execution time

processor performance equation.

$ET = N \times CPI \times \tau$ → 1/f.

IC instructions.

$CC = I_c \times CPI$

instruction counter.

$\frac{sec}{program} = \frac{Insts.}{Program\ insts.} \times \frac{cc}{} \times \frac{sec}{cc}$

$MIPS\ Rate = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6} = \frac{f \times I_c}{C \times 10^6}$

total cc of (cc)

Q. Consider the execution of an object code with 2×10^6 instruction on a ~~4 GHz~~ 400 MHz processor. The program consists of 4 major types of instruction. The instruction mix & no. of cycles (CPI) are as shown in table given below based on result of program based experiment. Calculate avg. CPI for implementation & calculate corresponding MIPS rate

Ans. → cycle per ~~inst~~ Instruction.

IT	cache hit	CPI	IM (instruction mix)
A & L	CH	1	60% (0.6)
L & S	CH	2	80% (0.8)
Br.		4	12% (0.12)
MR	CM	8	10% (0.1)
			1.60

cache miss.

$= 0.6 \times 1 + 0.8 \times 2 + 0.12 \times 4 + 0.1 \times 2$

$= 1.4 + 1.6 + 0.48$

$= \frac{3.48}{1.60} \approx 2.24$ cycle/instruction

CPI → 2.24

Ans.

$MIPS\ rate = \frac{400\ \text{~~MHz~~} \times 10^6}{2.24 \times 10^6} = \frac{178.}{\text{~~178.5~~}}$ million instruction/sec.

$$ET = (2 \times 10^6) \times (2.24) \times \left(\frac{1}{400 \times 10^6}\right)$$

$$= \frac{1.12}{400} \times \frac{4.48}{400} = 0.0112 \text{ seconds}$$

Q. ET (in seconds) of 4 program on 3 computers are given below

P ↓	3 comp. ET (in secs)		
	CA	CB	CC
P1	1	10	20
P2	1000	100	20
P3	500	1000	50
P4	100	800	100

Assume that 10^9 instr. are executed by each of the program. Calculate MIPS of each program on each of 3 m/c. Based on these ratings can you draw clear conclusion regarding their relative performance.

Q. Consider 3 process executing the same instruct^n set
P1 has clock rate 3 GHz & CPI 1.5.
P2 has clock rate 2.5 GHz & CPI 1
P3 " " " 4 GHz & CPI 2.2.

(a) Which processor has highest performance expressed in instr / seconds?

(b) if the pro. each execute a program in 10sec find no. of cycle & no. of instructions!

(c) We ~~are~~ are trying to reduce execut^n time by 30% while compromising CPI incr. by 20%. What clock rate should hv to have this reduction.

(a) P1 → $MIPS = \frac{3 \times 10^9}{1.5} = 2 \times 10^9$ instr. / seconds.

P2 → $= \frac{2.5 \times 10^9}{1} = 2.5 \times 10^9$

P3 → $= \frac{4 \times 10^9}{2.2} = 1.8 \times 10^9$

Hence, P2 is having highest performance in case of instr. / seconds.

(b) P1 → cycles. → $10 \times 1.5 \times 10^9$.

P2 → $10 \times 2.5 \times 10^9$

P3 → $10 \times 4 \times 10^9$

} cycles.

$\frac{40 \times 10^9}{22}$

100x + 100x

Page No.
Date

$P3 = \frac{15 \times 10^9}{1.5} = 10^{10}$

$P2 = \frac{25 \times 10^9}{1} = 25 \times 10^9$

$P3 = \frac{40 \times 10^9}{2.2} = 1.8 \times 10^{10}$

(c) (i) Pf

$f = \frac{N \times CPI}{ET}$

$f = 1.5x$

$CPI_n = CPI_0 \times 1.2 + CPI_0 \times 0.3$

$= CPI_0 \times (1.2)$

ET → 10sec → 30% reduction → ET_n = 7sec.

$f = \frac{N \times CPI}{ET} = \frac{10^{10} \times 1.8}{7}$

quadword ... why mov reg, mem not possible

RISC-V → Little endian (Least significant at lower address)

Register faster to access (as limited in no.)

Register Immediate Store C ?

RISC-V

dest → 2 operands

3 ADD R1, R2, R3 ← R1 → R2 + R3
2 ADD R5, R2. → R5 → R5 + R2
1 ADD R6
0 ADD

~~AX = AX + R6.~~

Z = K + (B * C).

3 operand instr

MUL R1, B, C
ADD Z, K, R1

~~ADD Z, K, B*C.~~

2 operands.

~~MUL R1,~~
MOV ~~R1, B~~ Z, C
MUL ~~R1, C~~ Z, B
~~ADD MOV R2,~~ ADD Z, K.

mov Z, C → Z → C
MUL Z, B → Z → B*
ADD Z, K.
↳ Z = K + B*C

1 operands -

~~LOD~~ ~~MOV~~ AX, K
~~STOS~~ ~~MOV~~ Y, AX

ldr C (AX → C)
mul B (AX → B*C)
add K (AX → K + B*C)
~~mo~~ ~~[illegible]~~
st Z. (Z → AX)

RISC-V

24-30-comp

Page No.
Date

- A 0 operands-

  $Z = K + B * C$

  postfix K B C * +

  Push K
  Push B
  Push C
  MUL → POPs C & B, multiplies & again push on stack
  ADD → POPs B*C & A, adds, push on stack
  POP Z.

