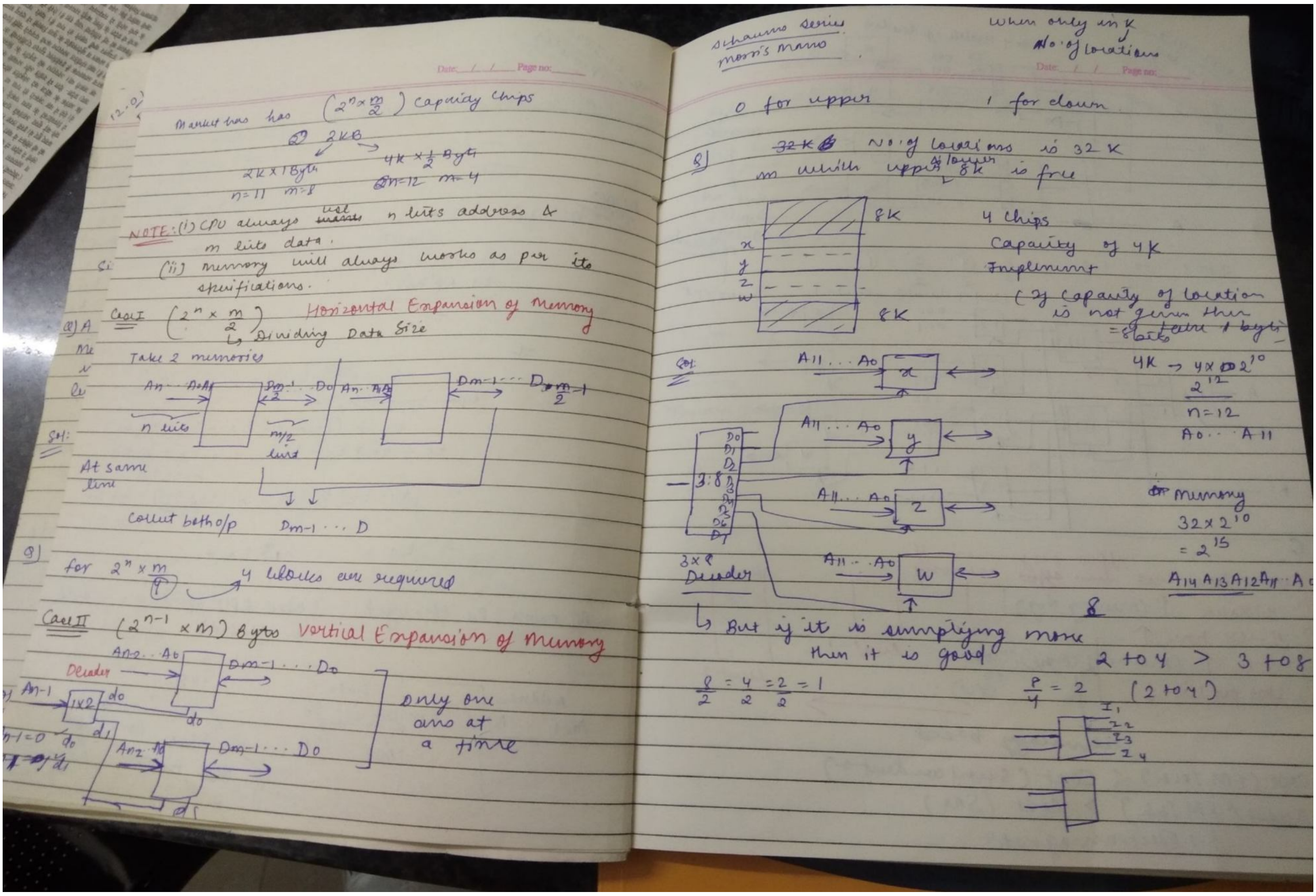
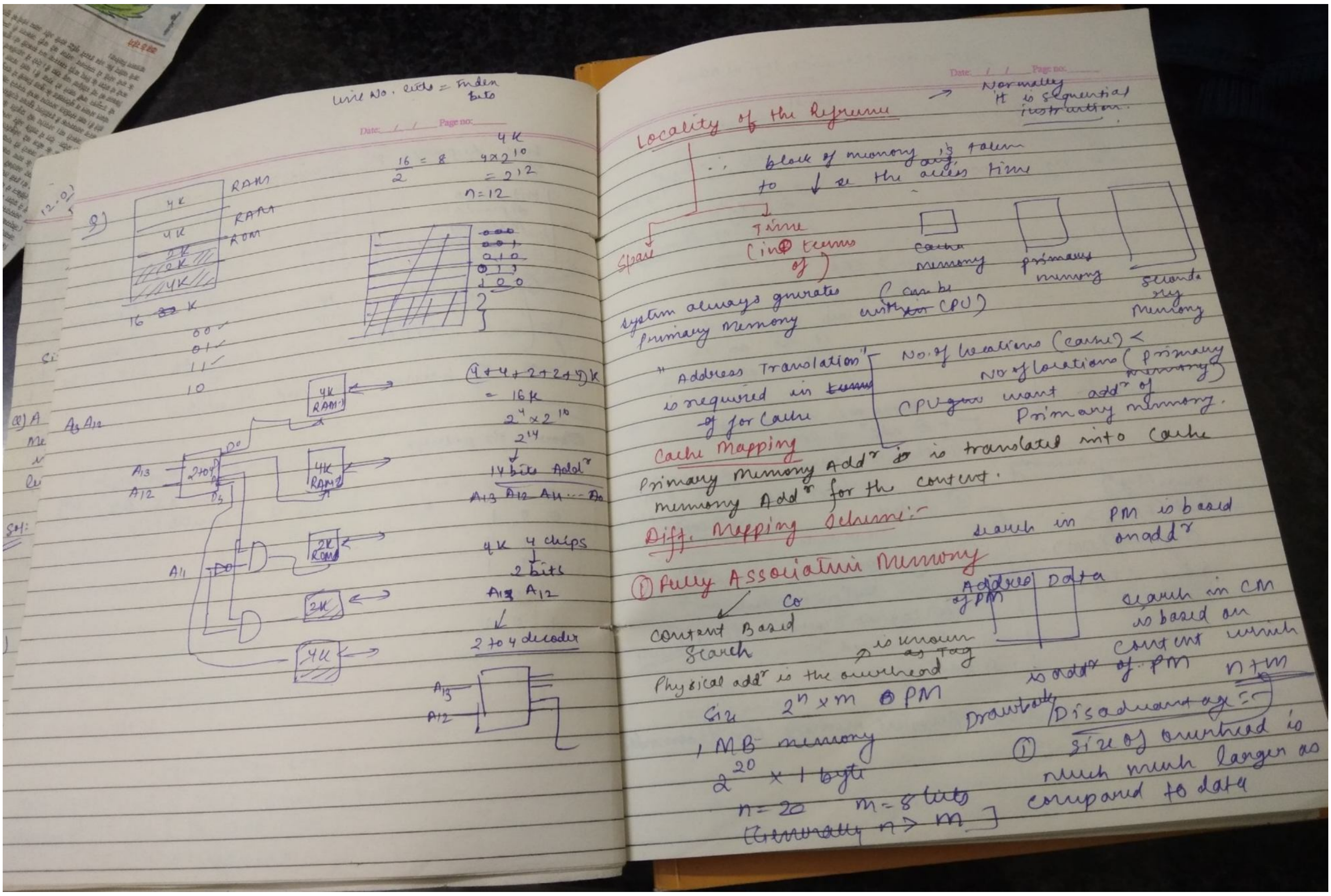




Line No. bits = Index bits

Date: / / Page no:

$\frac{16}{2} = 8$

4K

4×2^{10}

$= 2^{12}$

$n = 12$

Q)

16 ~~32~~ K

00 ✓

01 ✓

11 ✓

10

$A_{13}\, A_{12}$

$(4+4+2+2+4)K$

$= 16K$

$2^4 \times 2^{10}$

2^{14}

↓

14 bits Addr

$A_{13}\, A_{12}\, A_{11} \cdots A_0$

4K 4 chips

↓

2 bits

$A_{13}\, A_{12}$

↓

2 to 4 decoder

Date: / / Page no:

Locality of the Reference

→ Normally it is sequential instruction.

∴ block of memory is taken avg. to ↓se the access time

Space

Time (in terms of)

Cache memory

primary memory

Secondary Memory

System always generates primary memory (can be with CPU)

"Address Translation" is required in ~~terms~~ of for Cache

No. of locations (cache) < No of locations (primary memory)

CPU ~~gen~~ want addr of Primary memory.

Cache Mapping

Primary Memory Addr ~~is~~ is translated into Cache memory Addr for the content.

