Birla Institute of Technology & Science – Pilani
Hyderabad Campus
1st Semester 2019-2020
Computer Architecture (CS F342) – Mid Sem Test (Regular) – KEY

Q1.

(a) Given the CPU of a computing system where 20% of its total execution time is spent on floating point multiplication operations and 40% of its total execution time is spent on floating point addition operations.

Scenario 1: Enhance the floating point multiplication operations by a factor of 10

$Speedup_{mul} = 1/[(1 - 0.2) + 0.2/10] = 1.22$

Scenario 2: Scenario 2: Enhance the floating point addition operations by a factor of 1.5

$Speedup_{add} = 1/[(1- 0.4) + 0.4/1.5] = 1.15$

Therefore, we prefer Scenario 1 as $Speedup_{mul}$ is higher.

(b) The ideal instruction sequence for P1 is the one composed entirely of instructions from class A (which have CPI of 1). So P1's peak performance is $(4 \times 10^9$ cycles/ second)/(1 cycle/instruction) = **4000 MIPS**.
Similarly, the ideal sequence for P2 contains only instructions from A, B, and C (which all have a CPI of 2).
So P2's peak performance is $(6\times10^9$cycles/second)/(2 cycles/instruction) = **3000 MIPS**.

(c)
Execution Time = Instruction Count * CPI * Clock Cycle Time
Execution TimeA = 100,000 * 1.9 * (1/1.8 GHz)
Execution TimeB = Instruction CountB * 2.6 * (1/2.4 GHz)
If the two execution times should be equal, then,
Instruction CountB = (2.4GHz×1.9×100000) / (1.8GHz×2.6) = 97436
Therefore the number of instructions the same program needs to have when compiled for machine B = **97436**

(d)
(i) the limit of speedup on 16 processors = 16/(1 + (16 – 1)*.04) = **10**
(ii) the maximum speedup = 1/0.04 = **25**

Q2. (a)
(i)
xor $rs,$rs,$rt
xor $rt,$rs,$rt
xor $rs,$rs,$rt

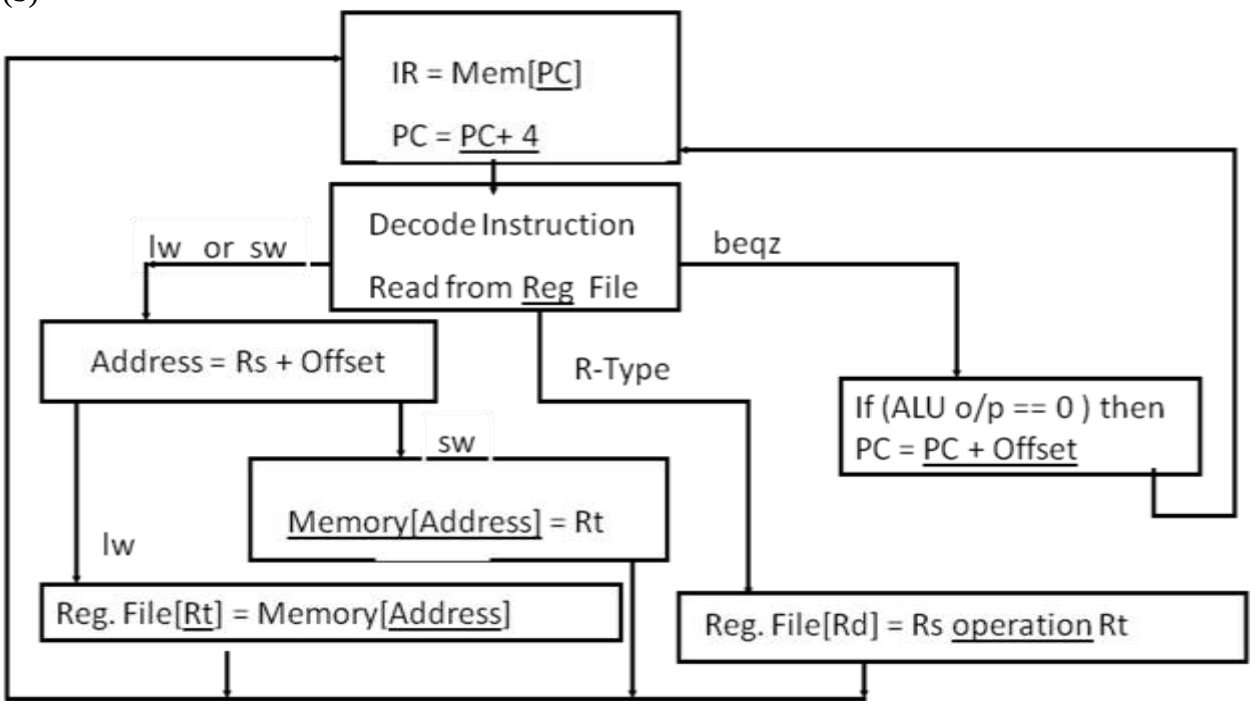
(ii) h/w and s/w implementation tradeoff
Software based implementation takes three clock cycles and hardware –based implementation takes one cycle.
Let R be the ratio of swaps in the code mix. Base CPI is given as 1
Avg time per instruction:
Let T be the clock period.