- ex. of little endian & big endian Architecture

  IS-A 's → x86 (RISC ← CISC)
  MIPS
  ARM
  RISC-V
  SPARC
  } instruction set architecture

stage 1. [instr. fetch.]

stage 2 [Src. registers]

stage 3 [ALU]

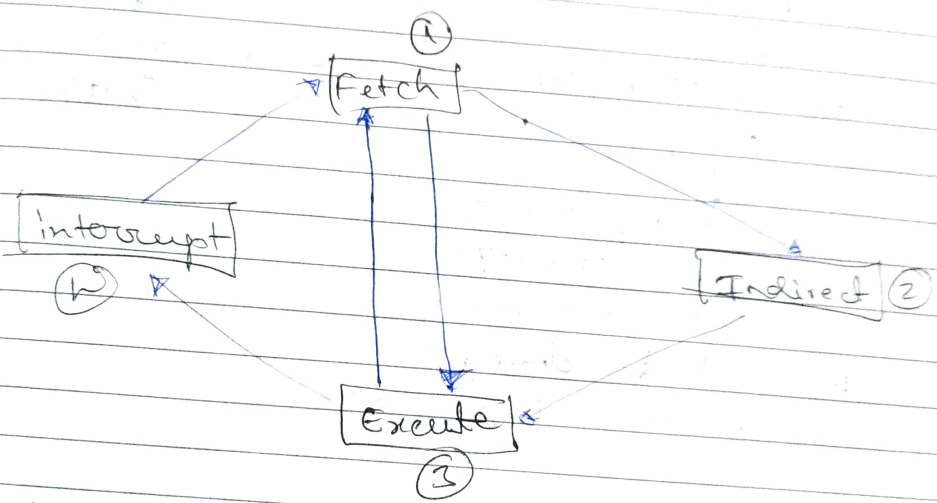
stage 4 [memory Access]

stage 5. [Destination register] } → Both no action for branching instructions.

Add R3, R4, 25

1. mean add, r [IR], send memory
   IR ← memory data, PC ← [PC] + 1
2. Decode instr, RA ← [IR], RB ← [PC]
3. RZ ← [RA] + [RB]
4. RY ← [RZ]
5. R3 ← [RY]

chpt 14th & 20th stallings 2nd
chpt 5th Zaki



000 or 03 or out of order. ↳ parallelism

ICC - instruct^n cycle code

complex instruct^n set computer CISC
Reduced " " " RISC

MFC - memory function complete → control signal

→ 178 pg. no

8086 → pentium. → x86 example

CISC → variable length instr.

RISC-I to RISC-V → RISC → fix length instr. → no. of instr.

↳ hardwired architecture implementation can't be altered.

RISC → less instr.

CISC

RISC

IF ID IE M WB

5 steps

more registers are used so 5 steps are enough.

maybe more than 5 steps.

It cant be changed as hardware implementat° is done.

* microprogram controller → CISC uses!!

2 steps → sequencing. (1)
→ execution (2)

static linking → compile time
dynamic linking → Run time

* William Stalling PPT

→ organisation on fig: 5.22

Page No.
Date

CISC

As less instr. are need, less program space

x86

variable length → may not fit in single word.

ADD R_1, R_2

may take more than 5 steps.

can be altered.

microprogram controller.

RISC

→ complement. RISC-I to RISC-V

fixed length instr. stored in word (32 bit).

ADD R_3, R_1, R_2

fixed 5 step execution.

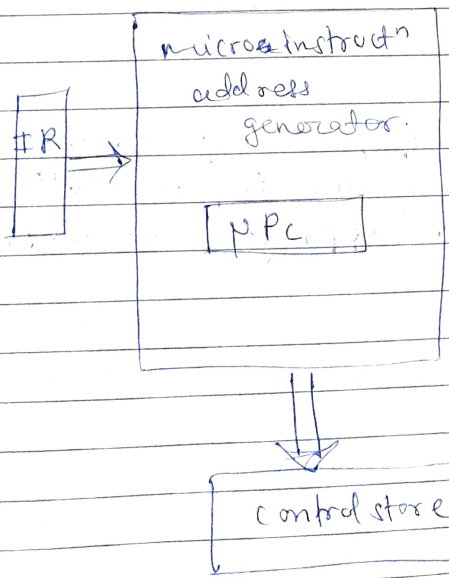
faster, but

can't be changed

hardware implementation

* Microprogram Controller – 184 pg. no. 5.7.2

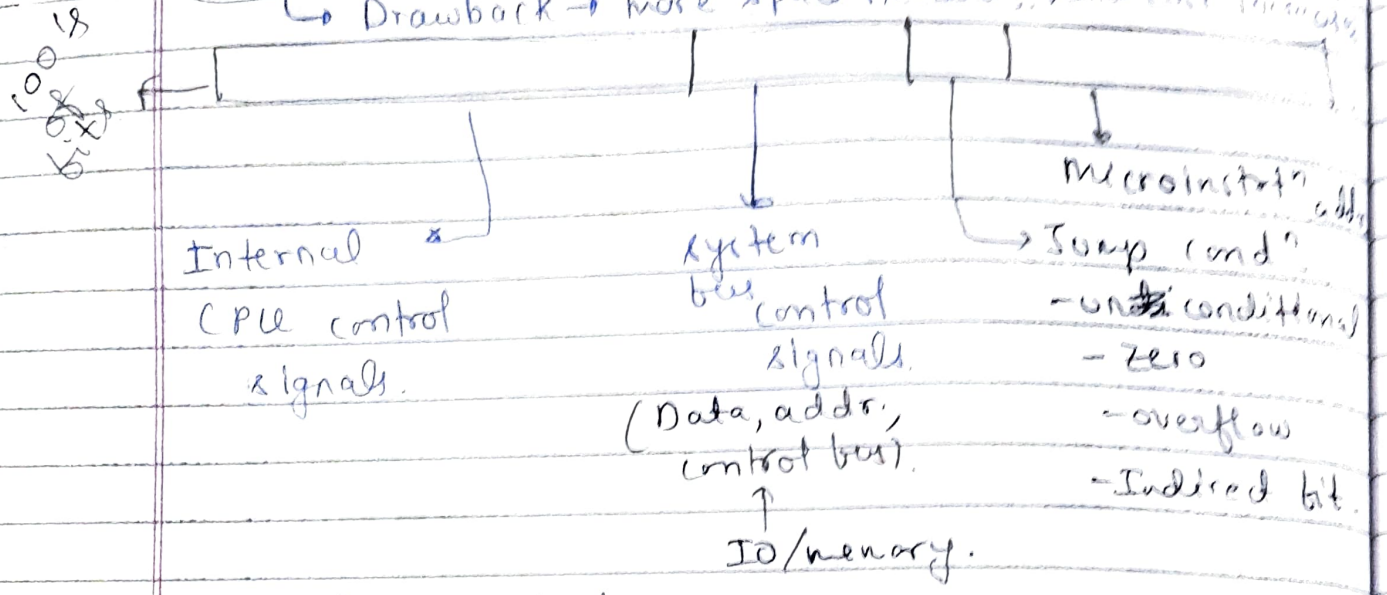
PC → μPC.



