Jan. 10, 2015

COMPUTER ENGINEERING DEPARTMENT

ICS 233

COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Final Exam

First Semester (141)

Time: 8:00-10:30 AM

Student Name : KEY____

Student ID. : ____

Question	Max Points	Score
Q1	20	
Q2	20	
Q3	26	
Q4	20	
Total	86	

Dr. Aiman El-Maleh
Dr. Samer Arafat

[20 Points]

(Q1)

(i) **(10 points)**

Suppose that you have a processor that executes a certain program with the following characteristics: 50% of the execution time is taken by multiply, 20% of the execution time is taken by divide, and the remaining 30% of the execution time is taken by other instructions.

We want the program to run faster. Suppose that we can improve the multiply to run 2 times faster and the divide to run 4 times faster. However, due to hardware cost, only one improvement can be made.

1. Which improvement should be done (multiply or divide?), assuming that the hardware cost is identical. Justify your answer. **(4 points)**
2. Given that the program executes on the processor without improvement in 10s, what will be the execution time of the program with the chosen improvement?
3. Suppose we can now improve both the multiply and divide instructions. What is the speedup of the improved machine relative to the original machine? **(4 points)**

We will use Amdahl's Law as follows:
ExT after improvement = ExT affected by improvement + unaffected ExT

(1)

Suppose the original execution time = 100

Improving the multiply by a factor of 2:
Execution time with improved multiply = 50/2 + 20 + 30 = 75

OR Speedup due to multiply = 1/(0.5/2 + 0.5) = 1/0.75 = 1.33

Improving the divide by a factor of 4:
Execution time with improved divide = 50 + 20/4 + 30 = 85

OR Speedup due to divide = 1/(0.2/4 + 0.8) = 1/0.85 = 1.176

If only one improvement should be made then it is better to improve the multiply.

(2)

Speedup of improving the multiply = 100 / 75 = 1.33
Execution time for a 10s program execution = 10 / 1.33 = 7.5 s

- another solution using Amdahl's Law -
Execution time for a 10s program execution = 5/2 + 2 + 3 = 7.5 s

(3)

Execution time with improved multiply and divide = 50/2 + 20/4 + 30 = 60

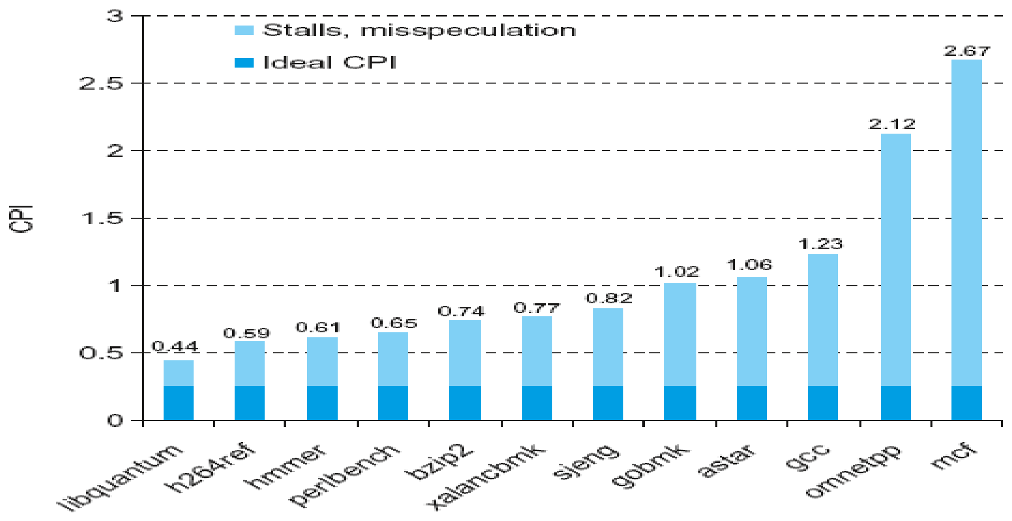
Speedup of improving both multiply and divide = 100 / 60 = 1.67

OR Speedup of improving both multiply and divide = 1/(0.5/2 + 0.2/4 + 0.3) = 1/0.6=1.67

(ii) **(3 points)**

A certain Intel i7 processor has been tested using SPEC2013, which uses 12 benchmark software programs. The program names are listed on the X-axis of the following figure, below, and the corresponding CPI was computed and shown on the Y-axis. The numbers on top of each bar show the CPI that includes stalls and misprediction (called missspeculation in the graph). The ideal CPI = 0.25 for all benchmark software programs.

