King Fahd University of Petroleum and Minerals
College of Computer Science and Engineering
Computer Engineering Department

COE 301 COMPUTER ORGANIZATION
ICS 233: COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Term 151 (Fall 2015-2016)
Final Exam
Sunday Dec. 20, 2015
8:00-11:00 AM

Time: 180 minutes, Total Pages: 14

Name:_KEY______________________________ ID:___________________ Section: ________

Notes:

- Do not open the exam book until instructed
- Answer all questions
- All steps must be shown
- Any assumptions made must be clearly stated
- Mobile phones must be switched off

Question	Max Points	Score
Q1	15	
Q2	18	
Q3	13	
Q4	17	
Q5	12	
Total	75	

Dr. Aiman El-Maleh
Dr. Mayez Al-Muhammad

[15 Points]

(Q1)

(i) We wish to compare the performance of two different computers: M1 and M2. The following measurements have been made for running a program (Program 1) on these computers:

Program 1	M1	M2
#Instructions Executed	5×10^9	6×10^9
CPI	1.2	1.25
Clock Rate	3 GHZ	5 GHZ

a. Which computer is faster for running the program and by how much?

(3 points)

Execution time for Program 1 on M1 = $5 \times 10^9 \times 1.2 \times 1/(3 \times 10^9) = 2$ s
Execution time for Program 1 on M2 = $6 \times 10^9 \times 1.25 \times 1/(5 \times 10^9) = 1.5$ s
Computer M2 is faster for program 1 and it is faster by a factor=2/1.5=1.33

b. Suppose the execution time of a second program (Program 2) on both computers is given below.

Program 2	M1	M2
Execution Time	5.0 seconds	10.0 seconds

Suppose that program 1 must be executed 1600 times each hour. Any remaining time should be used to run program 2. Which computer is faster for this workload? Performance is measured here by the throughput of program 2.

(4 points)

Executing program 1 on M1 1600 times each hour will consume 1600x2=3200 seconds. Remaining time for running program 2 on M1= 3600-3200=400 seconds. Thus, program2 can be run in M1 400/5=80 times.

Executing program 1 on M2 1600 times each hour will consume 1600x1.5=2400 seconds. Remaining time for running program 2 on M1= 3600-2400=1200 seconds. Thus, program2 can be run in M2 1200/10=120 times. Thus, for this workload computer M2 is faster.

(ii) Consider two different implementations, M1 and M2, of the same instruction set. There are three classes of instructions (A, B, and C) in the instruction set. M1 has a clock rate of 6 GHz and M2 has a clock rate of 3 GHz. The CPI for each instruction class on M1 and M2 is given in the following table:

Class	CPI on M1	CPI on M2	C1 Usage	C2 Usage
A	2	1	40%	60%
B	3	2	40%	15%
C	5	2	20%	25%

The above table also contains a summary of the usage of instruction classes generate by two different compilers: C1 and C2. Assume that each compiler generates the same number of instructions for a given program. Which computer and compiler combination give the best performance? **(5 points)**

Exec. Time for M1 using C1 compiler = Ix(2x0.4+3x0.4+5x0.2) x1/(6×10^9) =0.5I ns
Exec. Time for M2 using C1 compiler = Ix(1x0.4+2x0.4+2x0.2) x1/(3×10^9) =0.53I ns
Exec. Time for M1 using C2 compiler = Ix(2x0.6+3x0.15+5x0.25) x1/(6×10^9)=0.48I ns
Exec. Time for M2 using C2 compiler = Ix(1x0.6+2x0.15+2x0.25) x1/(3×10^9) =0.47I ns

Machine M2 and Compiler C2 will give the best performance.

(iii) A benchmark program runs for 200 seconds. We want to improve the execution time of the benchmark by a factor of 2.5. We enhance the floating-point unit to make the floating-point instructions run 4 times faster. How much of the initial execution time would floating-point instructions have to account for to show an overall speedup of 2.5 on this benchmark?