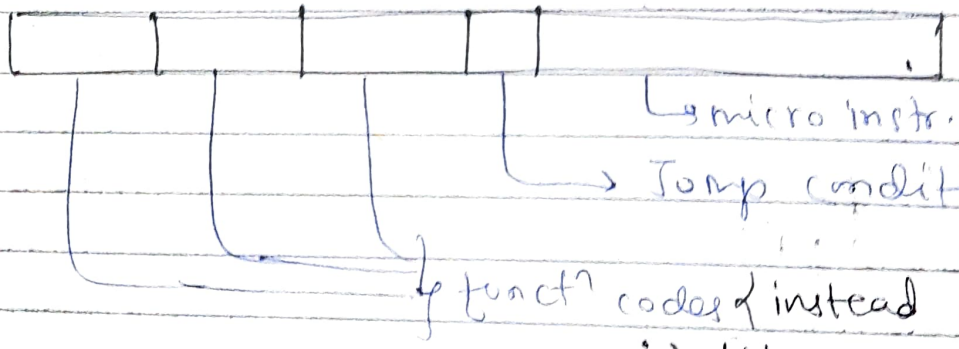
2 types of microinstructions

Horl. microInstructions.

↳ Drawback → more space needed !! hence cost increases



vertical microinstr.



- 2 ways of generating control signals
  - → Hardware implementat^n
  - → microprogrammed controller

Control signal generat^n

i) Hardware (RISC)

ii) Microproggram controller (CISC) → It has control memory i.e. separately from main memory.

on chip

* Control memory is usually implemented using FLASH ROM as it is writable yet non volatile

Page No.
Data

- On the pentium 4, machine instructions are converted into a RISC-like format, most of which are executed without the use of microprogramming.
- Microprogram or firmware is ~~somewhere~~ midway between hardware & software.

→ It is easier to design firmware than hardware but it is more difficult to write a firmware program than a software program.

coz 2 memories ~~etc~~ microprogrammed control has to deal d main memory & control memory
↳ approach is slow

control memory on ~~off~~-chip!!

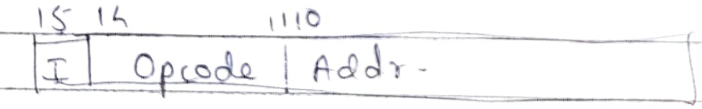
7 bit addr
control memory = 2^7 = 128 bytes

pg: 222, 223

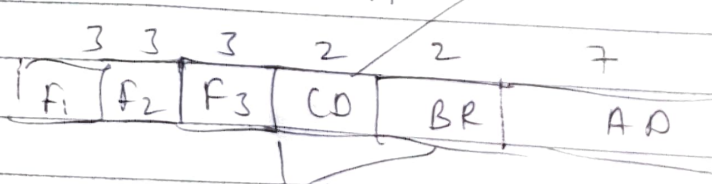
first 64 i.e. 0-63 reserved!!
used by or occupied by 16 instr.

64, 65, 66 are used for 3 microinstructions fetch routine needs 3 μinstructions

* Computer instr. format



ADD	0000
BRANCH	0001
STORE	0010
EXCHANGE	0011



condit^n for branching

micro instruction code format

branch field

SBR → subroutine.

Page No.
Date

F1

000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2

	NOP
	SUB
	OR
	AND
	READ
	ACTDR
	INCDR
	PCTDR

F3

		NOP
		XOR
		COM
		SHL
		INCPC
	PC ← AR	ARTPC
111	Reserved	
CD		
00		
01		
10		
11	AC = 0 → nothing gone to AC, gone to memory	

Morris Mano — unit - 7 chpts

↳ sir will give!!

