CAOS
Test 1

Mansi Uniyal.
19EE10039

Q1.	Clock rate	CPI Class A	Class B	Class C	Class D.
P1	1.5 GHz	1	2	3	4
P2	2.0 GHz	2	3	4	5

10^6 instructions : 10% Class A,
20% Class B,
50% Class C,
20% Class D.

(a) no. of clock cycles for execution by Processors P1 and P2.

(P1) → instructions × CPI

$= 10^6 \left[\frac{10\times1 + 20\times2 + 50\times3 + 20\times4}{100} \right] = 2.8\times10^6$ clock cycles

(P2)

$= 10^6 \left[\frac{10\times2 + 20\times3 + 50\times4 + 20\times5}{100} \right] = 3.8\times10^6$ clock cycles

(b) time required by P_1 and P_2 for execution.

time required = $\frac{\text{clock cycles}}{\text{clock frequency}}$.

(P1) → $\frac{2.8\times10^6}{1.5\times10^9} = 1.8667\times10^{-3}\,s = 1.8667$ ms

(P2) → $\frac{3.8\times10^6}{2\times10^9} = 1.9\times10^{-3}\,s = 1.9$ ms.

(c) global CPI for P_1 and P_2.

$$CPI = \frac{\text{total clock cycles}}{\text{instructions}}$$

$P_1 \rightarrow \frac{2.8 \times 10^6}{10^6} = 2.8$

$P_2 \rightarrow \frac{3.8 \times 10^6}{10^6} = 3.8$

d) which implementation is faster? indicate speed up factor of one with other.

CPU time P_1 < CPU time P_2.

P_1 is faster than P_2.

$$\text{Speed up factor} = \frac{T_{P_2}}{T_{P_1}} = \frac{1.9}{1.8667} = 1.018$$

Q.2. Write MIPS assembly program.

base address of A[n] → $s0
B[n] → $s1
C[n] → $s2
} registers.

$s3 → loop index.
loop limit → n. → $s4.

```
for (i=0; i<n; i++) {
  if (A[i] == B[i])
     C[i] = 0;
  else if (A[i] < B[i])
     C[i] = -1;
  else C[i] = 1; }.
```

```
addi $s5, $zero, 1
addi $s6, $zero, -1
addi, $s3, $zero, $zero      #initialize i=1
loop:
   slt $t0, $s3, $s4         #t0=0 if (i>=n)
   beq $t0, $zero, exit      #go to exit if (i>=n)
   sll $t1, $s3, 2           #$t1=(i*4)
   add $t2, $s0, $t1         #$t2=A+(i*4)
   add $t3, $s1, $t1         #$t3=B+(i*4)
   lw $t4, 0($t2)            #$t4=A[i]
   lw $t5, 0($t3)            #$t5=B[i]
   add $t2, $s2, $t1         #$t2=C+(i*4)
   bne $t4, $t5, else1       #go to else1 if (t4!=t5)
   sw $zero, 0($t2)          #C[i]=0
   j end                     #jump to end
else1:
   slt $t0, $t4, $t5         #t0=0 if (t4<t5)
   beq $t0, $zero, else2     #go to else2 if (A[i]>=B[i])
   sw $s6, 0($t2)            #C[i]=-1
   j end                     #jump to end
else2:
   sw $s5, 0($t2)            #C[i]=1
end:
   addi $s3, $s3, 1          #i=i+1
   j loop                    #jump to loop
exit.                        #exit
```

__/__/____

Q3. reconstruct C function from MIPS code. describe what the function does.

```
mipsfun:
        addi $t0, $a2, 1
loop:
        slt $t3, $a1, $t0
        addi $t4, $0, 1
        beq $t3, $t4, exit
        add $t1, $t0, $t0
        add $t1, $t1, $t1
        add $t1, $t1, $a0
        lw $t2, 0($t1)
        addi $t1, $t1, -4
        sw $t2, 0($t1)
        addi $t0, $t0, 1
        j loop
exit:
        jr $ra.
```

It left shifts the elements of an array at a0 from index a2 to index a1-1

C code

```
void mipsfun(int A0[], int a1, int a2) {
for (int i=a2; i<a1; i++) {
    A0[i] = A0[i+1]; } return; }
```