(3 points)

Speedup = 1 / (f/s + (1-f)) => 2.5 = 1 / (f/4+(1-f)) => f/4 + 1-f = 1/2.5 => f + 4 – 4f = 1.6 => 3f = 2.4 => f =0.8

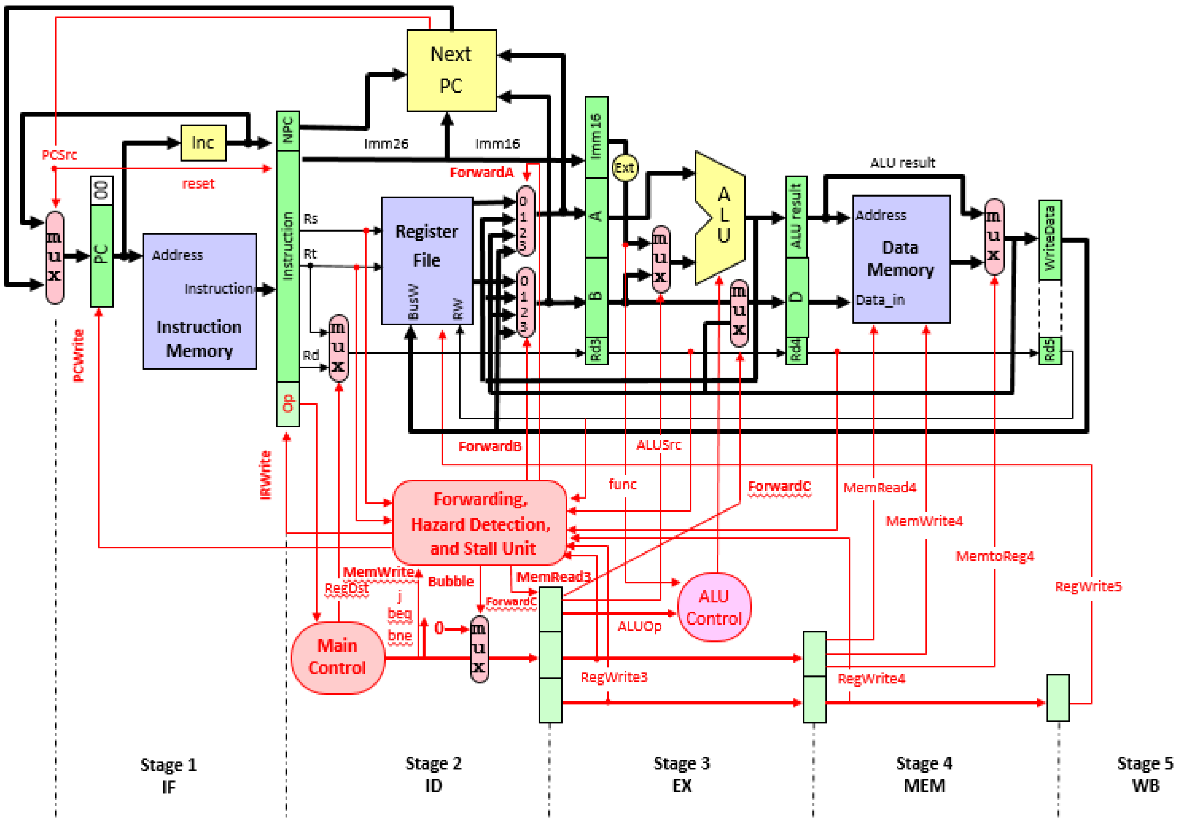
Thus, floating-point instructions must account for 80% of the initial execution time to show an overall speedup of 2.5 on this benchmark.

OR

80 = 200*f/4 + (1-f)*200 => 80 = 50 f + 200 – 200 f => 150 f = 120 => f =120/150 = 0.8

[18 Points]

(Q2) Consider the 5-stage pipelined CPU design given below:



(i) Show the control signals that will be used for stalling the pipeline due to data hazards along with their conditions. Add the necessary changes to the design, on the given diagram, to allow it to handle data hazards. **(5 points)**

if ((MemRead3 == 1) // Detect Load in EX stage
and (ForwardA==1 or ForwardB==1)) Stall // RAW Hazard

<u>OR</u>:

if ((MemRead3 == 1)
and (Rd3 $\neq$ 0) and ((Rs == Rd3) or (Rt == Rd3))) Stall

Stall means that the signals PCWrite=0 and IRWrite=0, which will freeze the content of PC and IR registers and bubble=1 which will introduce a bubble in stage 2 control register by setting the control signals to 0.

(ii) Consider the instruction sequence given below:

```
lw $t1, ($s0)
sw $t1, ($s1)
```

We can forward that data of **lw** instruction to the next **sw** instruction as required by the above example. However, such forwarding is not supported by the given 5-stage pipeline CPU design.

a. Show the required changes in the datapath and forwarding unit to support such forwarding.
b. Write the condition for generating the forwarding control signal. Identify the pipeline registers and control signals used by the **sw** and **lw** instructions when writing the condition.