BR	
00	JMP
01	CALL
10	RET
11	MAP

→ makes all 7 bits of AD to 0!! Again starts from new, next instruction

64, 65, 66.

```
FETCH:  ORG 64
        PCTAR
        READ, INCPC
        DRTAR
```

u	JMP	NEXT
u	JMP	NEXT.
u	MAP	

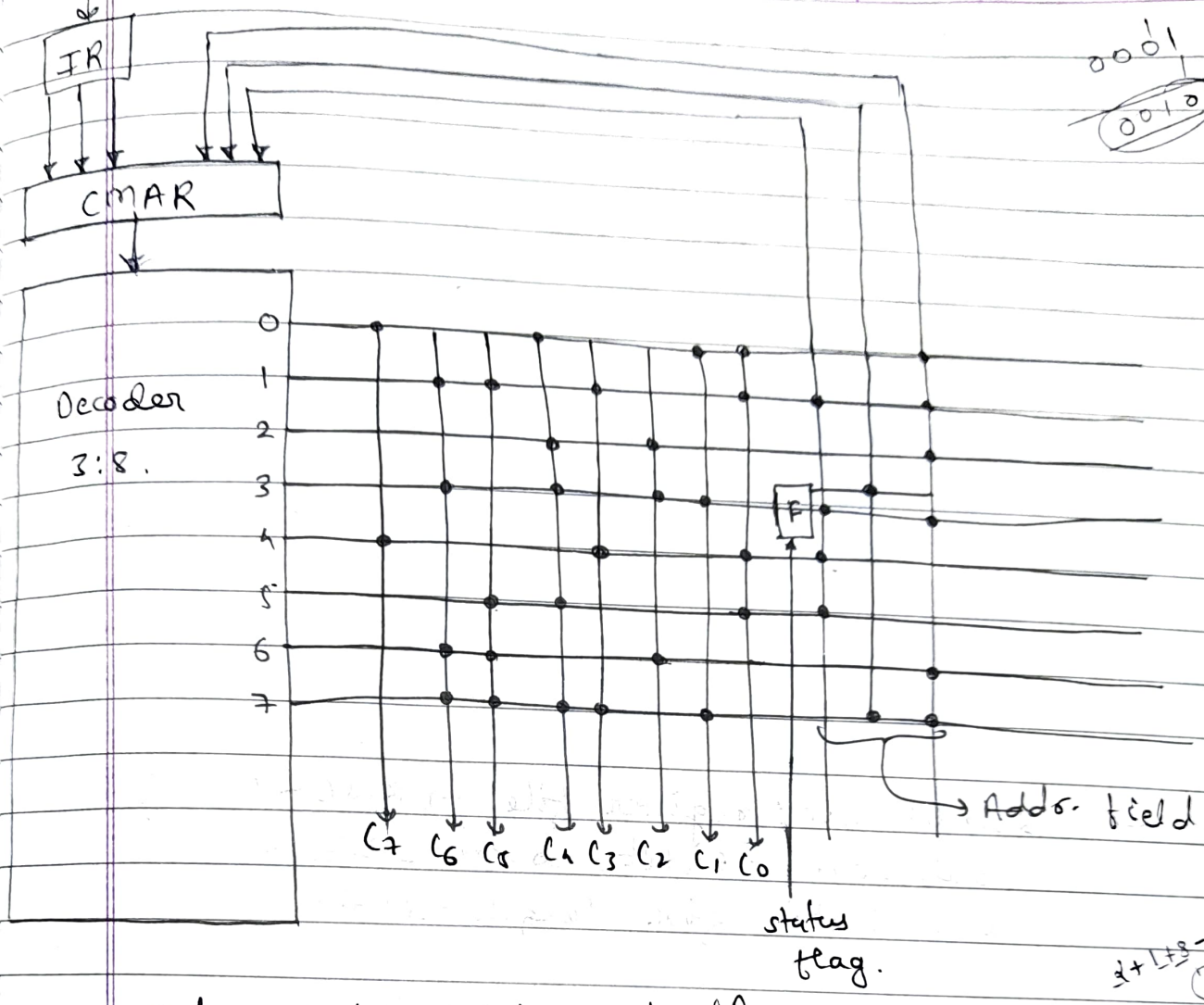
67, 68.

```
INDRCT:  READ
         DRTAR
```

u	JMP	NEXT
u	RET	

↳ returning the addr.

* Sign extension?



each row is a microinstruct^.

* Overflow rule - if 2 no.'s with the same sign (both +ve or both -ve) are added, then overflow occurs if and only if the result has the opp. sign.

* Subtraction rule - To subtract one no. (subtrahend) from another (minuend), take the 2's complement (negation) of the subtrahend and add it to the minuend.

RISC-V manual

PIC-18 → 4000 registers
ARM → 37 registers

Page No.
Date

CISC	RISC
– Complex instruction set computer	– Reduced instruction set computer
– flexible instr. format	– Rigid instr. format
– more addressing modes	– less addr. modes
– memory based	– register based (load, store)
– less registers	– more registers
– poor pipelining	– excellent pipelining
– variable execution time	– fixed execution time
– easier to program	– more complex
– microprogrammed CU	– hardware CU
– 8085, 8086, µp	– µcontrollers, ARM, PIC

→ 32 × 64 bit register file → RISC-V
doubleword = 64 bit, word = 32 bit.
each instr. fix length – 32 bit

- **R-format** (Register – Register)

Add R_7, R_5, R_3 $\quad R_7 \leftarrow R_5 + R_3$

32 bit →

funct^7	rs2	rs1	funct^3	rd	opcode
7bit	5bit	5bit	3bit	5bit.	7bit

add X9, X20, X21 $\quad X9 \leftarrow X20 + X21$

0	21	20	0	9
0000000	10101	10100	000	01001

opcode 51

0110011

1 word = 4 bytes = 32 bits

R, I, S, U, SB, UJ → beq, bge

S → sw, sb

U → lui

R → add, xor, mul

I → addi, lw, jalr, slli

	12 bit	5 bit	3 bit	5 bit	7 bit
I	imm[11:0]	rs1	funct^3	rd	opcode
S	imm[11:5] (7 bit)	rs2 (5 bit)	rs1	funct^3	imm[4:0] opcode
SB	funct^6	immed (6 bit)	rs1	funct^3	rd	opcode	shift4
U					
UJ					

beq rs1, rs2, L1

if (rs1 == rs2) branch to instr. labelled L1.

bne

↓

not equal!