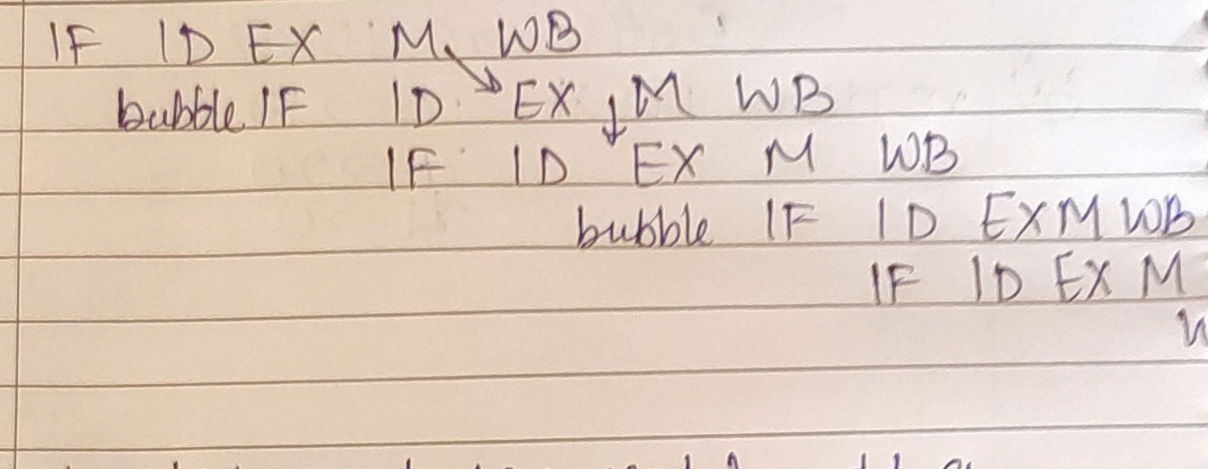
__/__/____

Q4. perform 7÷6 by first version of division hardware. with all iterations and sequence of steps clearly.

I	Step	Q	D	R.
0	Initial values	0000	0110 0000	0100 0111
1	rem=rem-div	0000	0110 0000	1111 0111
	2b. +div & LL 0,	0000	0110 0000	0100 1101
	0_0=0	0000	0011 0000	0100 1101
	3. Shift div right			
2.	rem=rem-div	0000	0011 0000	0001 1101
	2a. LL 0, 0_0=1	0001	0011 0000	0001 1101
	3.	0001	0001 1000	0001 1101
3.	rem=rem-div	0001	0001 1000	0000 0101
	2a	0011	0001 1000	0000 0101
	3	0011	0000 1100	0000 0101
4	rem=rem-div	0011	0000 1100	1111 1001
	2b	0110	0000 1100	0000 0101
	3	0110	0000 0110	0000 0101
5	rem=rem-div	0110	0000 0110	1111 1111
	2b	1100	0000 0110	0000 0101
	3	1100	0000 0011	0000 0101

__/__/____

Q5.



1→2 load: hazard, we need to add a bubble because of S2 which is loaded in 1 has to be used again in EX stage in 2.
This can be done with forwarding 1 bubble

2→3 S3 obtained after Ex of 2nd stage is needed before EX of 3rd stage.
This can solved by forwarding

3→4 Branch reset is obtained at end of EX only after which If of 4 is forwarded

4→5 Similar to 2→3 here. S5 is needed in 5 and calculated in 5.

total delay 3 time slots.

Execution time.

Q6. (i) single cycle approach

given,

$t_{IF} = 300ps$

$t_{ID} = 400ps$

$t_{EX} = 350ps$

$t_{MEM} = 500ps$

$t_{WB} = 100ps$

$t_c = (300 + 400 + 350 + 500 + 100) = 1650ps.$

$f_{clock} = 1/t_c = 6.060606 \times 10^8 Hz.$

(ii) 5 stage pipeline without forwarding.

$t_c = max(t_{IF}, t_{ID}, t_{EX}, t_{MEM}, t_{WB})$

$= 500ps.$

$f_{clock} = \frac{1}{t_c} = \frac{1}{500ps} = 2 \times 10^9 Hz$

(iii) 5 stage pipeline with data forwarding

$t_c = max(t_{ID}, t_{IF}, t_{EX}, t_{MEM}, t_{WB})$

$= 500ps$

total time $= 12 \times 500 = 6000ps$

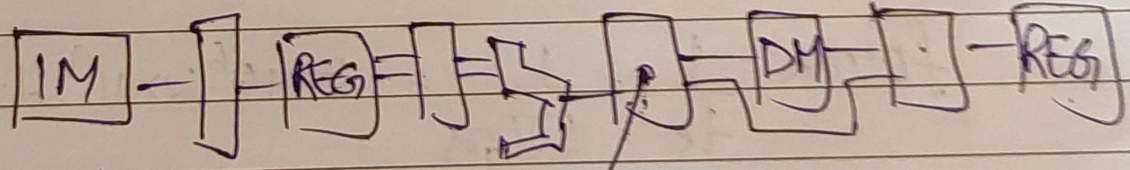
$f_{clock} = 1/500ps = 2 \times 10^9 Hz$