What is the processor CPI, according to SPEC2013?



Processor's CPI = average of benchmarks' CPIs with stalls and missspeculation = 1.06

(iii) **(7 points)**

Given a certain processor that has the following operation times for processor components: instruction and data memories: 150 ps, ALU and adders: 140 ps, decode and register file access (read or write): 100 ps, which of the following would be faster and by how much, a single-cycle implementation for all instructions, or a multi-cycle implementation optimized for every class of instructions.

Assume the following instruction mix: 50% ALU, 10% Loads, 10% stores, 15% branches and 15% jumps. Ignore the delays in PC, mux, extender, and wires.

Class	IM	RR	ALU	DM	RW	Total (ps)
ALU	150	100	140		100	490
Load	150	100	140	150	100	640
Store	150	100	140	150		540
Branch	150	100	140			390
Jump	150	100				250

For a single-cycle implementation, the clock cycle is determined by longest delay, which is load instruction, and that is 640 ps.

For a multi-cycle implementation, the clock cycle is determined by longest delay at any step, which is the IM or DM steps, and that is 150 pm.

Next, we show the CPI for each instruction class:

Instruction	# cycles	Instruction	# cycles
ALU & Store	4	Branch	3
Load	5	Jump	2

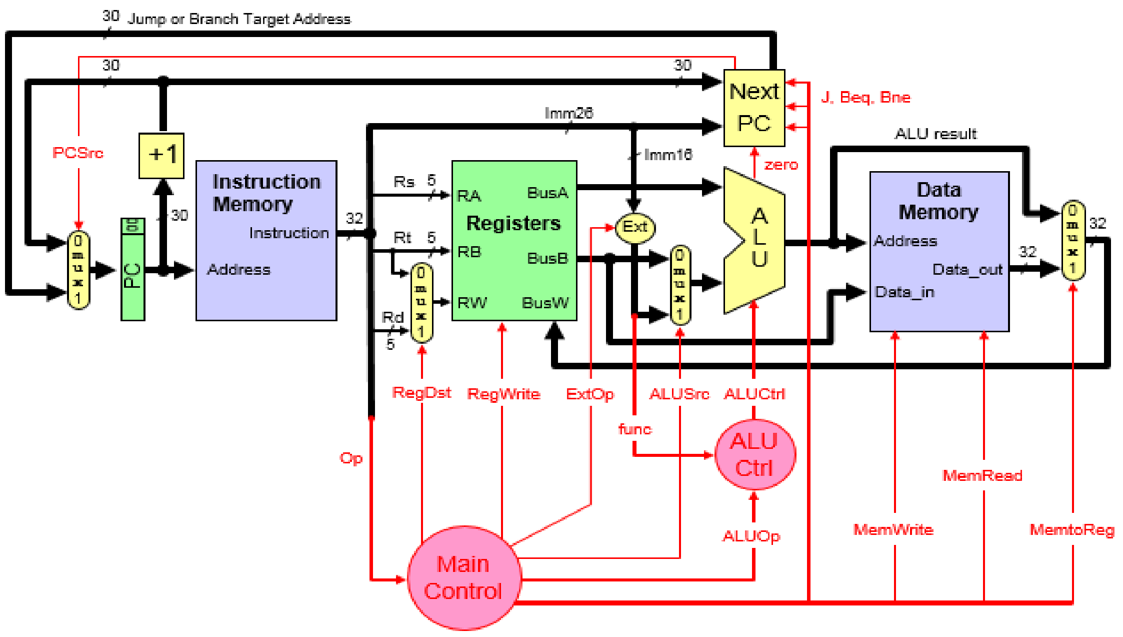
Average CPI = 0.5x4 + 0.1x5 + 0.1x4 + 0.15x3 + 0.15x2 = 3.65

Multiple cycle is faster by a factor = 640/(3.65x150) = 1.17

(Q2) **[20 Points]**

(i) **(13 points)**

Consider the single-cycle CPU design given below.



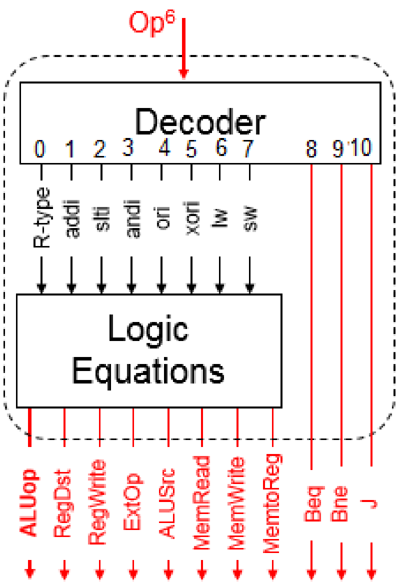
You are required to complete the single cycle processor main control unit design.