Diff. Mapping Scheme :-

search in PM is based on addr

(1) Fully Associative Memory

Co

Content Based Search

Address of PM | Data

search in CM is based on content which is addr of PM

Physical addr is the overhead is known as Tag

Size $2^n \times m$ PM

$n+m$

1 MB memory

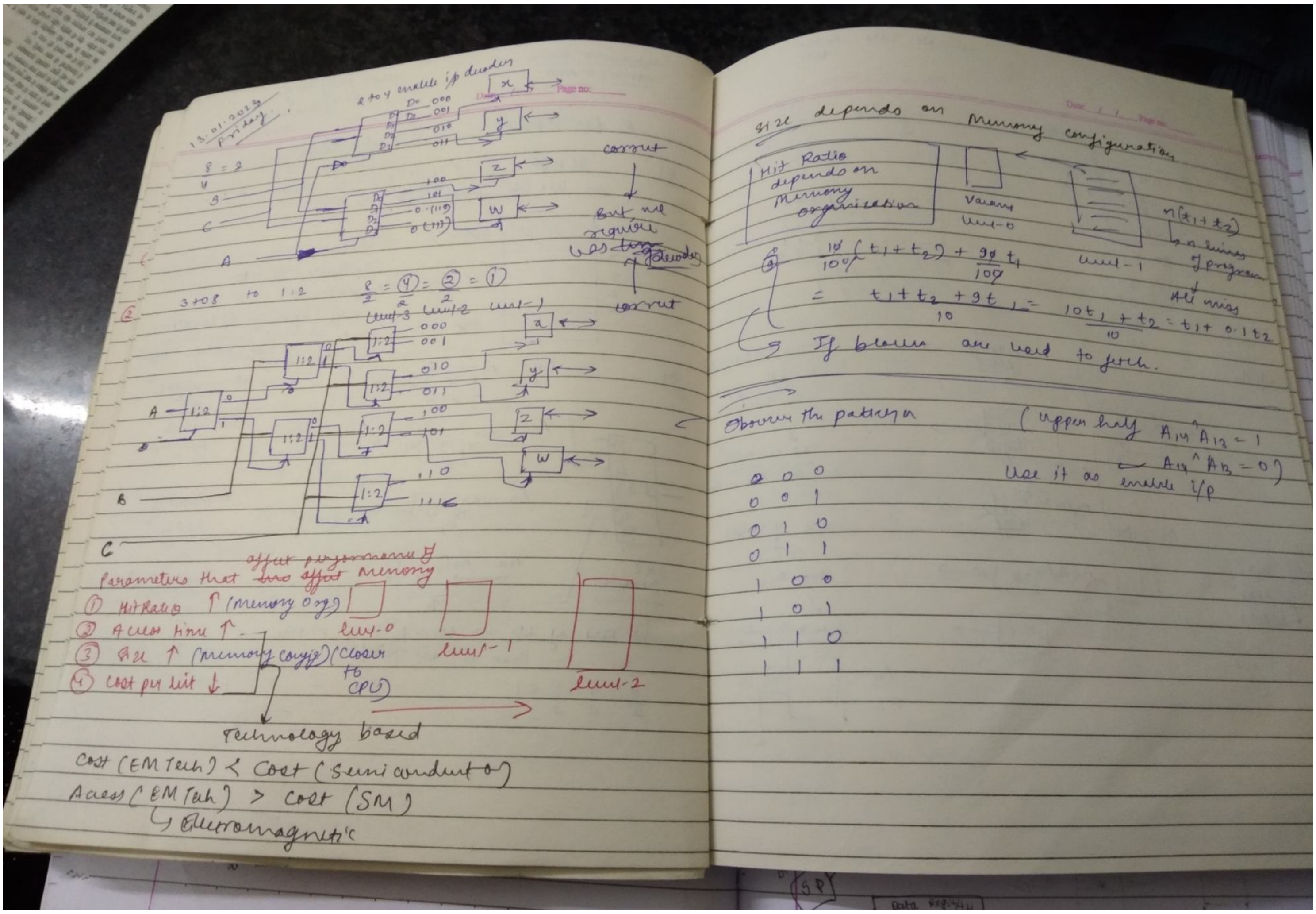
$2^{20} \times 1$ byte

$n = 20 \quad m = 8$ bits

[Generally $n > m$]

Drawback/Disadvantage :-

(1) Size of overhead is much much larger as compared to data



Parameters that ~~two~~ affect memory ~~affect performance of~~

① HitRatio ↑ (memory org)

② Access time ↑

③ Size ↑ (memory config) (closer to CPU)

④ cost per bit ↓

level-0 level-1 level-2

Technology based

Cost (EM Tech) < Cost (Semiconductor)

Acess (EM Tech) > Cost (SM)

↳ Electromagnetic

Size depends on Memory configuration

Hit Ratio depends on Memory organization

Vacant level-0

level-1

n(t1 + t2)

↳ n lines of program

↓

All miss

$\frac{10}{100}(t_1 + t_2) + \frac{90}{100}t_1$

$= \frac{t_1 + t_2 + 9t_1}{10} = \frac{10t_1 + t_2}{10} = t_1 + 0.1t_2$

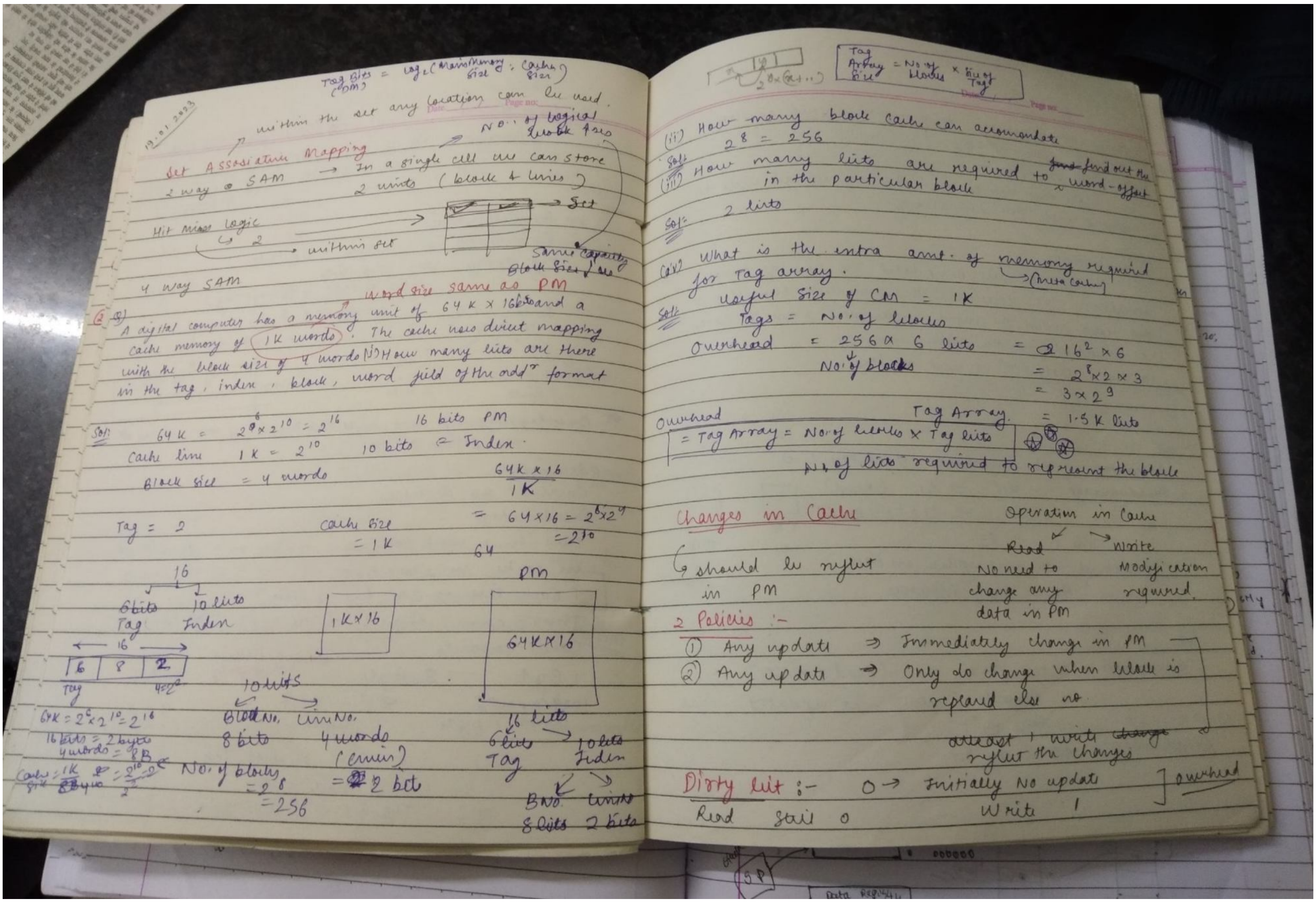
↳ If blocks are used to fetch.

Observe the pattern

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

(upper half $A_{14} \wedge A_{13} = 1$
$A_{14} \wedge A_{13} = 0$)

Use it as enable I/p



Tag Bits (DM) = $\log_2$(Main Memory Size ÷ Cache Size)

19.01.2023

within the set any location can be used.

No. of logical block = no

Set Assosiative Mapping

2 way SAM → In a single cell we can store 2 units (block & lines)

→ Set

Hit miss logic ↳ 2 → within set

same capacity Block size / no

4 way SAM

word size same as PM

(2) Q) A digital computer has a memory unit of 64 K × 16 bits and a cache memory of 1K words. The cache uses direct mapping with the block size of 4 words (i) How many bits are there in the tag, index, block, word field of the addr format

Sol: 64 K = $2^6 \times 2^{10} = 2^{16}$ 16 bits PM

Cache line 1 K = 2^{10} 10 bits = Index.

Block size = 4 words

$\frac{64K \times 16}{1K}$ $= 64 \times 16 = 2^6 \times 2^4 = 2^{10}$

Tag = 2

Cache size = 1K

64

16 → 6 bits Tag, 10 bits Index

1K×16

PM 64K×16

← 16 →

6	8	2
Tag		$4 = 2^2$

10 bits → Block No. 8 bits, Line No. 4 words (min) = 2 bit

$64K = 2^6 \times 2^{10} = 2^{16}$

16 bits = 2 bytes

4 words = 8B

Cache size $= \frac{1K}{8 Bytes} = \frac{2^{10}}{2^3} = 2^7$ No. of blocks $= 2^8 = 256$

16 bits → 6 bits Tag, 10 bits Index → BNo. 8 bits, Line No 2 bits

Tag Array Size = No. of blocks × Size of Tag

$2^8 \times (x+ ..)$

(ii) How many block cache can accomondate

Sol: $2^8 = 256$

(iii) How many bits are required to ~~find~~ find out the word-offset in the particular block

Sol: 2 bits

(iv) What is the extra amt. of memory required for Tag array. ↳(Meta Caching)

Sol: useful size of CM = 1K

Tags = No. of blocks

Overhead = 256 × 6 bits $= 16^2 \times 6$

No. of blocks $= 2^8 \times 2 \times 3$

$= 3 \times 2^9$

Tag Array = 1.5 K bits

Overhead

$\boxed{= \text{Tag Array} = \text{No. of blocks} \times \text{Tag bits}}$

No. of bits required to represent the block

Changes in Cache

↳ should be reflect in PM

Operation in Cache → Read: No need to change any data in PM; Write: Modification required.