Software: R*3*T + (1 – R)*1*T = (2R + 1) * T

Hardware: T
Hardware implementation makes sense only if: T < = (2R + 1) * T
Given that the implementation of this instruction in hardware will increase the clock period of a single-instruction implementation by 8%,
Clock period = 1.08 * T

i.e. **if the swap instructions are greater than 4% of the instruction mix** (R > = 0.04), then a hardware implementation would be preferable.

(b)



(c)

Instruction	output
sll $t1, $t2, 8	$t1= **0xcd123400**
srl $t1, $t2, 4	$t1= **0x0abcd123**

Q3.

(a) Convert the number to binary.

$286.75_{10} = 100011110.11_2$

Now normalize the number: 1.0001111011×2^8

Thus mantissa is 0001111011

Biased exponent E = true exponent + bias = $8 + 127 = 135_{10}$ = E = 10000111_2

Thus the IEEE 754 representation of the given number is

0	1000 0111	0001 1110 1100 0000 0000 000
Sign bit (1)	Exponent (8) bits	Mantissa (23) bits

(b)

$x1 = 125.125_{10}$ and $x2 = 12.065_{10}$.

	S1	E1	M1
X1 =	0	10000101	11110100100000000000000

	S2	E2	M2
X2 =	0	10000010	10000010000000000000000

X3= X1 * X2

Find the sign bit by xor-ing sign bit of x1and x2

i.e. Sign bit of x3 = > (0 xor 0) => 0

Multiply the mantissa values including the "hidden one". The Resultant product of the 24 bits mantissas (M1 and M2) is 48 bits (2 bits are to the left of binary point)

$M3 = 1.M1 * 1.M2 = (10).111100101010100100000000000000000000000000000000$

$M3 = 1.011110010101010010000000000000000000000000000000 \times 2^1$

(Normalized binary)

Hidden "1"

Find exponent of the result. = E1 + E2 -bias + (normalized exponent from step 2) = $(10000101)_2 + (10000010)_2$ - bias +1 = 133 + 130 - 127 + 1 = 137.

Add the exponent value after normalization to the biased exponent obtained in step 2. i.e. 136+1 = 137 => exponent value.