+185, +122

1 0 1 1 1 0 0 1 — (ve) — 185

1 1 1 1 0 1 0 (+ve) — 122

1 0 1 0 0 1 1

−128 + 32 + 16 + 8 + 1

32
16
9
57

128
57
71

−71

185
128
57
32
25
16
9
8
1

122
64
58
32
26
16
10

-2^{n-1} to $+2^{n-1}-1$ — 2nd unit pattern

Page No.
Date

1) 6 + 13 = 19
2) −6 + 13 = 7
3) 6 − 13 = −7
4) −6 − 13 = −19

11010001

② 11001100 − 00101110
11001100 + 00010010 = 011011110

③ 1100011 − 1101000 = 11011011
1100011 + 0011000 = 1101101

① 111000 − 110011
111000 + 001101 = 1000101

④ 11110000111 − 11001111001 = 1100

① 00000110 + 00001101 = 00010011 = +19

② 00000110 + 00001101... 1 0000011 = +7

001100
1
001101

0011
1
010

③ 00000110 + 11110011 = 11111001 = −7

④ 11111010 + 11110011 = 1 11101101 = −19 (carry)

11110010

4+2+1
7

1+8+16+32
+64
−128

111000 → 111000
110011 + 001101 = 1000101

00010111
1
00011000

64
32
16
9
121

11110000 + 001100001100 = 1 0010000111000

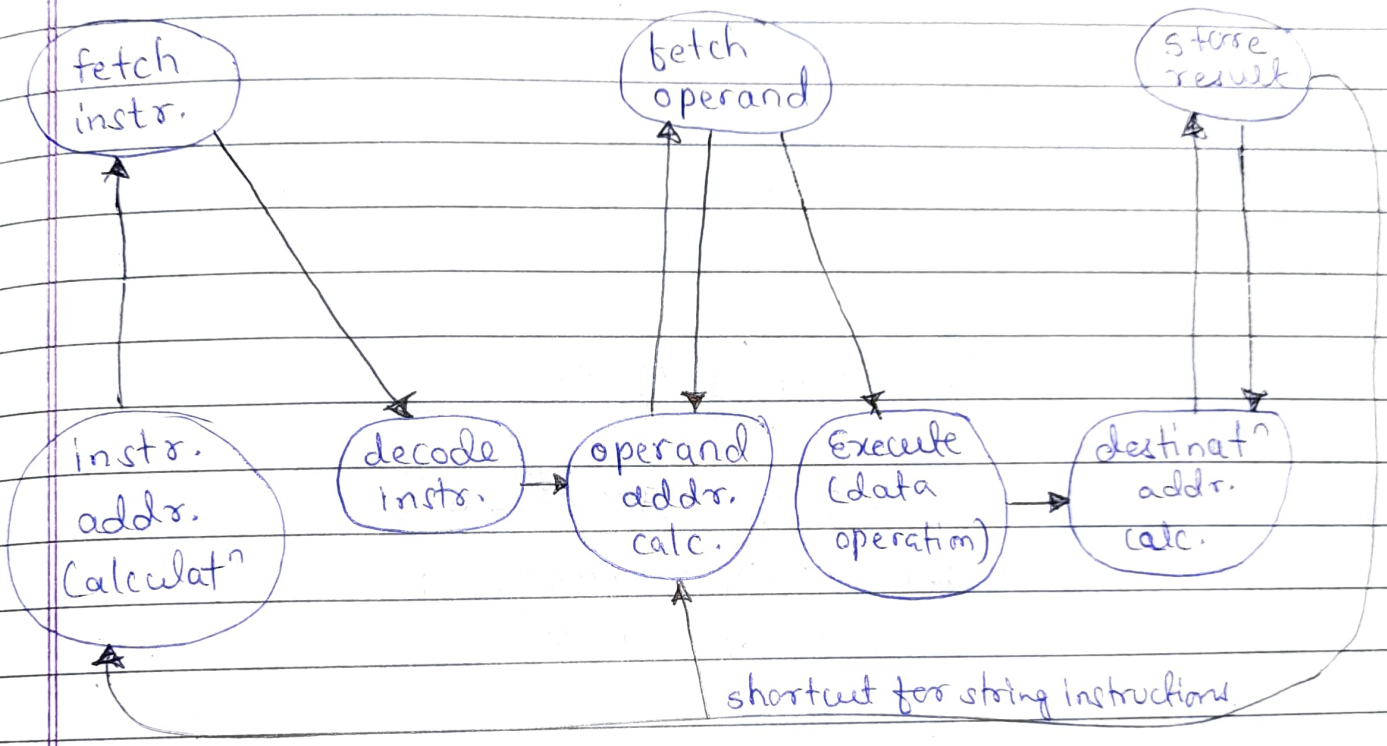
1+4+8+32
+64
96 − 13 = ...
128 − 128

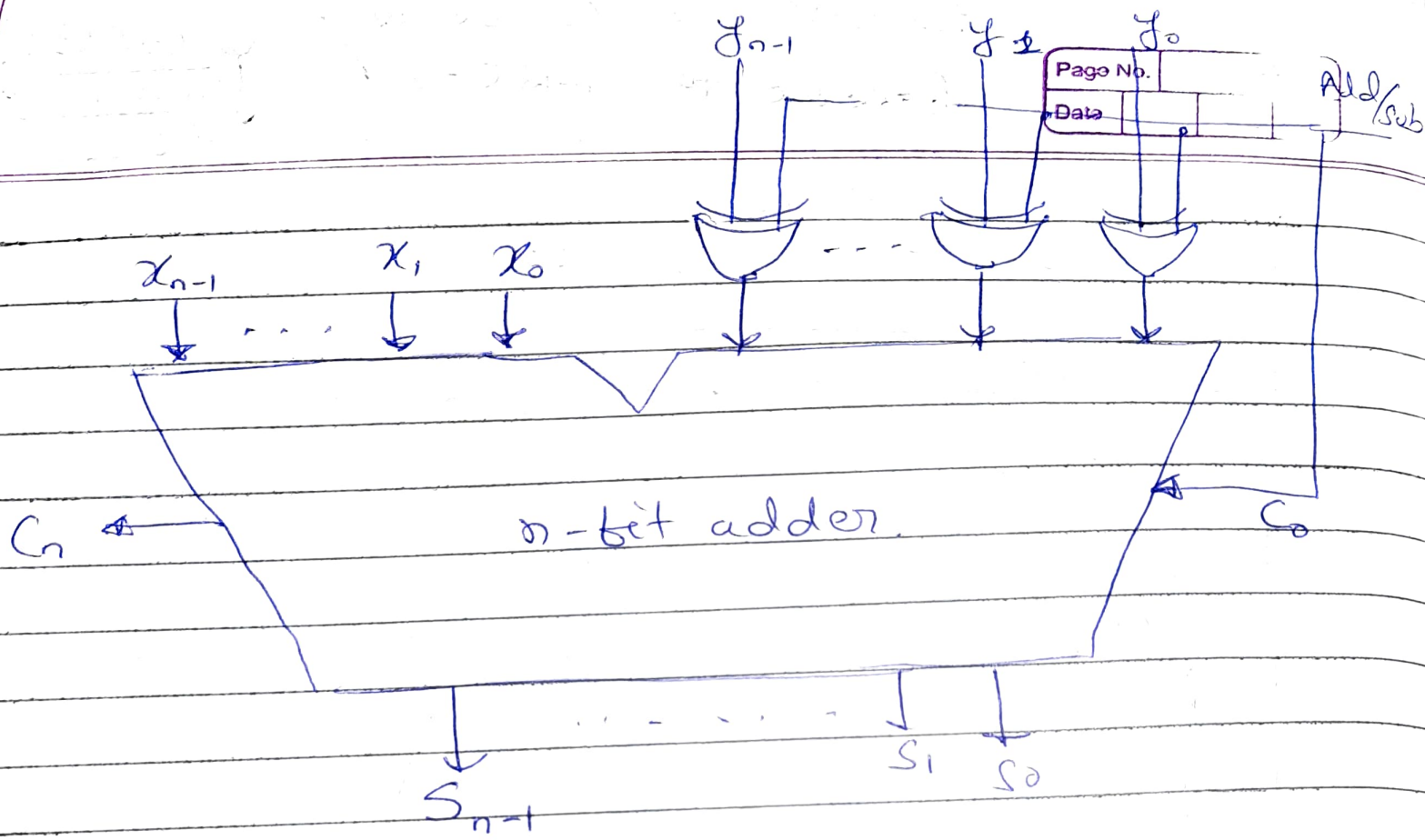
RISC-V to C code / C-code to RISC-V

Page No.
Date

- find the following differences using 2's complement arithmatic.

- Instruction cycle state transition diagram -





$$S_i = x_i \oplus y_i \oplus c_i$$

$$C_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$C_{i+1} = x_i y_i + (x_i + y_i) c_i$$