2 Policies :-

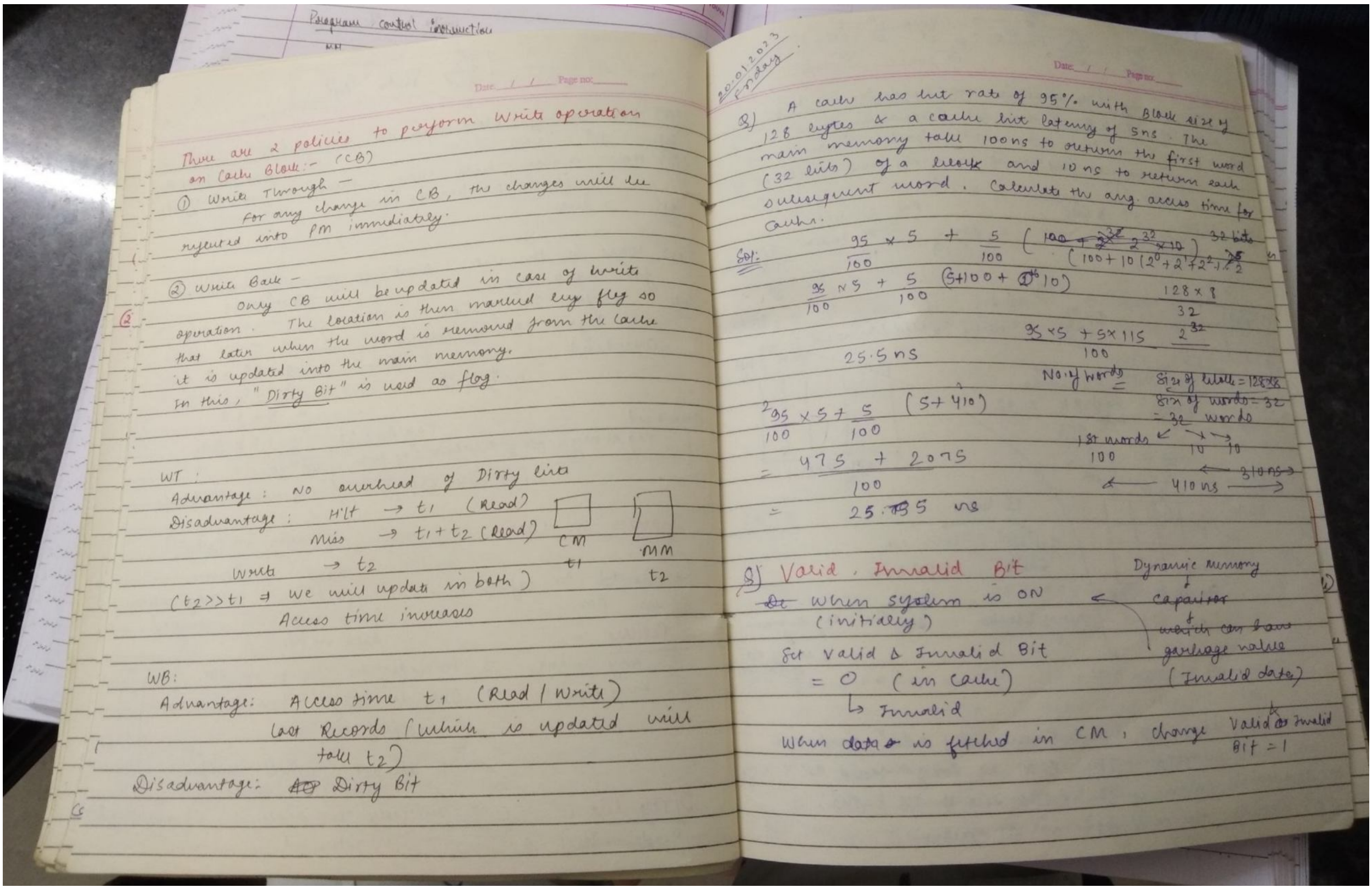
① Any update ⇒ Immediately change in PM

② Any update ⇒ Only do change when block is replaced else no

~~atleast 1 write change~~ reflect the changes

Dirty bit :- 0 → Initially No update] overhead

Read still 0   Write 1



Program control instruction

Date: / / Page no:

There are 2 policies to perform Write operation on Cache Block:- (CB)

① Write Through –

For any change in CB, the changes will be reflected into PM immediately.

② Write Back –

Only CB will be updated in case of write operation. The location is then marked by flag so that later when the word is removed from the cache it is updated into the main memory.

In this, "Dirty Bit" is used as flag.

WT :

Advantage : No overhead of Dirty bits

Disadvantage : Hit → t_1 (Read)

Miss → $t_1 + t_2$ (Read)

Write → t_2

($t_2 >> t_1$ ∵ we will update in both)

Access time increases

CM t_1 MM t_2

WB:

Advantage: Access time t_1 (Read / Write)

Last Records (which is updated will take t_2)

Disadvantage: ~~AO~~ Dirty Bit

20.01.2023 Friday.

Date: / / Page no:

Q) A cache has hit rate of 95% with Block size of 128 bytes & a cache hit latency of 5ns. The main memory take 100ns to return the first word (32 bits) of a block and 10 ns to return each subsequent word. Calculate the avg. access time for cache.

Sol:

$$\frac{95}{100} \times 5 + \frac{5}{100}\left(\cancel{100 + 2^{32}} \; \cancel{2^{32} \times 10}\right) \quad (100 + 10(2^0 + 2^1 + 2^2 + \ldots 2^{\cancel{25}}) \quad 32 \text{ bits}$$

$$\frac{95}{100} \times 5 + \frac{5}{100}(5 + 100 + \cancel{1}^{6}\, 10)$$

$$\frac{128 \times 8}{32}$$

$$\frac{95 \times 5 + 5 \times 115}{100} \qquad \underline{2^{32}}$$

25.5 ns

No. of words = Size of block = 128×8

Size of words = 32

= 32 words

1st words ← 100, → 10 → 10

← 310 ns →

← 410 ns →

$$\frac{^{2}95}{100} \times 5 + \frac{5}{100}(5 + \overset{?}{4}10)$$

$$= \frac{475 + 2075}{100}$$

$$= 25.~~75~~5 \text{ ns}$$

Q) Valid, Invalid Bit

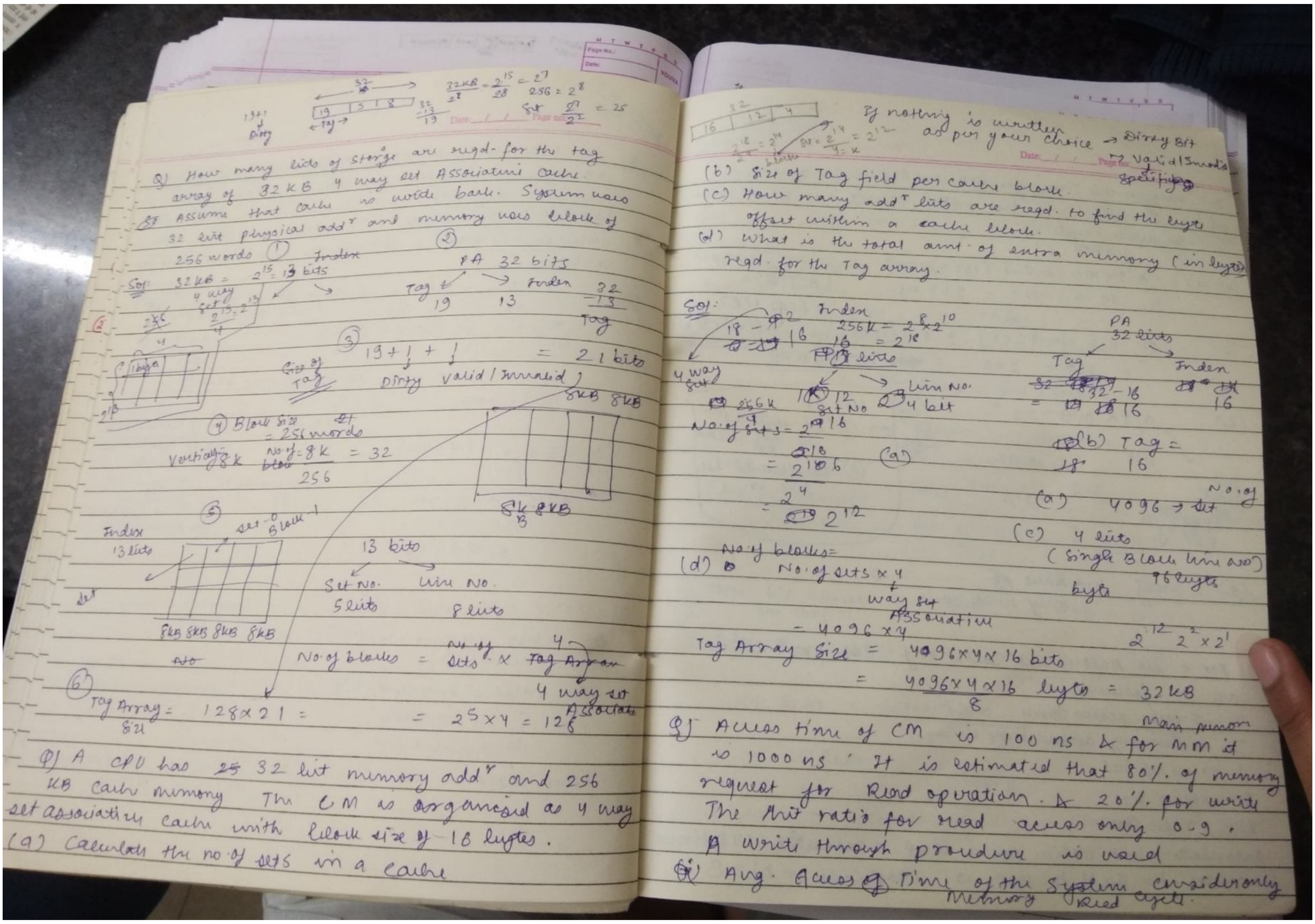
~~At~~ When system is ON (initially)