Q7. Data hazards occur when pipeline must be stalled because one step must wait for another to complete.
A MEM/WB merged may occur betⁿ an instruction in EX stage and instruction from 2 cycles again.
An example:

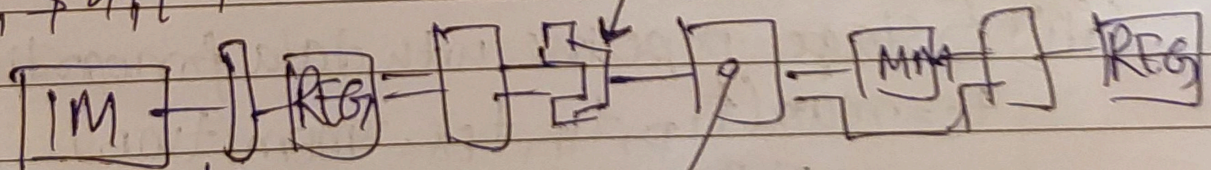
```
add $t1, $t2, $t3
add $t1, $t1, $t4
Sub $t5, $t5, $t1
```

register $t1 is written by both of previous instructions but always the most recent has to founded.

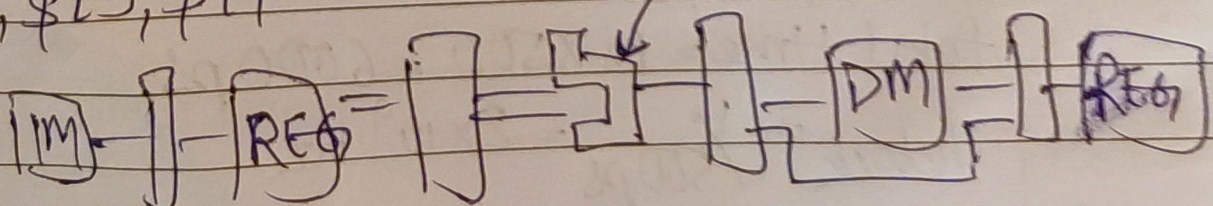
add $t1, $t2, $t3.



add $t1, $t1 $t4



sub $t5, $t5, $t1



∴ The hazard eqn becomes:

if |MEM| WB. RegWrite = 1 and
MEM|W.B. RegRd = |D|Ex Rg Rs and
|E|MEM. Reg Rd ≠ ID|Ex Reg RS or
Ex|MEM RegWrite
then forwd A = 1

if |MEM| WB. Regwrite = 1 and
MEM|WB. RegisterRD = ID|Ex. Reg
and |EX| MEM. Registered ≠ |D| Ex RegRt
or EX|MM. RegWrite
then forward B = 1.

Q8. Decade instruction

$r_s = 17$
$r_t = 20$
constant = 600

100011 | 10001
opcode | rs
10100 | 00000
rt | rd
01001 | 011000
shant | fnt

(a) left shift 2

600 << 2
= (600 x 4)
= 2400

O/p of sign extentd:

0000000 000 000 000 000 0000100101 1000

O/p of & 2 left shift

000 000 000 000 000 000 0010010 1100 00

(b) As it is lw instruction
ALU op = 00
ALU control = 0010
desired instruction is add