The final result is

	S3	E3	M3
X3 =	0	10001001	01111001010101001000000

$x3 = 1509.3203125_{10}$

(c)

A = 9.75

B = 0.5625

Equivalent floating point binary words are

	S1	E1	M1
X1 =	0	10000010	00111000000000000000000

	S2	E2	M2
X2 =	0	01111110	00100000000000000000000

Step (1) Abs (A) > Abs (B)? Yes.
Step (2) Result of Initial exponent E1 = 10000010 = 130_{10}
Step (3) E1 - E2 = (10000010 - 01111110) => (130-126) = 4
Step (4) Shift the mantissa M2 by (E1-E2) so that the exponents are same for both numbers.

$$1.M2 = 1.00100000000000000000000$$
$$= 00001.00100000000000000000000$$
$$= 0.00010010000000000000000 \quad \text{(Aligned mantissa)}$$

Step (5) Check whether the sign bits of both are equal? Yes.
Step (6) now, add the mantissas

$$\begin{array}{rl} & 1.00111000000000000000000 \ (1.M1) \\ + & 0.00010010000000000000000 \ (\text{aligned M2}) \\ \hline & 1.01001010000000000000000 \ \ 1.M3 \end{array}$$

Step (7) the result is

	S3	E3	M3
X3 =	0	10000010	01001010000000000000000

The result in decimal is x3 = 10.3125

(d)

Value10	Rounded	Action performed and reason
7.8949999	7.89	(Less than half way)
7.8950001	7.90	(Greater than half way)
7.8950000	7.90	(Greater than half way)
7.8850000	7.88	(Half-way - round down so that the LSD is even)

Q4.

(a)

If any of the inputs is negative we need to take its 2's complement. For multiplier, now a 0 is added to its right. According to the Booth's encoding technique, for each pair from LSB to MSB, add 1 bit at a time.

i.e., 00 = 0, 01 = +1, 10 = -1, 11 = 0

Here, Multiplier Q = 0111 0111 1011 1101

Add 0 to rightmost position => 0111 0111 1011 11010

Now, apply Booth's code: (add 1 bit at a time, from LSB to MSB)

= 01, 11, 11, 10, 01, 11, 11, 11, 10, 01, 11, 11, 11, 10, 01, 10

= +1 0 0 -1 +1 0 0 0 -1 +1 0 0 0 -1 +1 -1

Thus the total number of additions/ subtractions = 8

(b)

n	M	A	Q	Operation
4	00011	00000	1011	initialize
	00011	00001	011_	shift left AQ
	00011	11110	011_	A = A - M
	00011	00001	0110	Q[0] = 0 and restore A
3	00011	00010	110_	shift left AQ
	00011	11111	110_	A = A - M
	00011	00010	1100	Q[0] = 0
2	00011	00101	100_	Shift left AQ
	00011	00010	100_	A = A - M
	00011	00010	1001	Q[0] = 1
1	00011	00101	001_	shift left AQ
	00011	00010	001_	A = A - M
	00011	00010	0011	Q[0] = 1

Therefore quotient = register Q contents = 3 and

Remainder = register A contents = 2.

Q5.
(a)Given

	Fetch	Decode	Execute	Mem	Write back
(i)	300ps	400ps	350ps	550ps	100ps
(ii)	200ps	150ps	100ps	190ps	140ps

(i) non-pipelined:
Clock cycle time = 1700ps
Latency = 1700ps
Throughput = 1/1700 instructions per ps
(ii) non-pipelined:
Clock cycle time = 780ps
Latency = 780ps
Throughput = 1/780 instructions per ps

(i) Pipelined:
If we have to split one of the pipeline stages into 2 equal halves, we would want to choose the longest stage to split in half. The new cycle time becomes the originally 2nd longest stage length. Now, we need to calculate the new latency and new throughput correspondingly. Also, now the number of stages would be 6 instead of 5.
Clock cycle Time = 400 + 20 = 420 ps
Latency = 6 * 420 = 2520 ps
Throughput = 1/420 inst/ps
(ii) Pipelined:
Clock cycle Time = 190 + 20 = 210 ps
Latency = 6 * 210 = 1260 ps
Throughput = 1/210 inst/ps

(b)
(i) A non-pipelined processor with six execution stages has latencies 50ns,50ns,60ns,60ns, 50ns and 50ns, to compute the instruction latency in this machine and the time taken to execute 100 instructions.

Instruction latency = 50+50+60+60+50+50= 320 ns
Time to execute 100 instructions = 100*320 = 32000 ns

(ii) If we introduce pipelining with an overhead of 5ns to each execution stage, to find the instruction latency in the pipelined machine and the time taken to execute 100 instructions.