(6 points)

We need a multiplexer at the B input in the EX/MEM register as shown in the diagram. The data loaded from the Data Memory should be fed back at the input of the multiplexer. A control signal, ForwardC, is needed to control the selection of this multiplexer. The Forwarding unit in the ID stage will generate the ForwardC signal and pipeline it, after detecting the dependency between a SW and a previous LW instruction.

ForwardC=0 means no forwarding,
ForwardC=1 means forward the load data from the MEM stage.

Condition for generating ForwardC:

If (MemRead3 == 1 and MemWrite == 1 and Rd3 ≠ 0 and Rt == Rd3)
ForwardC = 1
Else ForwardC = 0.

(iii) Consider the following MIPS assembly language code: **(7 Points)**

```
I1:     ADDI $s0, $0, 10
I2:     ADD  $s0, $s0, $s0
I3:     SLL  $s0, $s0, 4
I4:     LW   $s1, 4($s0)
I5:     ADDI $s2, $s1, -1
I6:     SW   $s2, 4($s0)
```

Complete the following table showing the timing of the above code on the 5-stage pipeline given in part (i) (IF, ID, EX, MEM, WB) assuming that it supports **forwarding** and **pipeline stall**. 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 to represent a stall cycle). Determine the number of clock cycles to execute this code.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1: ADDI	**IF**	**ID**	**EX**	-	**WB**										
I2: ADD		**IF**	**ID**	**EX**	-	**WB**									
I3: SLL			**IF**	**ID**	**EX**	-	**WB**								
I4: LW				**IF**	**ID**	**EX**	**M**	**WB**							
I5: ADDI					**IF**	**X**	**ID**	**EX**	-	**WB**					
I6: SW							**IF**	**ID**	**EX**	**M**	-				

The number of clock cycles to execute this code is 10.

[13 Points]

(Q3)

(i) Fill in the blanks in each of the following questions: **(8 points)**

1. The Direct Mapped Cache organization leads to a fast access time but lacks **flexibility** in block placement compared to fully associative mapping.

2. The Fully Associative Cache organization leads to a **slow** access time due to the multiplexer used but has a lot of **flexibility** in block placement.

3. Access time of a **Set Associative** Cache **varies** with the set size.

4. With the Write Through policy, every write to cache is propagated to DRAM, which makes DRAM data always **coherent** with the cached data. A Write Through cache may not use a **write allocate** policy.

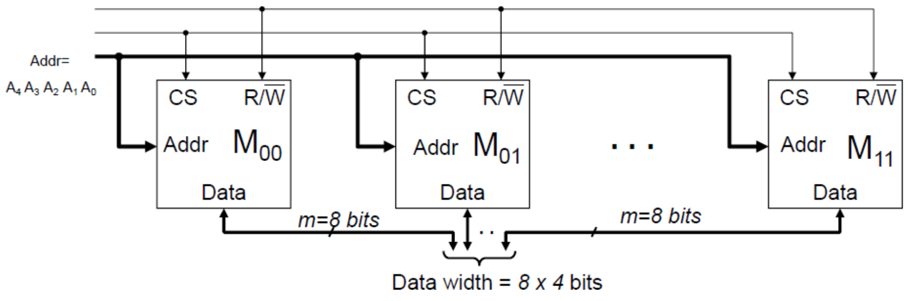
5. With the Write Back policy, multiple writes to a block accumulate in the cache, but the block will be written back to DRAM if it has been **modified**. A Write Back cache must use a **write allocate** policy.

6. In a 4-way set-associative cache with 64 Kbyte data capacity (i.e. not counting tag and other bits), and with a 64 byte block size, the number of bits used for the offset is **Six** and the number of bits for the index is **Eight**.

7. In a **FIFO** replacement policy, one counter is used per set to replace the **Oldest** block.

(ii) **(5 points)**

1. A memory system consists of memory modules M_{00} , M_{01} , M_{10} and M_{11} which are interconnected using the below drawing, where the address is 5-bit $A_4 A_3 A_2 A_1 A_0$.