Set valid & Invalid Bit = 0 (in cache)

↳ Invalid

Dynamic memory ↓ ~~capacitor~~ ↓ ~~which can have~~ garbage value (Invalid data)

When data ~~s~~ is fetched in CM, change Valid ~~or~~ & Invalid Bit = 1



1 9 + 1 Dirty ← Tag → 19 | 5 | 8 ← 32 → $\frac{32KB}{2^8} = \frac{2^{15}}{28} = 2^7$ 256 = 2^8 32 − 13 = 19 Set $\frac{2^7}{2^2}$ = 25

Q) How many bits of storage are reqd. for the tag array of 32 KB 4 way set Associative cache. Assume that cache is write back. System uses 32 bit physical addr and memory uses block of 256 words

Sol: ① 32 KB = 2^{15} = 13 bits Index

4 way set $\frac{2^{15}}{4} = 2^{13}$

② PA 32 bits → Tag 19, Index 13   32 − 13 = Tag

③ Size of Tag: 19 + 1 + 1 = 21 bits (Dirty, valid / Invalid)

④ Block size = 256 words

Vertically 8k   No. of block = $\frac{8k}{256}$ = 32

8KB 8KB 8KB 8KB

⑤ Index 13 bits, set-0 Block-1, set

13 bits → Set No. 5 bits, Line No. 8 bits

8KB 8KB 8KB 8KB

No. of blocks = No. of sets × 4 ~~Tag Array~~ 4 way set Associative = $2^5 \times 4$ = 128

⑥ Tag Array size = 128 × 21 =

Q) A CPU has ~~25~~ 32 bit memory addr and 256 KB cache memory. The CM is organised as 4 way set associative cache with block size of 16 bytes.
(a) Calculate the no. of sets in a cache
(b) Size of Tag field per cache block.
(c) How many addr bits are reqd. to find the byte offset within a cache block.
(d) What is the total amt. of extra memory (in bytes) reqd. for the Tag array.

16 | 12 | 4 (32) $\frac{2^{18}}{2^4} = 2^{14}$ blocks   Set = $\frac{2^{14}}{4} = 2^{12}$

If nothing is written as per your choice → Dirty Bit → Valid / Invalid specifying

Sol: Index 256k = $2^8 \times 2^{10}$ = 2^{18} 18 − 2 = 16

18 bits → 12 Set No, 4 bit Line No.   4 way set

No. of sets = $\frac{256k}{4}$ = $\frac{2^{18}}{2^4 \cdot 2^2}$ = 2^{12} (a)

PA 32 bits → Tag, Index   = 32 − 16 = 16   Index 16

(b) Tag = 16

(a) 4096 → No. of set

(c) 4 bits (single Block line no) byte 16 bytes

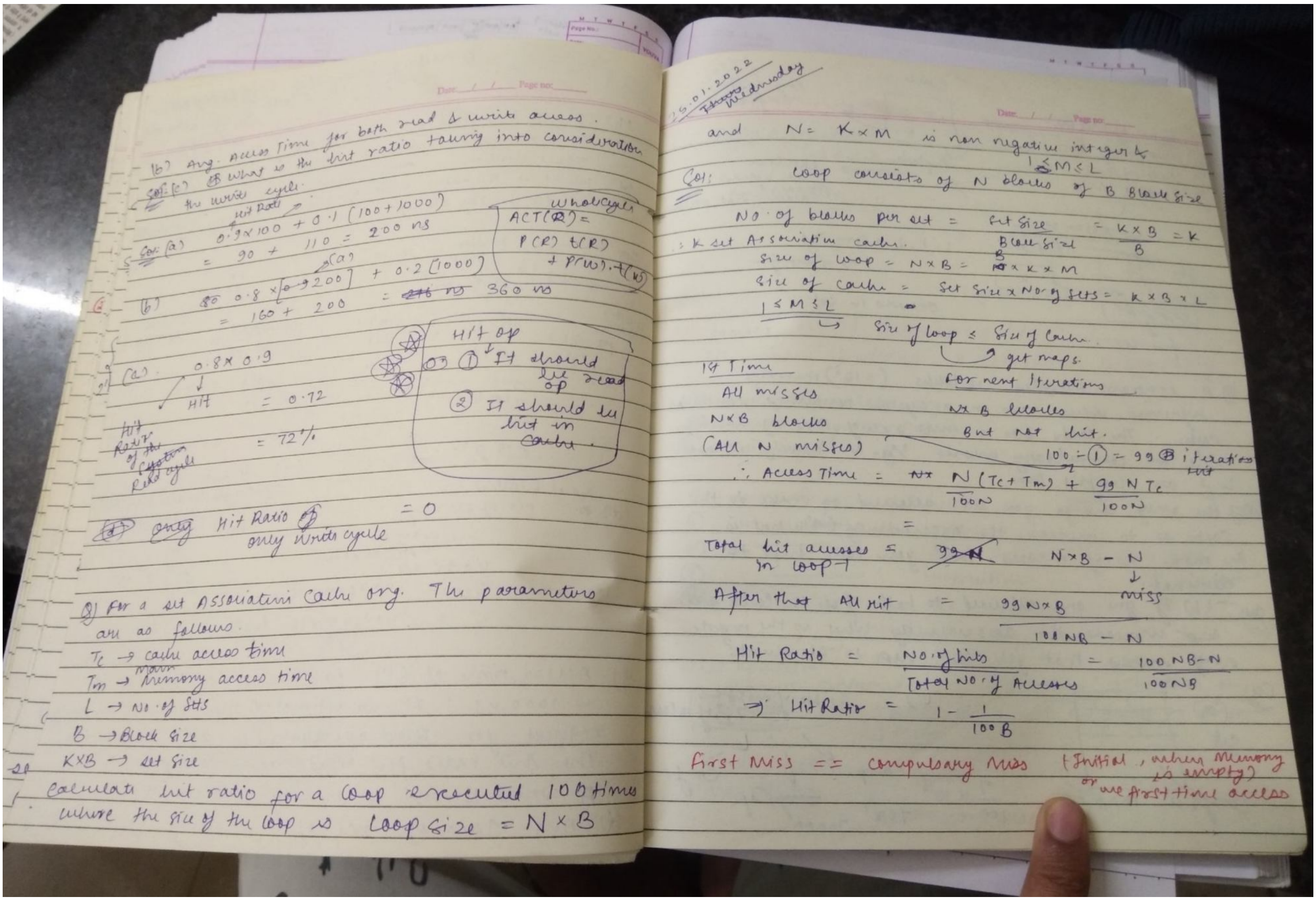
(d) No. of blocks = No. of sets × 4 (way set Associative) = 4096 × 4

$2^{12} \; 2^2 \times 2^1$

Tag Array Size = 4096 × 4 × 16 bits

= $\frac{4096 \times 4 \times 16}{8}$ bytes = 32 KB

Q) Access time of CM is 100 ns & for MM (Main memory) it is 1000 ns. It is estimated that 80% of memory request for Read operation & 20% for write. The hit ratio for read access only 0.9. A write through procedure is used.
(i) Avg. Access Time of the System considering only Memory Read cycle.



(b) Avg. Access Time for both read & write access.

Sol: (c) ~~Ø~~ What is the hit ratio taking into consideration the write cycle.

hit Ratio

Sol: (a) $0.9 \times 100 + 0.1\,(100+1000)$

$= 90 + 110 = 200$ ns

(a)

(b) ~~80~~ $0.8 \times [\cancel{0.9}\ 200] + 0.2\,[1000]$ = ~~216~~ ns 360 ns

$= 160 + 200$

Whole cycle

$ACT(R) = P(R)\,t(R) + P(W).t(W)$

(a) 0.8×0.9

↓

Hit $= 0.72$

Hit Ratio of ~~the system~~ Read cycle $= 72\%$

Hit op

① It should be read op

② It should be hit in cache.

(d) ~~only~~ Hit Ratio ~~of~~ only write cycle $= 0$

Q) For a set Associative Cache org. The parameters are as follows.

T_c → cache access time

T_m → main memory access time

L → No. of sets

B → Block size

K×B → set size

Calculate hit ratio for a loop executed 100 times where the size of the loop is Loop size $= N \times B$

25.01.2022 ~~Thursday~~ Wednesday

and $N = K \times M$ is non negative integer & $1 \leq M \leq L$

Sol: Loop consists of N blocks of B Block size

No. of blocks per set $= \frac{\text{Set size}}{\text{Block size}} = \frac{K \times B}{B} = K$

∴ K set Associative cache.

Size of loop $= N \times B = B \times K \times M$

Size of cache = Set size × No. of sets $= K \times B \times L$

$1 \leq M \leq L$ → Size of loop ≤ Size of cache → get maps.

1st Time

All misses

N×B blocks

(All N misses)

For next Iterations

N× B blocks

But not hit.

