ ns

Without pipeline $= 50 \times 100 = 5000$

$$\text{speedup} = \frac{5000}{1050} = 4.76$$

$$S_{max} = 4.76\ \frac{50}{10} = 5$$

Ques Determine the number of clock cycles that it take to process 200 tasks in a 6 segment pipeline.

→ $n = 200, k = 6$

Total clock pulses $= k + n - 1$

$= 6 + 200 - 1$

$= 205$

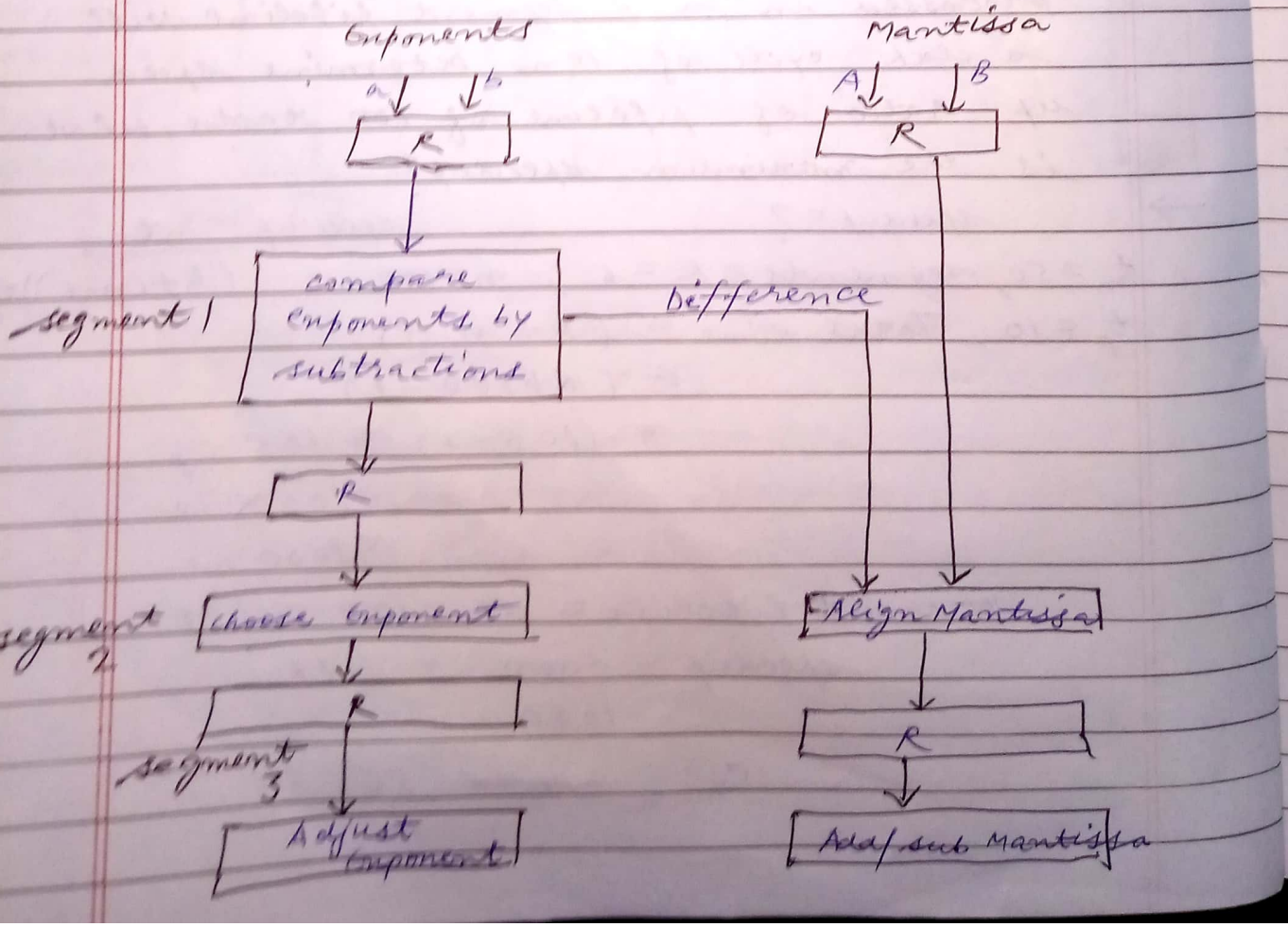
→ Ques 9.5, 9.7 from book

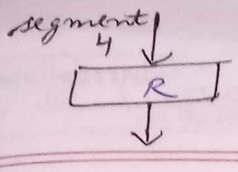
exponent

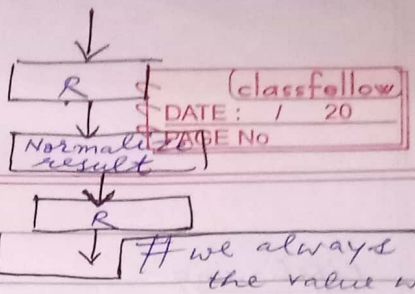
$x = \underbrace{0.9504}_{\text{mantissa}} \times 10^{3}$

Arithmetic pipeline:

1. Compare the exponents
2. Align the Mantissas
3. Add or subtract the mantissa
4. Normalize the result







// we always take the value with greater exponent

$X = 0.9504 \times 10^3$

$Y = 0.8200 \times 10^2$

$y = 0.0820 \times 10^3$

$Z = 1.032 \times 10^3$

$\boxed{Z = 0.1032 \times 10^4}$

parallel processing:

concurrent data processing, amount of hardware increases, cost increases

Parallel processing is established by distributing the data among the multiple functional units

Diagram shows one possible way of separating the execution unit into eight functional units operating in parallel. Operands in registers are applied to one of the units depending on the operation specified by instruction associated with operands.

(1) Adder and integer multiplier perform the arithmetic operations with integer numbers. floating point operations are separated into 3 circuit operating in parallel. The logic, shift, and increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented.

→ Parallel processing may occur in instruction stream, in data stream, or in both

Pipelining:

It is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.

Each segment performs partial processing

Result obtained from each segment is transferred to next segment. Overlapping of computation is made possible by associating a register with each segment in pipeline. Registers provide isolation b/w each segment so that each can operate on distinct data simultaneously.

→ Input register holds the data and combinational circuit performs the suboperation in particular segment. Output of one segment is applied to input register of next segment. Clock is applied to all registers.

Arithmetic Pipeline:

usually found in very high speed computers. Used to implement floating point operations, multiplication of fixed-point numbers and similar computations. Used to reduce execution time of suboperations.

The larger exponent is choosen as the exponent of result. Then mantissa is shifted accordingly.