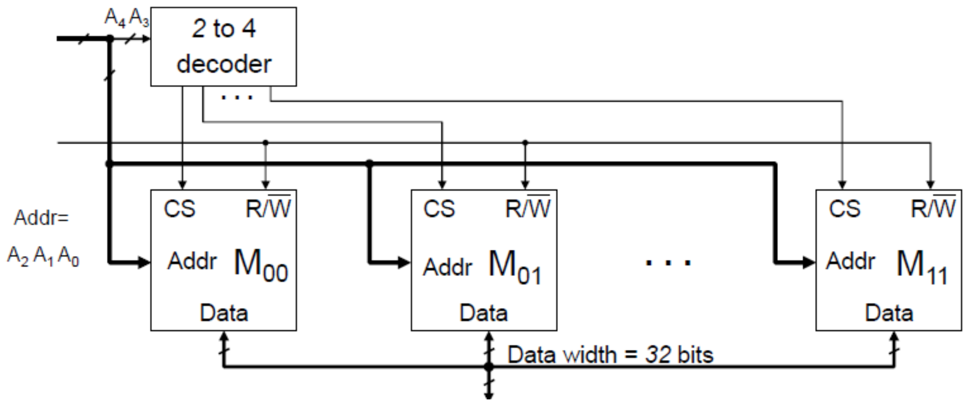
Assume two 32-bit data to be written in the above memory system at the corresponding addresses as shown in the table below:

Address $A_4 A_3 A_2 A_1 A_0$	01010	10100
Data Values (Hex)	4326016A	A3414640

Fill in the table below to show where the above data values for the given addresses will be written in the above memories:

$A_4 A_3 A_2 A_1 A_0$	**M_{00}**	**M_{01}**	**M_{10}**	**M_{11}**
01010	6A	01	26	43
10100	40	46	41	A3

2. Now the four memory modules M_{00} , M_{01} , M_{10} and M_{11} are interconnected using the below drawing:



Assume three 32-bit data to be written in the above memory system at the corresponding addresses as shown in the table below:

Address $A_4 A_3 A_2 A_1 A_0$	01010	10100	11000
Data Values (Hex)	4326016A	A3414640	FEF12306

Fill in the table below to show where the data values for the given addresses will be stored in the memories:

$A_4 A_3 A_2 A_1 A_0$	**M_{00}**	**M_{01}**	**M_{10}**	**M_{11}**
01010		4326016A		
10100			A3414640	
11000				FEF12306

[17 Points]

(Q4) A 4 GHz CPU uses a unified cache memory (Both IC and DC) with the following specs:

1. The cache hit time is 2 ns.
2. The cache hit probability is 0.93
3. The main memory access is 30 ns.
4. The CPI=2 clocks when instruction fetch and data fetch results in hits.
5. The probability of load/store instructions P(L/S)= 0.25

Answer each of the following questions:

(i) How many clocks there are in one DRAM access time? **(1 point)**

CR= 4 GHz; CT=0.25 ns; Tmm= 30 ns,
Hence, DRAMclocks = 30/0.25=120 clocks

(ii) Evaluate the Average Memory Access Time (AMAT), the average number of stalls per access (stalls/access), average number of stalls per instruction (stalls/ins) and the CPI. **(4 points)**

AMAT = 2 + 0.07*30 = 4.1 ns ;
stalls/access = 0.07*120 = 8.4 clocks
stalls/instr. = 1.25*0.07*120= 10.4 clocks
CPI = 2 + stalls/instr. = 2 + 10.4 = 12.4 clocks

(iii) Suppose the CPU is enhanced by increasing its clock rate to 5 GHz. Evaluate the speedup of the enhanced CPU compared to the original CPU. **(4 points)**

If CR=5 GHz, CT=0.2 ns, then DRAMclocks= 30/0.2 ns= 150 clocks
CPInew = 2 + 1.25*0.07*150 = 15.125 clocks
Speedup S = (CPIo * CTo) / (CPIn * CTn)=
(12.4 * 0.25)/ (15.125 * 0.2)= 3.1/3.025= 1.025
The enhanced CPU is 2.5% faster than the previous one.

(iv) Suppose the CPU uses a Write-Through policy with a write buffer. The write buffer access time is 10 ns but the probability to be found free is 0.75 and it takes 15ns to free one buffer slot if found full. Assume that when the buffer is full, it will be written after a slot is made free. Evaluate the AMAT for data accesses only. Assume that the probability of load and store instructions is the same. **(4 points)**

AMAT= Prob(read)xAvReadTimes + Prob(write)x[(Prob(WBfree) x WritePenalty + Prob(WBfull)x(TimetoFree +WritePenalty)] =

0.5* 4.1 + 0.5*(0.75*10 + 0.25* (15 + 10)) = 8.925 ns

(v) Suppose we use a split cache with IC and DC having:

1. IC: 1 ns access time and 0.95 hit probability,
2. DC: 2 ns access time and 0.91 hit probability.

The main memory is the same as above. Evaluate the AMAT(IC) and AMAT(DC) and the CPU AMAT. **(4 points)**