$100 - ① = 99$ iterations hit

∴ Access Time = ~~N×~~ $\frac{N\,(T_c+T_m)}{100N} + \frac{99\,N\,T_c}{100N}$

=

Total hit accesses in loop-1 = ~~99 N~~ $N \times B - N$

↓

miss

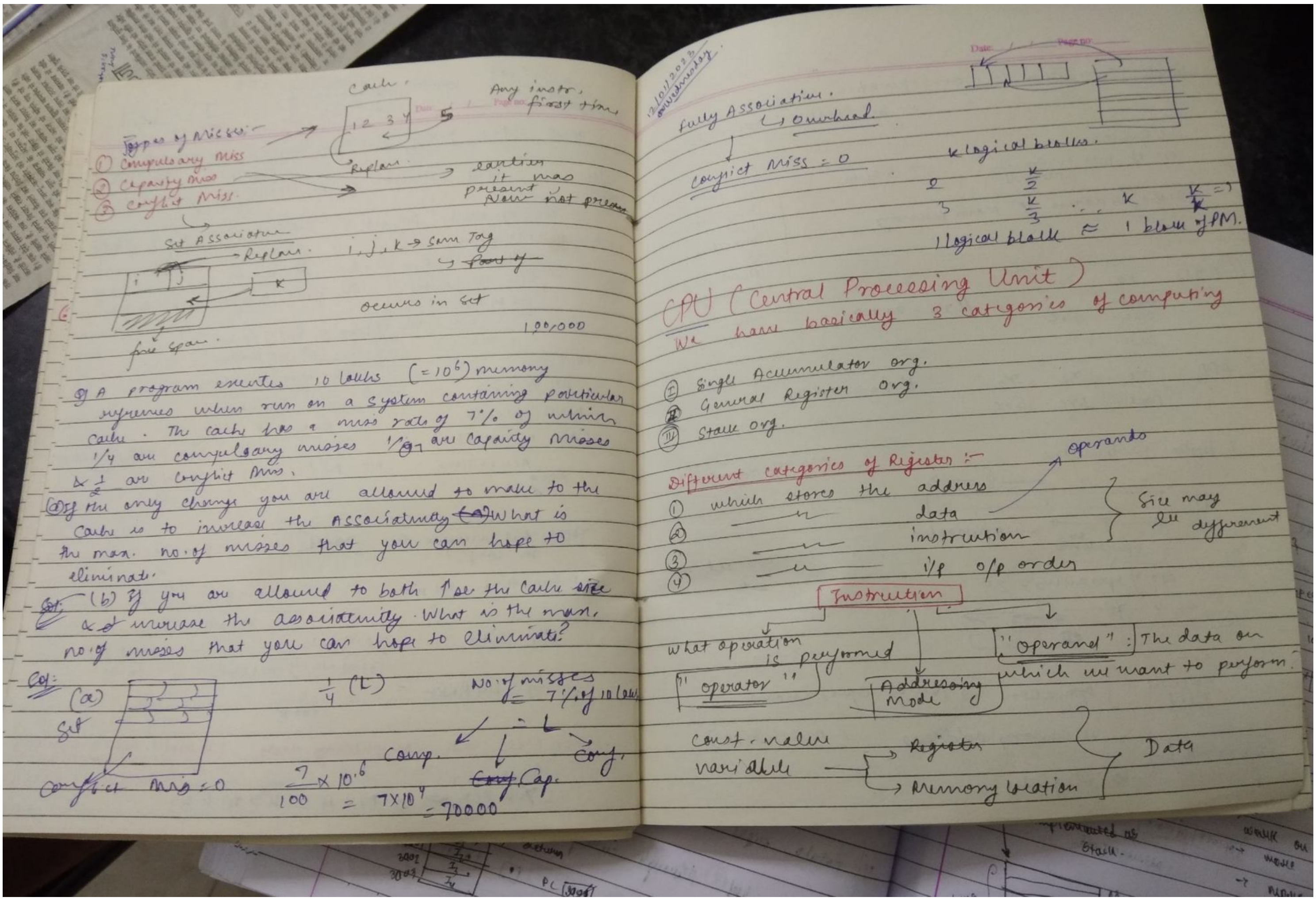
After that All hit $= 99\,N \times B$

$100\,NB - N$

Hit Ratio $= \frac{\text{No. of hits}}{\text{Total No. of Accesses}} = \frac{100\,NB - N}{100\,NB}$

⇒ Hit Ratio $= 1 - \frac{1}{100B}$

First Miss == Compulsory Miss (Initial, when Memory is empty) or we first time access



Types of Misses :-

① Compulsory Miss

② Capacity Miss

③ Conflict Miss.

Cache.

1 2 3 4 5

Replace.

Any instr. first time

earlier it was present, Now not present

Set Associative

Replace.

i j k

free space.

i, j, k → same Tag
↳ ~~Part of~~

occurs in set

1,00,000

Q) A program executes 10 lakhs $(=10^6)$ memory references when run on a system containing particular cache. The cache has a miss rate of 7% of which 1/4 are compulsory misses 1/~~8~~ are Capacity Misses & $\frac{1}{2}$ are conflict Miss.

(a) If the only change you are allowed to make to the cache is to increase the Associativity ~~(a)~~ What is the max. no. of misses that you can hope to eliminate.

Sol: (b) If you are allowed to both ↑se the cache size & ~~d~~ increase the associativity. What is the max. no. of misses that you can hope to eliminate?

Sol:

(a)

Set

Conflict Miss = 0

$\frac{1}{4}$ (L)

No. of misses = 7% of 10 lakh

= L

Comp.

~~Conf~~ Cap.

Conf.

$\frac{7}{100} \times 10^6$

$= 7 \times 10^4 = 70000$

12/01/2023
Wednesday

Fully Associative.
↳ Overhead.

Conflict Miss = 0

k logical blocks.

2 $\frac{k}{2}$

3 $\frac{k}{3}$ ... k $\frac{k}{k}$ ⇒

1 logical block ≃ 1 block of PM.

CPU (Central Processing Unit)

We have basically 3 categories of computing

Ⓘ Single Accumulator org.

Ⓘⓘ General Register org.

Ⓘⓘⓘ Stack org.

Different categories of Register :-

① which stores the address

② — " — data → operands

③ — " — instruction

④ — " — i/p o/p order

} Size may be different

Instruction

what operation is performed
"operator"

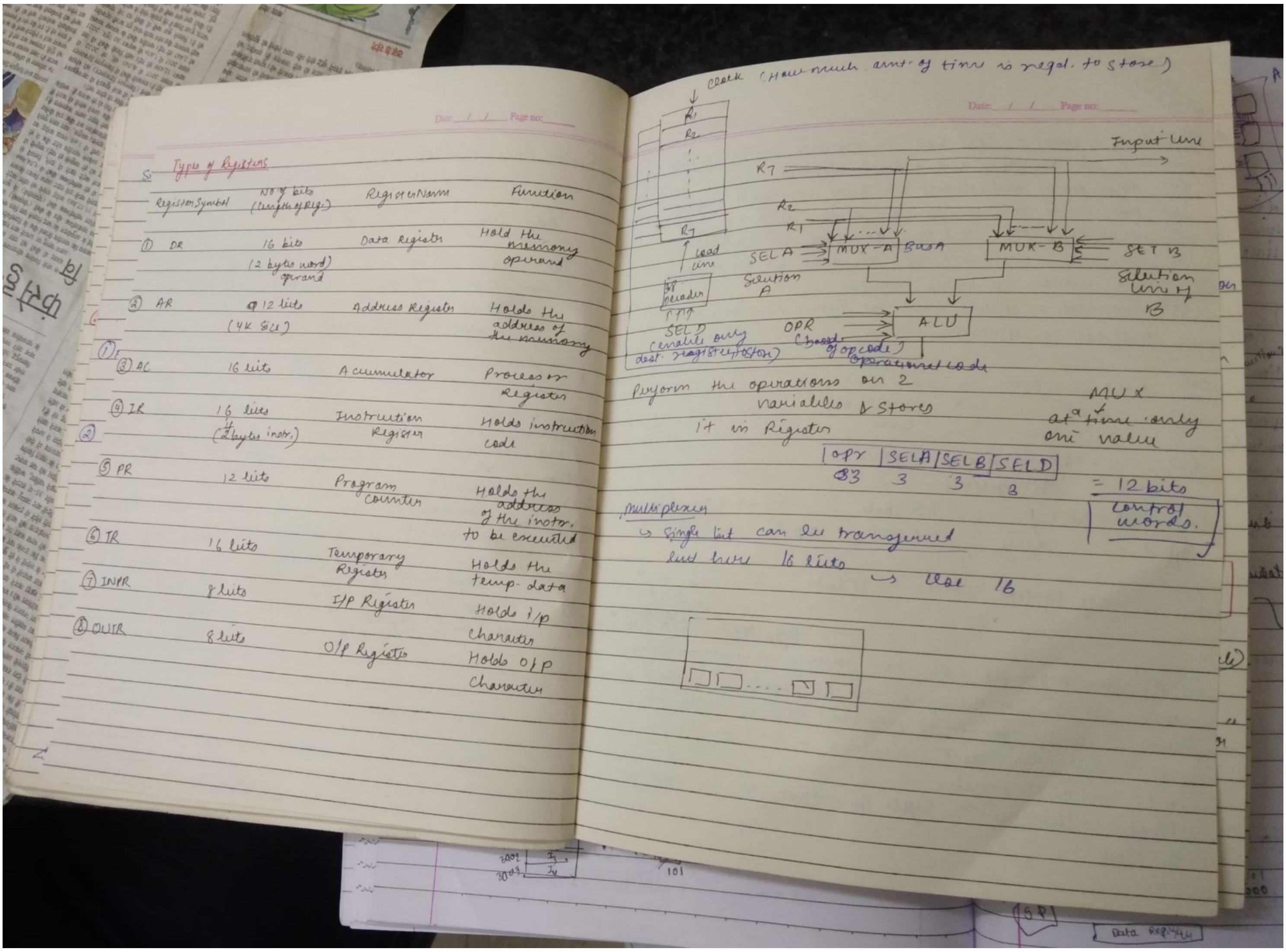
Addressing Mode

"Operand" : The data on which we want to perform.