$$C_{i+1} = G_i + P_i c_i$$

$$G_i = x_i y_i$$

$$c_i = x_i + y_i$$

mm – 16 KB → 2^{14}

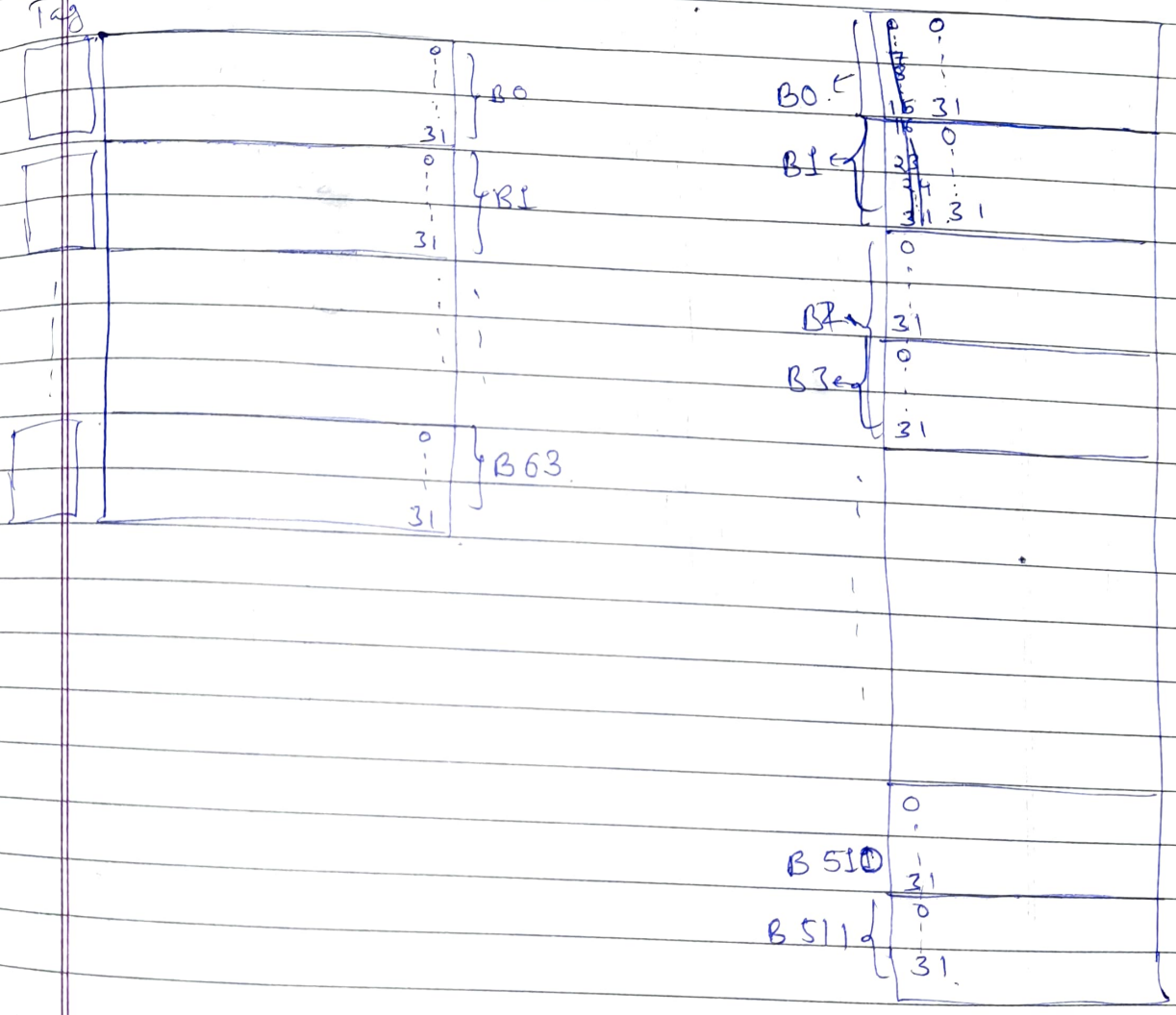
C.C – 2KB → 2^{11}

Block size – 32 bytes → 2^5.

Block C.C → $\frac{2^{11}}{2^5} = 2^6 = 64$

no. of blocks in m.m = $\frac{2^{14}}{2^5} = 2^9 = 512$.

each block consist of 32bytes



m.m address

tag	location/Block
(9)	(5 bit)

9+5 → 14 bit.

Tag → (0, 511)

Tag → 512 (2^9)

Tag → 9 bit.