AMAT(IC) = 1 + 0.05*30 = 2.5 ns; AMAT(IC)= 2 + 0.09*30 = 4.7 ns
AMAT (CPU) = AMAT(IC) *1/1.25 + AMAT(IC)* 0.25/1.25= 2.94 ns

[12 Points]

(Q5)

(i) (9 points)

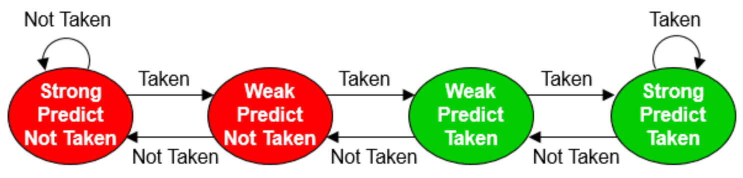
A sequence of four branches are shown in the first column of the Table below with their respective PC value, branch target address, and next PC. These braches are executed in four passes (pass 1 to 4). The actual branch outcomes of each branch in each pass are shown in columns 2, 3, 4 and 5, respectively, where T represents a taken branch and NT represents a not taken branch. Assume a Branch Target Buffer (BTB) is used for early prediction of taken branches.

Four Branches with their respective PC values, branch target address, and next PC.	Branch outcomes in four passes			
	Actual Branch outcome In pass 1	Actual Branch outcome In pass 2	Actual Branch outcome In pass 3	Actual Branch outcome In pass 4
00010 Beq -,-, (target=00100) 00011 …. ..	NT	T	NT	NT
00101 Beq -,-, (target=00111) 00110 …. ..	T	T	NT	T
01000 Beq -,-, (target=10010) 01001 …. ..	NT	T	T	NT
01011 Beq -,-, (target=01101) 01100 ….	T	NT	NT	T

1. Fill in the BTB entries for PC, target and initial prediction (T for all) for the above four branches. Fill in the prediction in BTB table following each pass (1 to 4) by assuming a 1-bit prediction. **(2 points)**

BTB			BTB Prediction just after pass k (1 to 4)			
PC	Target	Prediction initial	Prediction pass 1	Prediction pass 2	Prediction pass 3	Prediction pass 4
00010	00100	T	NT	T	NT	NT
00101	00111	T	T	T	NT	T
01000	10010	T	NT	T	T	NT
01011	01101	T	T	NT	NT	T

2. Repeat the above question by predicting the branch outcome using a 2-bit saturating counter (given below). Denote by NT1 and T1 the weak NT and weak T, respectively. Assume initial prediction of strong predict taken. **(2 points)**



BTB			BTB Prediction just after pass k (1 to 4)			
PC	Target	Prediction initial	Prediction pass 1	Prediction pass 2	Prediction pass 3	Prediction pass 4
00010	00100	T	T1	T	T1	NT1
00101	00111	T	T	T	T1	T
01000	10010	T	T1	T	T	T1
01011	01101	T	T	T1	NT1	T1

Answer the following questions:

a. Using all the above branches, evaluate the probability of correct prediction (Pcorrect). **(2 points)**

Probability of correct prediction Pcorrect = (1+3+2+1)/ 16= 7/16 = 0.4375

b. Assume that each miss-prediction incurs 2 stalls to the CPU and the probability an instruction to be branch is 0.18. Assume the CPI =2 when all the predictions are correct. Evaluate the CPI based on the above four passes. **(3 points)**

CPI = CPI + 0.18* (1-7/16)* 2 = 2 + 0.18* (9/16)* 2 = 2.2025

(ii) (3 points)

A loop can take one of the following two constructs:

```
Repeat:   …
           …
          Beq  -,-, Repeat
```

or

```
Repeat:   Beq  -,-, Exit
           …
           J  Repeat
Exit:
```

Prove that a 2-bit history prediction is always better than a 1-bit history prediction when the above loop (either of its constructs) is executed many times and its entry is maintained in BTB.

<u>Solution:</u>

Using a 1-bit history prediction, the prediction will be NT (T for type 2) at the end of the first pass because the last iteration the branch is not taken (taken for type 2). This is true disregarding the initial prediction. Hence, when the same loop is executed again there will always be one mis-prediction in first iteration of each subsequent execution of the loop for both type-1 and type-2.

Consider now a 2-bit history, the prediction will be T1 (N1 for type-2) at the end of the first pass because the last iteration the branch is not taken (T for Type 2). This is true disregarding the initial prediction. Hence, when the same loop is executed again there will always be correct prediction in first iteration of each subsequent execution of the loop.