Const. value

variable → Register / Memory location

} Data



Date: / / Page no:
clock (How much amt. of time is reqd. to store)
R1
R2
R7
load line
3:8 decoder
SELD (enable only dest. register/(storing))
R7
R2
R1
SEL A
MUX-A
Bus A
Selection A
MUX-B
SET B
Selection line of B
Input line
OPR (based of opcode) Operational code
ALU

Types of Registers

Register Symbol	No of bits (length of Regs)	Register Name	Function
① DR	16 bits (2 bytes word) operand	Data Register	Hold the memory operand
② AR	12 bits (4K size)	Address Register	Holds the address of the memory
③ AC	16 bits	Accumulator	Processor Register
④ IR	16 bits (2 bytes instr.)	Instruction Register	Holds instruction code
⑤ PR	12 bits	Program counter	Holds the address of the instr. to be executed
⑥ TR	16 bits	Temporary Register	Holds the temp-data
⑦ INPR	8 bits	I/P Register	Holds i/p character
⑧ OUTR	8 bits	O/P Register	Holds O/P character

Perform the operations on 2 variables & stores it in Register

MUX at a time only one value

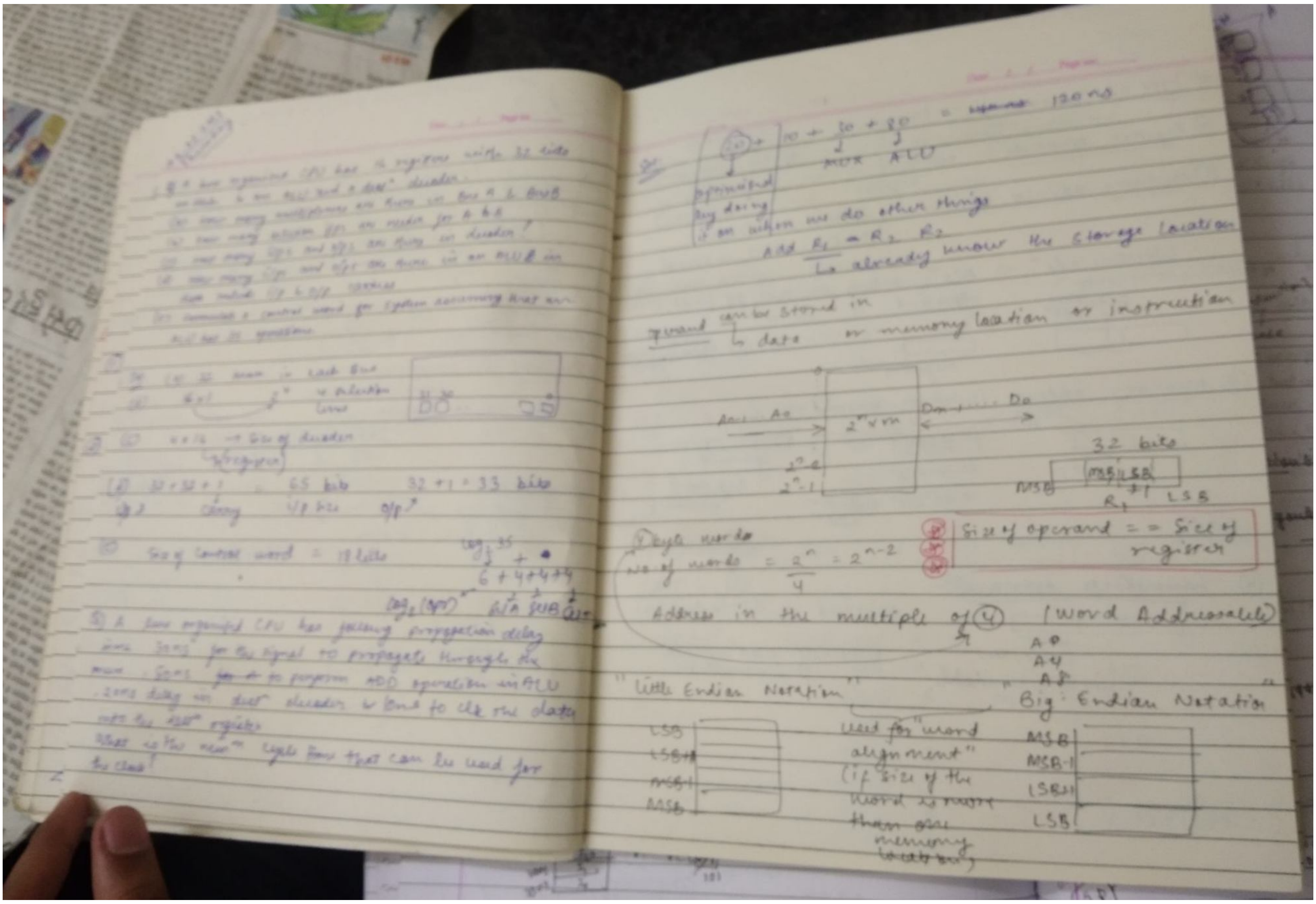
opr	SELA	SELB	SELD
3	3	3	3

= 12 bits control words.

Multiplexer

↳ Single bit can be transferred but here 16 bits

↳ use 16



Q. A bus organised CPU has 16 registers with 32 bits in each, 1 an ALU and a dest^n decoder.

(a) How many multiplexers are there in bus A & bus B

(b) How many selection I/Ps are needed for A & B

(c) How many I/Ps and O/Ps are there in decoder?

(d) How many I/Ps and O/Ps are there in an ALU in data include I/p & O/p carries

(e) Formulate a control word for system assuming that an ALU has 35 operations

(a) 32 mux in each bus

(b) 4×1 2^4 4 selection lines

(c) 4×16 → size of decoder (16 registers)

(d) 32+32+1 = 65 bits 32+1 = 33 bits

i/p carry i/p bits o/p

(e) Size of control word = 18 bits

$\log_2 35$ + 6+4+4+4

$\log_2(35)$ A/A SUB dest

Q A bus organised CPU has following propagation delay times 30ns for the signal to propagate through the mux, 80ns ~~for it~~ to perform ADD operation in ALU, 20ns delay in dest^n decoder & 10ns to clk the data into the dest^n register.

What is the min^m cycle time that can be used for the clock?

Sol: 10 + 30 + 80 = ~~[illegible]~~ 120ns

MUX ALU

optimised by doing it on when we do other things

Add $R_1 \leftarrow R_2, R_2$

↳ already know the storage location

operand can be stored in

↳ data or memory location or instruction

$A_{n-1} \ldots A_0$ $2^n \times m$ $D_{m-1} \ldots D_0$

0 2^n-2 2^n-1

32 bits

MSB MSB LSB LSB

R_1

4 byte words

no. of words = $\frac{2^n}{4} = 2^{n-2}$

Size of operand == Size of register

Address in the multiple of 4 (Word Addressable)

A0

A4

A8

"Little Endian Notation"

LSB

LSB+1

MSB-1

MSB

used for "word alignment" (if size of the word is more than one memory location)

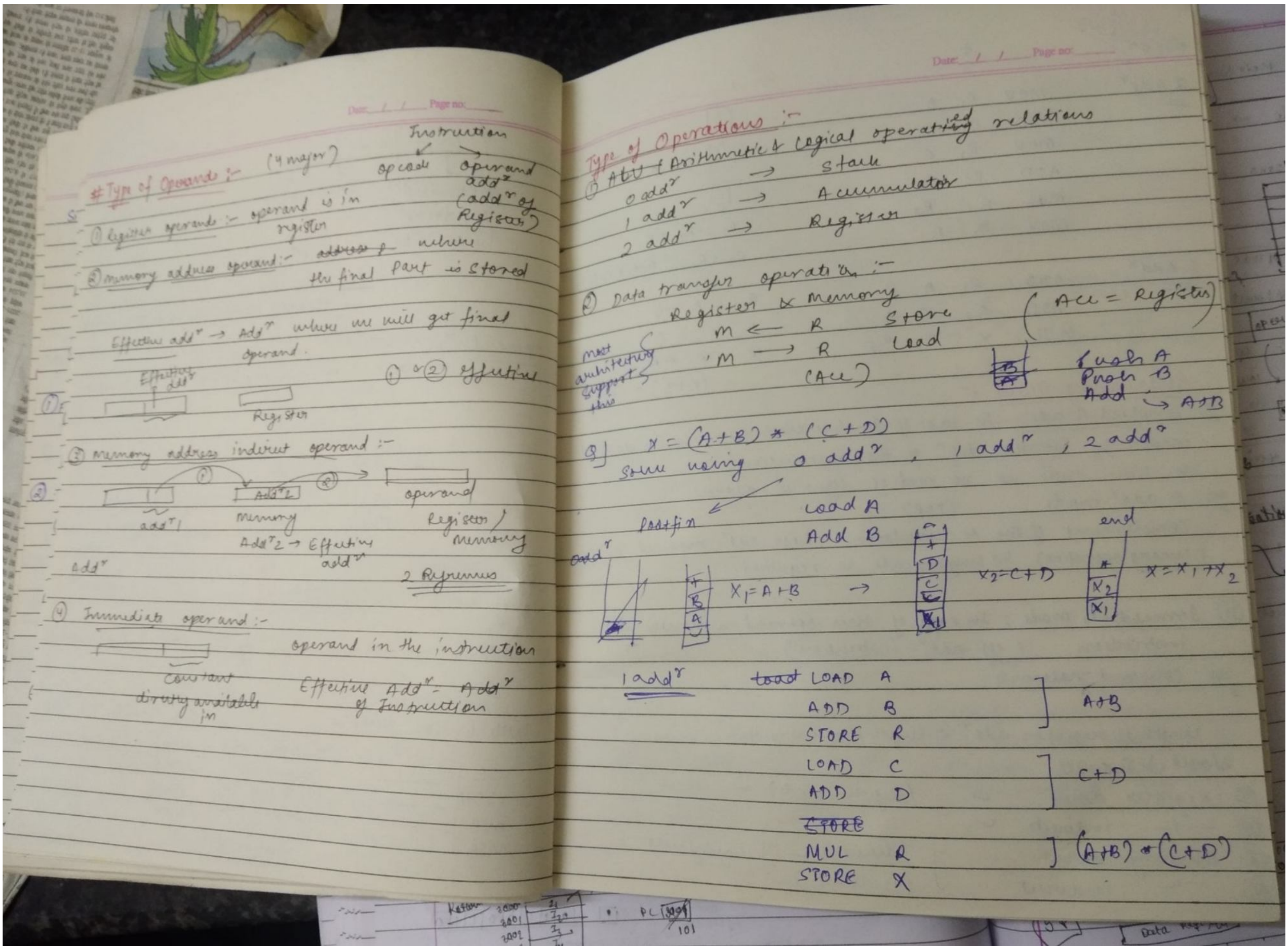
"Big Endian Notation"

MSB

MSB-1

LSB+1

LSB



Date: / / Page no:

Type of Operand :- (4 major)

Instruction

opcode → operand add^r (add^r of Register)

(1) Register operand :- operand is in register

(2) memory address operand :- ~~address &~~ where the final part is stored

Effective add^r → Add^r where we will get final operand.