1. Complete the main control signal values in the table below **(6 points)**.

Op	**Reg Dst**	**Reg Write**	**Ext Op**	**ALU Src**	**ALU Op**	**Beq**	**Bne**	**J**	**Mem Read**	**Mem Write**	**Mem toReg**
R-type	1=Rd	1	x	0=BusB	Rtype	0	0	0	0	0	0
addi	0=Rt	1	1=sign	1=Imm	ADD	0	0	0	0	0	0
slti	0 =Rt	1	1=sign	1=Imm	SLT	0	0	0	0	0	0
andi	0 =Rt	1	0=zer	1=Imm	AND	0	0	0	0	0	0
ori	0 =Rt	1	0=zer	1=Imm	OR	0	0	0	0	0	0
xori	0 =Rt	1	0=zer	1=Imm	XOR	0	0	0	0	0	0
lw	0 =Rt	1	1=sign	1=Imm	ADD	0	0	0	1	0	1
sw	X	0	1=sign	1=Imm	ADD	0	0	0	0	1	x
beq	X	0	x	0=BusB	SUB	1	0	0	0	0	x
bne	X	0	x	0=BusB	SUB	0	1	0	0	0	x
j	X	0	x	x	x	0	0	1	0	0	x

2. Derive the logic design of the control unit for all the control signals given in the table above, based on the given instructions, except ALUOp. Assume that the opcode of these instructions is a 6-bit opcode such that the opcode for R-type instructions is 0, the opcode for addi is 1, the opcode for slti is 2, and so on for the rest of the instructions. **(7 points)**

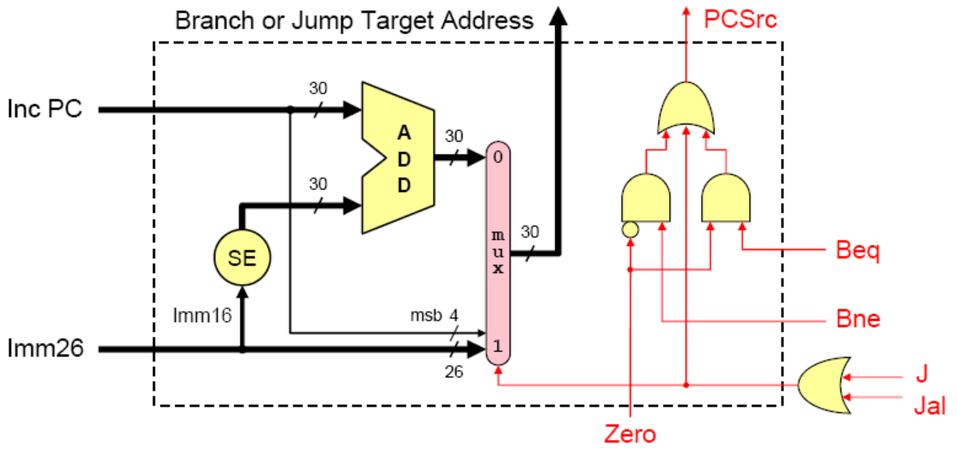
RegDst	<=	R-type
RegWrite	<=	$\overline{(\text{sw} + \text{beq} + \text{bne} + \text{j})}$
ExtOp	<=	$\overline{(\text{andi} + \text{ori} + \text{xori})}$
ALUSrc	<=	$\overline{(\text{R-type} + \text{beq} + \text{bne})}$
MemRead	<=	lw
MemWrite	<=	sw
MemtoReg	<=	lw

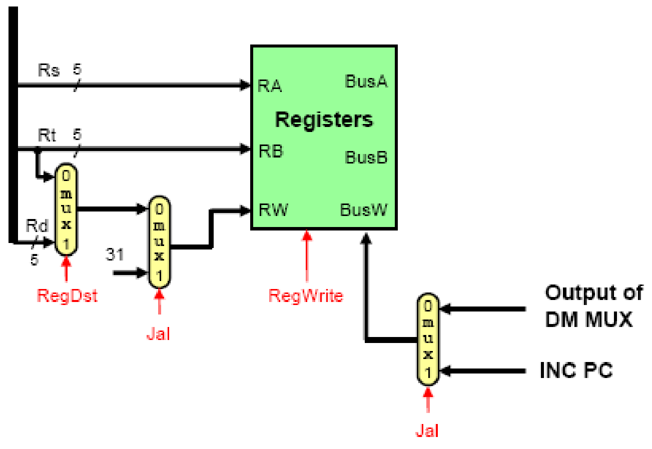


(ii) **(7 points)**

Modify the given single cycle datapath so that it implements the Jump and link instruction, jal. Draw only the blocks (resources) that must be modified. If the NextPC block needs to be changed, show its internal design details.