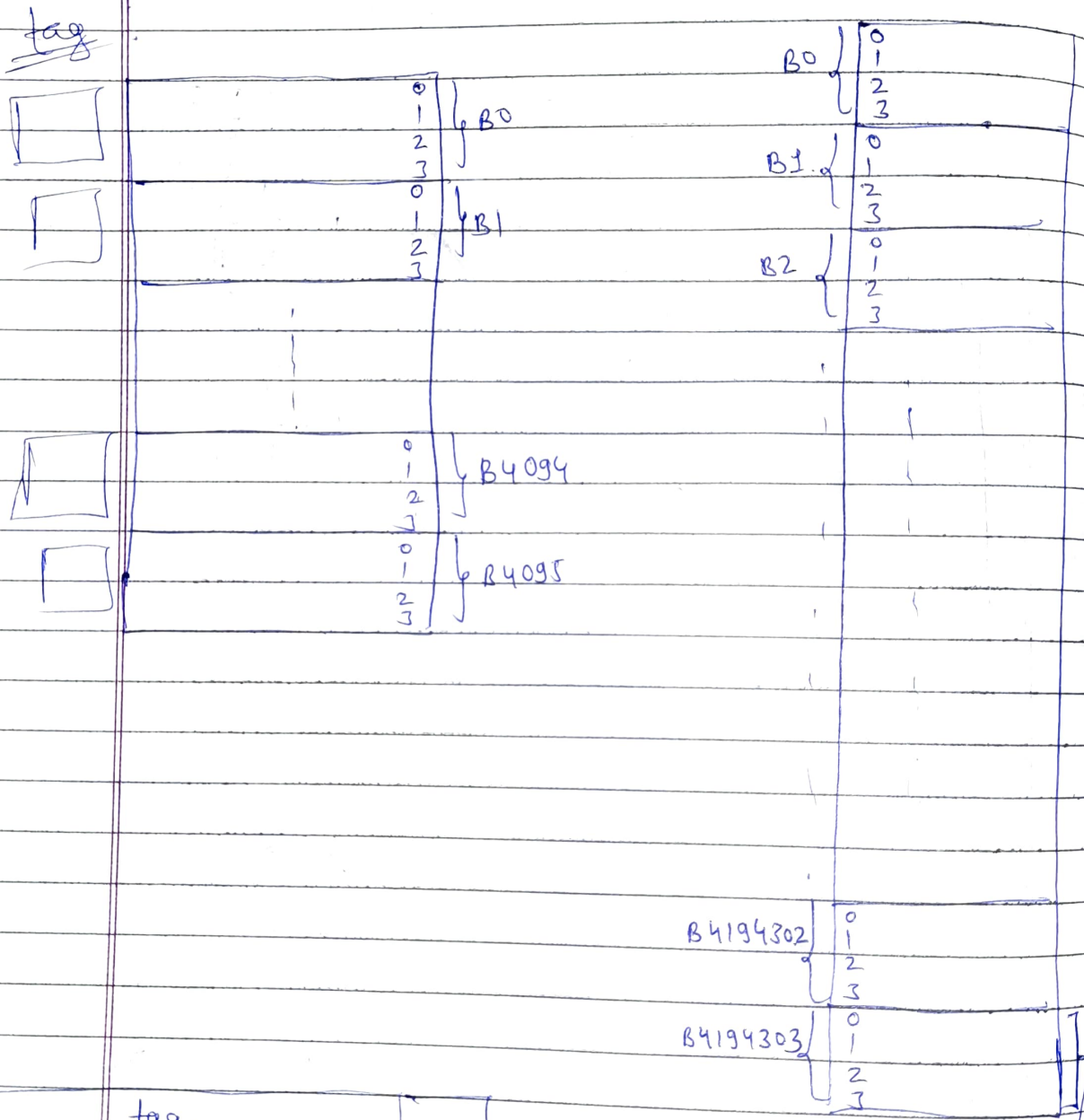
mm - 16MB → 2^{24}

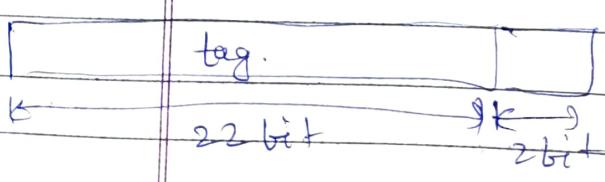
C.m - 16K → 2^{14}

block size - 32 bits → 4 bytes → 2^2

no. of blocks in mm = $2^{24}/2^2 = 2^{22} = 4194304$

no. of blocks in C-m = $2^{14}/2^2 = 2^{12} = 4096$.





4 byte.

tag → [0, 4194303)

tag → $4194304 - 2^{22}$

(Hex)

tag		word			
000000	0 0	00	13	57	92	46
000001	0 1	00				
			33	33	33	33
			11	22	33	44
3FFFFF	1 1	00	24	68	24	68

n mod K

0 mod 4 → 0
1 mod 4 → 1
2 mod 4 → 2
3 mod 4 → 3
4 mod 4 → 0
5 mod 4 → 1

Direct

m.m → 128 bytes → 2^7

block size → 32 bit → 4 bytes → 2^2.

C.m → 16 bytes → 2^4 (1)

C.m block size → $\frac{2^4}{2^2} = 2^2$ (4)

m.m block = $\frac{2^7}{2^2} = 2^5$ → 0, 31

L_0	B_0	B_4	B_8	B_{12}	B_{16}	B_{20}	B_{24}	B_{28}
L_1	B_1	B_5	B_9	B_{13}	B_{17}	B_{21}	B_{25}	W_{29}
L_2	B_2	B_6	B_{10}	B_{14}	B_{18}	B_{22}	B_{26}	B_{30}
L_3	B_3	B_7	B_{11}	B_{15}	B_{19}	B_{23}	B_{27}	B_{11}

	W_0	W_1	W_2	W_3	B_0
	W_4	W_5	W_6	W_7	B_1
	W_{124}	W_{125}	W_{126}	W_{127}	B_{31}

128 bytes

7 bit: | tag (3 bits) | line (2 bits) | offset (2 bit) |

32 total blk = 8 tag size

4 cacheline

tag → 2^3 → 3 bits.

line → 4 cacheline → 2^2 → 2 bits

offset → 4 bytes in a block → 2^2 → 2 bits.

L_0	1 block	4 words
L_1	1 block	4 words
L_2	1 block	4 words
L_3	1 block	4 words
		16 words

$= \frac{16 \text{ bytes}}{4 \text{ bytes}}$

$= 4$

Page No.
Date

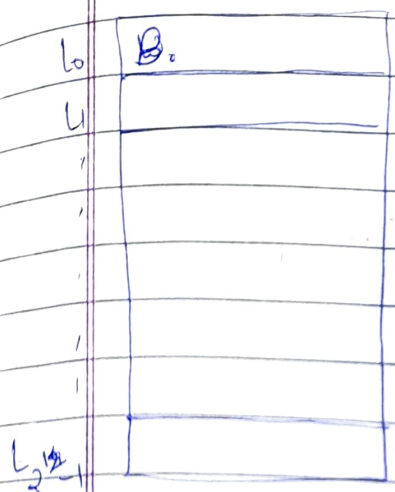
m.m → 16 MB → 2^{24}

C.M → 2^{14}

block size → 32 bits → 4 bytes → 2^2.

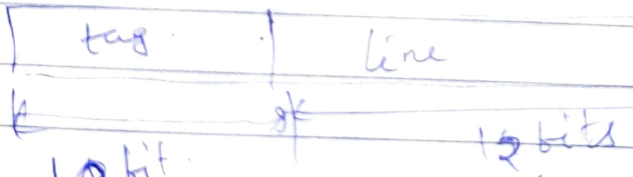
m.m block no.'s → $\frac{2^{24}}{2^2} \rightarrow 2^{22}$.

c.m block no.'s → $\frac{2^{14}}{2^2} \rightarrow 2^{12}$.



tagsize = $\frac{2^{22}}{2^{14}} = 2^8$

8bits 14bits



→ 4 words

→ 2^2 → 2bits.

C.M → 2^{14}

Lines → $\frac{2^{14}}{2^2} = 2^{12}$

$= 2^{22}$

Page No.
Date

Direct.

main memory = 16MB = 2^{24}

cache memory = 16KB = 2^{14}

block size = 32 bits = 4 bytes = 2^2

no. of blocks in main memory = $\frac{2^{24}}{2^2} = 2^{22}$

cache lines = no. of blocks in cache memory = $\frac{2^{14}}{2^2} = 2^{12}$.

tag	line	offset
24 − (12+2) = 10.	12 bit → total cache line = 2^{12}.	2 bit. (4 bytes in a block)

10 bit	12 bit	2 bit

8 bit	14 bit	2 bit	?