(1) & (2) effective

Effective add^r
Register

(3) memory address indirect operand :-

(1)
(2)
Add^r 2
add^r 1
memory
operand
Register / memory

Add^r 2 → Effective add^r

Add^r

2 References

(4) Immediate operand :-

operand in the instruction

Effective Add^r = Add^r of Instruction

Date: / / Page no:

Type of Operations :-

(1) ~~ALU~~ (Arithmetic & logical operated relations

0 add^r → Stack

1 add^r → Accumulator

2 add^r → Register

(2) Data transfer operation :-

Register & Memory

most architecture support this { M ← R Store ; M → R Load (Acc) }

(Acc = Register)

Push A
Push B
Add → A+B

Q] $x = (A+B) * (C+D)$
Solve using 0 add^r , 1 add^r , 2 add^r

postfix

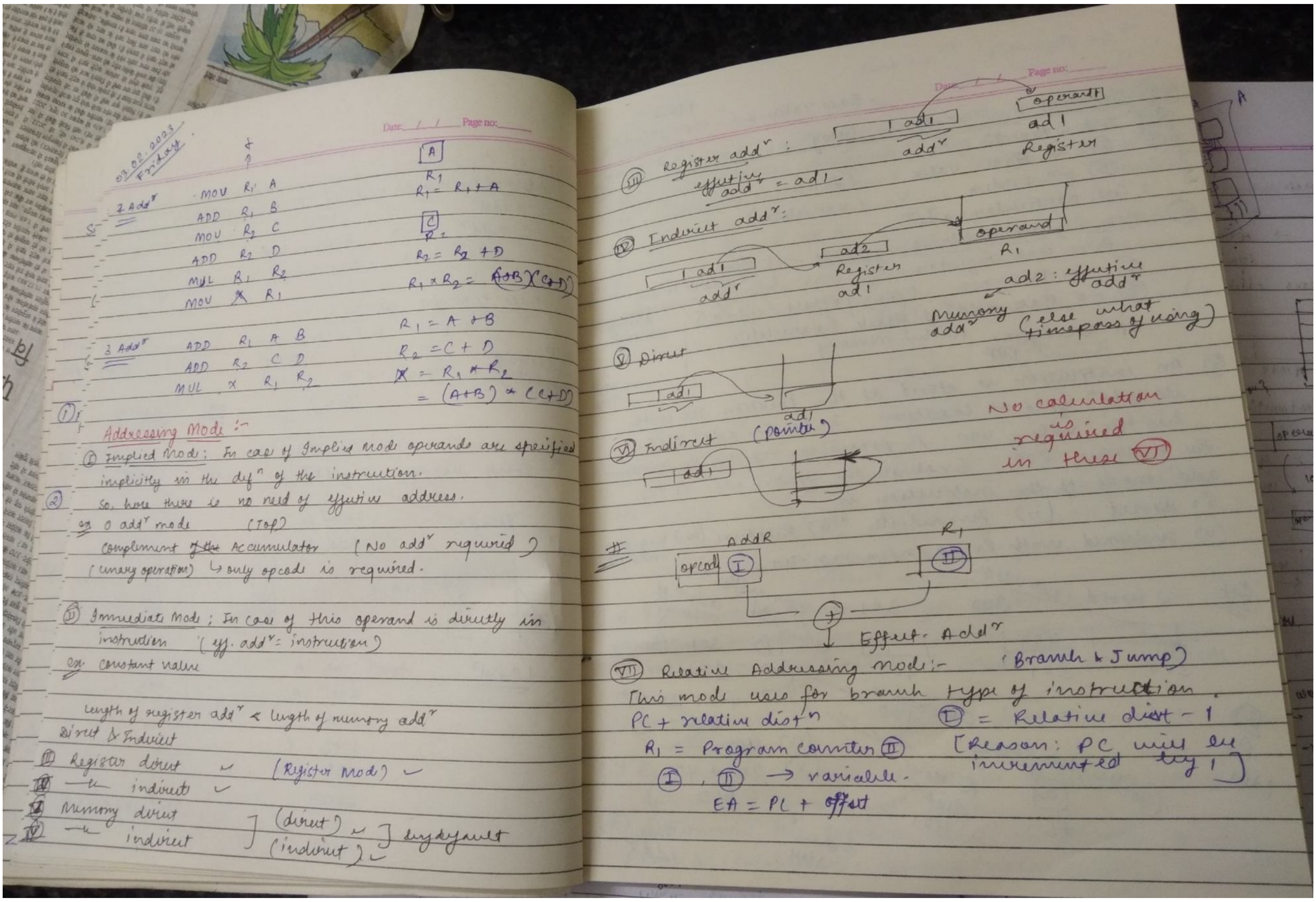
~~0 add~~^r

Load A
Add B

$X_1 = A+B$ → $X_2 = C+D$ $X = X_1 + X_2$

1 add^r

```
~~load~~ LOAD  A   ]
ADD   B            ]  A+B
STORE R
LOAD  C   ]
ADD   D   ]  C+D
~~STORE~~
MUL   R   ]  (A+B) * (C+D)
STORE X
```



03.02.2023
Friday

Date: / / Page no:

2 Add^r

```
MOV  R1  A        A
                  R1
                  R1 = R1 + A
ADD  R1  B
MOV  R2  C        C
                  R2
ADD  R2  D        R2 = R2 + D
MUL  R1  R2       R1 * R2 = (A+B)(C+D)
MOV  X   R1
```

3 Add^r

```
ADD  R1  A  B     R1 = A + B
ADD  R2  C  D     R2 = C + D
MUL  X   R1 R2    X = R1 * R2
                    = (A+B) * (C+D)
```

Addressing Mode :-

(I) Implied Mode: In case of Implied mode operands are specified implicitly in the def^n of the instruction.

So, here there is no need of effective address.

ex 0 add^r mode (Top)

Complement ~~of the~~ Accumulator (No add^r required)

(unary operation) ↳ only opcode is required.

(II) Immediate Mode; In case of this operand is directly in instruction (eff. add^r = instruction)

ex. constant value

Length of register add^r < length of memory add^r

Direct & Indirect

(III) Register direct ✓ (Register Mode) ✓

(IV) — " — indirect ✓

(V) Memory direct] (direct) ✓] ~~[illegible]~~

(VI) — " — indirect] (indirect) ✓

(III) Register add^r :

ad1 — add^r — operand — ad1 — Register

effective add^r = ad1

(IV) Indirect add^r :

ad1 — add^r — ad2 — Register ad1 — operand — R1

ad2 : effective add^r

Memory add^r (else what timepass of using)

(V) Direct

ad1 — ad1

(VI) Indirect

ad1 — ad1 (pointer)

No calculation is required in these (VI)

#

ADDR

opcode (I) — R1 (II) — + — Effect. Add^r

(VII) Relative Addressing mode:- (Branch & Jump)

This mode uses for branch type of instruction.