Instruction	Meaning	Format	
jal label	\$31=PC+4, jump	$op^6 = 3$	imm^{26}





[26 Points]

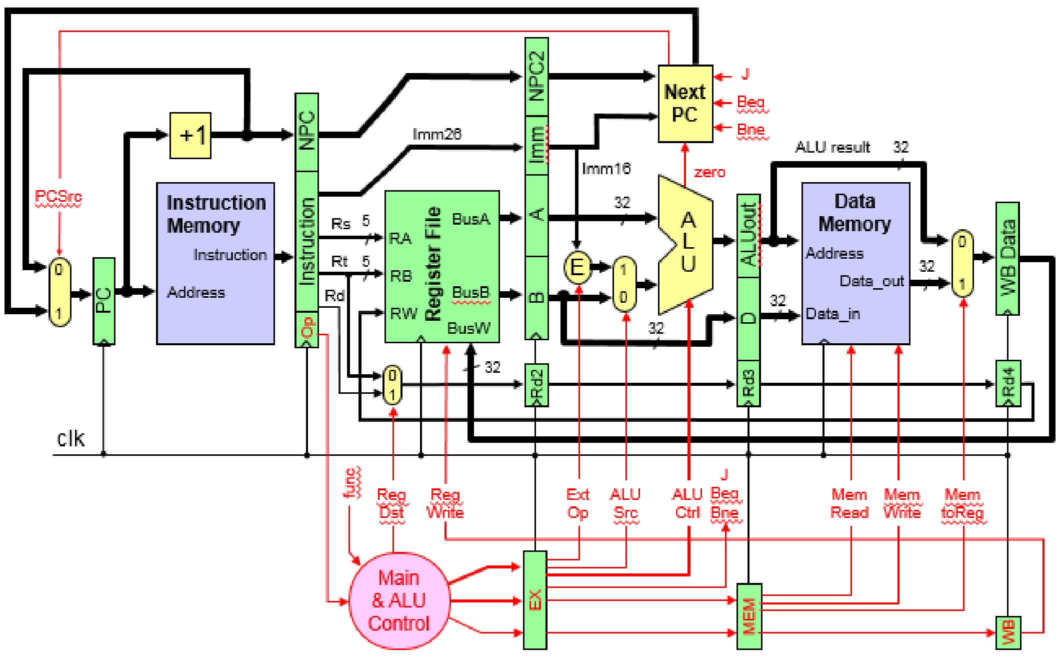
(Q3)

(i) Fill the blank in each of the following questions: **(7 Points)**

1. For a 20-stage pipeline, the maximum speedup that can be achieved over serial execution is 20.
2. In a pipelined CPU design, structural hazards may occur due to the attempt of the use of the same hardware resource by two different instructions during the same cycle.
3. In the MIPS 5-stage pipeline, data hazards that may occur are Read After Write (RAW) hazards.
4. Hazards due to jump and branch instructions are called control hazards.
5. In order to achieve a zero delay for a jump or a taken branch, we need to use Branch Target Buffer (BTB) in the IF stage.
6. Given the branch outcomes of a branch instruction as shown in the table below, the accuracy of prediction using a **2-bit predictor**, initialized to **weakly predict not taken** is 4/6=66.67%.

Branch Outcome	N	T	N	N	N	T
Branch Prediction	N	N	N	N	N	N

(ii) Consider the single-cycle CPU design given in Q2, show the necessary modifications in the data path and control unit to implement this CPU as a 5-stage Pipeline without considering any type of hazards. Label all the added parts clearly. **(7 Points)**