(c) PC_new = PCold + (600 << 2)
= PCold + 2400

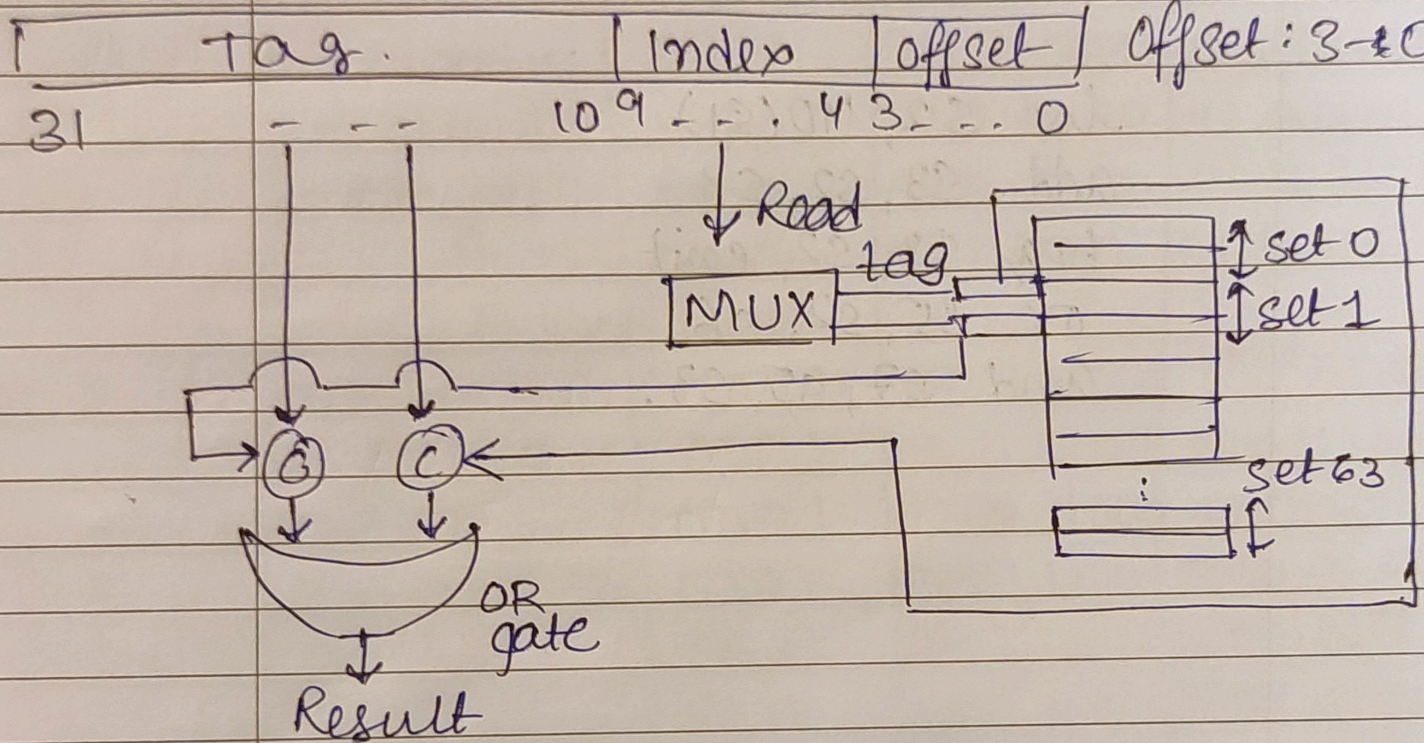
(e) ALU inputs → $rs, $rt
Both are 1000

(f) register unit rs → 17
rt → 20
rd → 0

Q9. 2way associative cache design with 32 bit address.

(a) Hardware implementation.

Tag: 31–10
Index: 9–4
Offset: 3–0



(b) Cache block size = 2^{offset} bytes
$= 2^4 = 16$ bytes
$= 4$ words

(c) no. of blocks = $2^{no.\ of\ sets} \times$ associations
$= 2^6 \times 2 = 2^7$
$= 128$

no. of entries = block size × no. of blocks
$= 4 \times 128$
$= 512$ words.

(d) Data storage supported by cache - 512 words
$= 2^9 \times 2^2$ bytes $= 2^{11}$
$= \underline{2 \text{ KiB}}$.

(e) total overhead involved
= (size of valid set + tag size) × no. of block
= (1 + 22) × 128 ~~bytes~~ bits
= 2944 bits
= 368 Bytes

(f) (i) Block address = $\frac{\text{reference address}}{\text{block size in bytes}}$.

no. of blocks in cache = $\frac{\text{line}}{\text{block ID}}$.

Address	line ID	Hit/Miss	Replace (Y/N)
160	10	Miss	N
200	12	Miss	N
172	10	Hit	N
1188	10	Miss	N
1224	12	Hit	N
2218	10	Hit	Y
2248	12	Miss	Y
4136	2	Miss	N
760	15	Miss	N
3268	12	Hit	N

(ii) no. of misses observed = 6

(iii) no. of block replacements observed = 2

(iv) Hit ratio = $\frac{\text{no. of hits}}{\text{total}} = \frac{4}{10} = 0.4$.