The length of pipelined stage = max (lengths of non- pipelined stages) + overhead = 60 + 5 = 65 ns
Instruction latency = 65 ns
Time to execute 100 instructions = 65*6*1 + 65*1*99 = 390 + 6435 = 6825 ns

(iii) the speedup obtained for 100 instructions
Average instruction time non pipelined = 320 ns
Average instruction time pipelined = 65 ns
Speedup for 100 instructions = 32000 / 6825 = 4.69

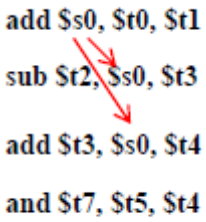
Q6.

(a)

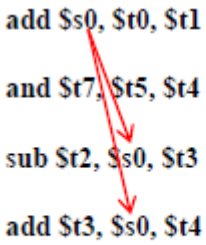
Slno.	Scenario	Type	Slno.	Scenario	Type	Slno.	Scenario	Type
1	I: R2 <- R1 + R3 J: R4 <- R2 + R3	RAW	2	I: R2 <- R1 + R3 J: R3 <- R4 + R5	WAR	3	I: R2 <- R1 + R3 J: R2 <- R4 + R5	WAW

(b)

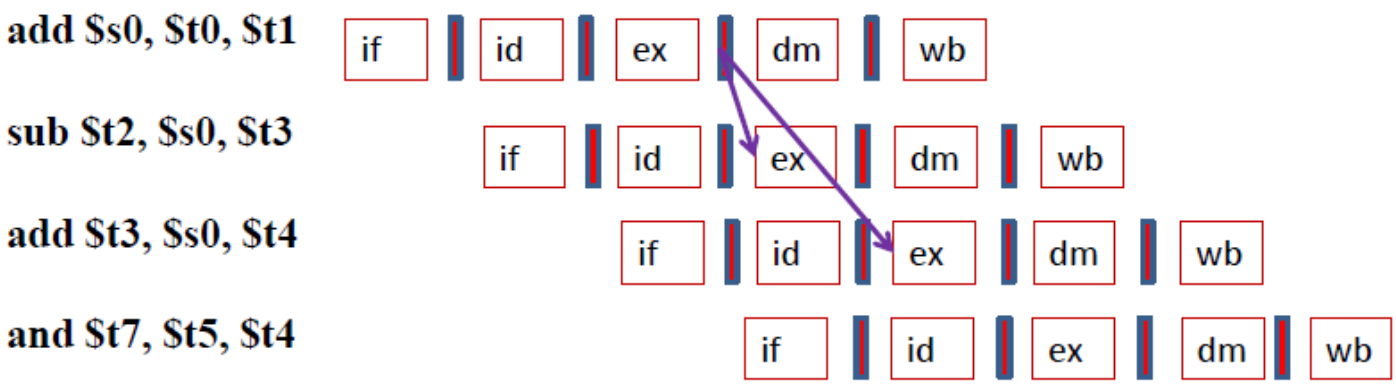
(i) Identify the data hazards in the code.



(ii) Apply instruction reordering as a solution



(iii) Use forwarding as a solution and show the forwarding paths.



Q7.

(a)

(i)0x000AE430	(ii)0x00014432	(iii)0x000B0737	(iv)0x0E0D8844

Since MIPS ISA is word aligned and the word size is 4 bytes, aligned addresses will be divisible by word size. In this case, (i) and (iv) are hexa addesses divisible by 4 and hence word aligned whereas addresses (ii) and (iii) are not divisible by 4 and hence not aligned.

(b)

1. The 32-bit address in $10 is: **0x00400000**
2. The offset is sign-extended to 32 bits as: **0x00000060**
3. The memory address computed as the 32-bit sum of the above is: **0x00400060**
4. Main memory is asked for data from the address: **0x00400060**
5. After a one machine cycle delay the data reaches $8.