(iii) Consider the following MIPS assembly language code:

```
I1:     ADDI $a1, $0, 10
I2:     LW $t2, 0($t5)
I3:     ADD $t3, $t2, $t2
I4:     SW $t3, 0($t5)
I5:     ADDI $a1, $a1, -1
I6:     ADDI $t5, $t5, 4
I7:     BNE $a1, $0, I2
```

a. Complete the following table showing the timing of the above code on a 5-stage pipeline (IF, ID, EX, MEM, WB) that supports **forwarding**. Draw an arrow showing forwarding between the stage that provides the data and the stage that receives the data. Show all stall cycles (draw an X in the box). Assume that the branch delay is 1 clock cycle. **(9 Points)**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1: ADDI	**IF**	**ID**	**EX**	-	**WB**													
I2: LW		**IF**	**ID**	**EX**	**M** $\downarrow$	**WB**												
I3: ADD			**IF**	**X**	**ID**	**EX** $\downarrow$	-	**WB**										
I4: SW					**IF**	**ID**	**EX**	**M**	-									
I5: ADDI						**IF**	**ID**	**EX**	- $\downarrow$	**WB**								
I6: ADDI							**IF**	**ID**	**EX**	-	**WB** $\downarrow$							
I7: BNE								**IF**	**ID**	-	-	-						
I2: LW									**X**	**IF**	**ID**	**EX**	**M** $\downarrow$	**WB**				
I3: ADD											**IF**	**X**	**ID**	**EX**	**M**	**WB**		

b. Assume that branch instructions are implemented using a 1-cycle delayed branch. Rearrange the given code to reduce the number of stall cycles. Justify your solution? **(3 Points)**

We rearrange the solution as follows:

```
I1:   ADDI $a1, $0, 10
I2:   LW $t2, 0($t5)
I3:   ADDI $a1, $a1, -1
I4:   ADD $t3, $t2, $t2
I5:   SW $t3, 0($t5)
I6:   BNE $a1, $0, I2
I7:   ADDI $t5, $t5, 4
```

This will eliminate the stall cycle after I2 as now I3 does not depend on I2. In addition, since we are using a delayed branch of 1 cycle, the instruction after the branch will always be executed. Thus, we fill this slot with the instruction ADDI $t5, $t5, 4. Thus, with this code rearrangement there will be no stall cycles.

[20 Points]

(Q4)

(i) Fill the blank in each of the following questions: **(7 Points)**

1. Using a larger block size in cache memory reduces compulsory misses while increases conflict misses.
2. Increasing cache size reduces capacity misses and conflict misses.
3. Small cache size is used for L1 caches to reduce the hit time.
4. Multi-level caches are used to reduce miss penalty.
5. Valid and Modified bits are required for write back policy.

(ii) Assume that you have a 32-bit address and a cache with **8K byte data size** (not including tag and valid bits).

a. Assuming that the cache is organized as **direct-mapped** with a **16-byte block size**, determine the number of bits in the offset, index and tag fields. **(3 Points)**

Offset = $\log_2$ (block size) = $\log_2 16 = 4$ bits
Index = $\log_2$ (# locations) = $\log_2 (8K/16) = \log_2 512 = 9$ bits
Tag= 32- (9+4) = 19 bits

b. Assuming that the cache is organized as **four-way set associative** with a **16-byte block size**, determine the number of bits in the offset, index and tag fields. **(3 Points)**

Offset = $\log_2$ (block size) = $\log_2 16 = 4$ bits
Index = $\log_2$ (# locations) = $\log_2 (8K/16*4) = \log_2 (128) = 7$ bits
Tag= 32- (7+4) = 21 bits

(iii) A processor runs at 2.5 GHz and has a CPI=1.6 for a perfect cache (i.e. without including the stall cycles due to cache misses). Assume that load and store instructions are 18% of the instructions. The processor has an I-cache with a 4% miss rate and a D-cache with 5% miss rate. The hit time is 1 clock cycle. Assume that the time required to transfer a block of data from the RAM to the cache, i.e. miss penalty, is 31 ns.

a. What is the number of stall cycles per instruction and the overall CPI? **(4 Points)**

Miss penalty in clock cycles = $31 * 10^{-9} * 2.5 * 10^{9} = 77.5$ => 78 cycles
Number of stall cycles per instruction = 0.04*78+0.18*0.05*78 = 3.822
Overall CPI = 1.6 + 3.822 = 5.422

b. What is the average memory access time (AMAT) in ns? **(3 Points)**
No of memory accesses = I + 0.18*I=I*(1+0.18)=1.18*I
Access Time = 1*I + 0.04*I*78 + 0.18*I + 0.18*I*0.05*78=
I*(1+0.18+(0.04+.18*.05)*78)=5.002*I clock cycles
AMAT = Access Time / No of memory accesses = 5.002*I/1.18*I=4.239 clock cycles = $4.239 * 1/(2.5 * 10^{9})$= 1.696 ns