PC + relative dist^n

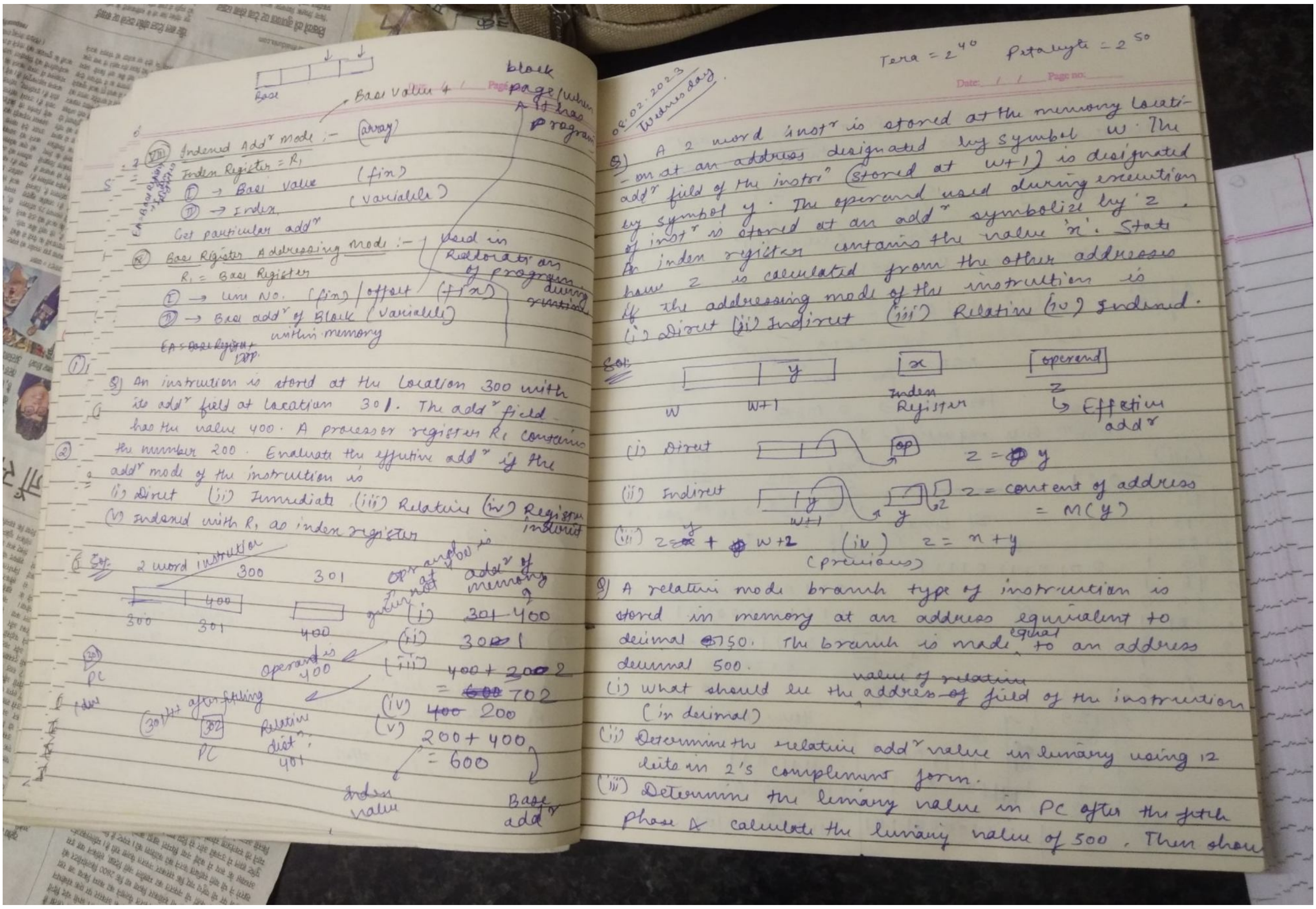
(I) = Relative dist - 1

R1 = Program counter (II)

[Reason: PC will be incremented by 1]

(I), (II) → variable.

EA = PC + offset



Base

Base value 4

block page (when A it has program)

(VIII) Indexed Addr mode :- (array)

Index Register = R_1

(I) → Base value (fix)

(II) → Index (variable)

Get particular addr

EA = Base Register + Index

(IX) Base Register Addressing mode :- Used in Relocation of program during execution

R_1 = Base Register

(I) → line No. (fix) / offset (fix)

(II) → Base addr of Block (variable) within memory

~~EA = Base Register +~~ DISP.

Q) An instruction is stored at the Location 300 with its addr field at Location 301. The addr field has the value 400. A processor register R_1 contains the number 200. Evaluate the effective addr if the addr mode of the instruction is
(i) Direct (ii) Immediate (iii) Relative (iv) Register indirect (v) Indexed with R_1 as index register

Sol. 2 word instruction

300 301

400

300 301

400

Opr anyhow is at addr of memory given not ?

(i) ~~301~~ 400

(ii) 30~~2~~1

(iii) 400 + ~~200~~ 2 = ~~600~~ 702

(iv) ~~400~~ 200

(v) 200 + 400 = 600

Index value

Base addr

operand is 400

(301) PC

$(301)^{++}$ after fetching 302 PC

Relative dist: 401

08.02.2023 Wednesday.

Tera = 2^{40} Petabyte = 2^{50}

Q) A 2 word instr is stored at the memory location at an address designated by symbol W. The addr field of the instrn (stored at W+1) is designated by symbol y. The operand used during execution of instr is stored at an addr symbolize by z. An index register contains the value 'x'. State how z is calculated from the other addresses if the addressing mode of the instruction is (i) Direct (ii) Indirect (iii) Relative (iv) Indexed.

Sol: y

W W+1

x

Index Register

operand

z ↳ Effective addr

(i) Direct op $z = y$

(ii) Indirect y W+1 y z z = content of address = M(y)

(iii) $z = \overset{y}{\cancel{x}} + w + 2$ (previous)

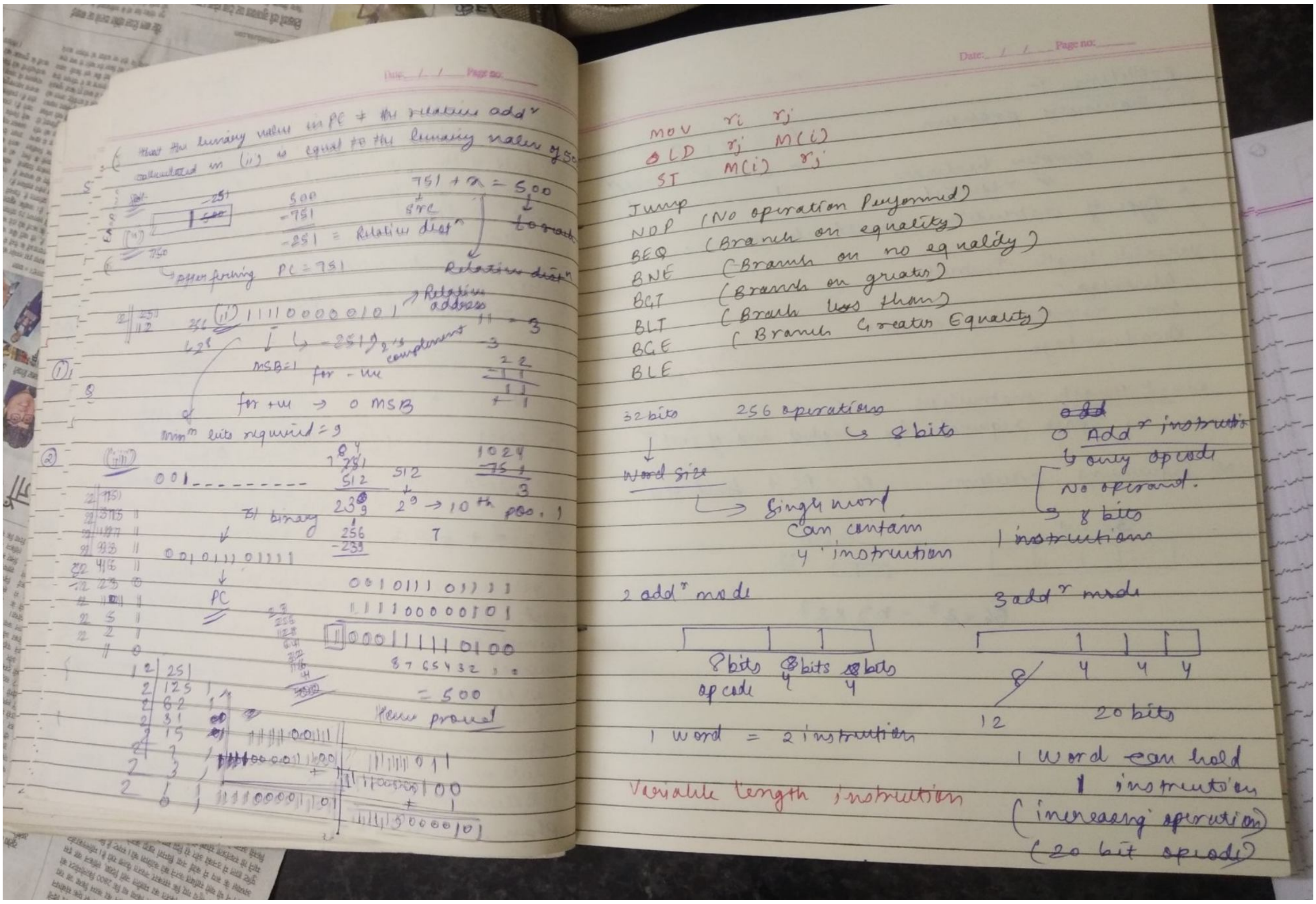
(iv) $z = x + y$

Q) A relative mode branch type of instruction is stored in memory at an address equivalent to decimal ~~7~~750. The branch is made equal to an address decimal 500.

(i) What should be the value of relative address ~~of~~ field of the instruction (in decimal)

(ii) Determine the relative addr value in binary using 12 bits in 2's compliment form.

(iii) Determine the binary value in PC after the fetch phase & calculate the binary value of 500. Then show



(i) that the binary value in PC + the relative addʳ calculated in (ii) is equal to the binary value of 500

751 + x = 500
↓ src
↓ ~~to reach~~

−251 ~~500~~

500
−751
−251 = Relative distⁿ

~~Relative distⁿ~~

↳ After fetching PC = 751

(ii) 1111 0000 0101 → Relative address

↳ −251 2's complement

11 → 3
−3
22
−11
11
+ 1

MSB=1 for −ve

for +ve → 0 MSB

minᵐ bits required = 9

(iii) 001 ----------

751
−512
239
256
−239

512 ↓ 2^9 → 10th pos. 1

1024
−751
3

7

751 binary ↓

0010111 01111 ↓ PC

2 | 751
2 | 375 1
2 | 187 1
2 | 93 1
2 | 46 1
2 | 23 0
2 | 11 1
2 | 5 1
2 | 2 1
1 0

2 | 251
2 | 125 1
2 | 62 1
2 | 31 0
2 | 15 1
2 | 7 1
2 | 3 1
2 | 1 1
0 1

111110 0111
0000 0011 1100
+
1111 0000 0101

1111111 011
1111 0000 0100
+ 1
1111 0000 0101

0010111 01111
1111 0000 0101
[1] 0001 1111 0100
8 7 6 5 4 3 2 1 0
= 500
Hence proved

MOV r_i r_j
LD r_j M(i)
ST M(i) r_j
Jump
NOP (No operation Performed)
BEQ (Branch on equality)
BNE (Branch on no equality)
BGT (Branch on greater)
BLT (Branch less than)
BGE (Branch Greater Equality)
BLE

32 bits → ~~Word size~~ ↳ Single word can contain 4 instruction

256 operations ↳ 8 bits

~~add~~
0 Addʳ ~~instructio~~
↳ only opcode
No operand.
↳ 8 bits
1 ~~instructions~~

2 addʳ mode

8 bits opcode | ~~8~~ 4 bits | ~~8~~ 4 bits

1 word = 2 instruction

3 addʳ mode

~~8~~ 12 | 4 | 4 | 4

20 bits

1 word can hold 1 instruction (increasing operation) (20 bit opcode)

Variable length instruction