

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 171 (Fall 2017-2018)
Final Exam
Tuesday Jan. 2, 2018
7:00-9:30 PM

Time: 150 minutes, Total Pages: 12

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 | 17 | |
| Q2 | 21 | |
| Q3 | 13 | |
| Q4 | 24 | |
| Total | 75 | |

Dr. Aiman El-Maleh
Dr. Marwan Abu-Amara

[17 Points]**(Q1)**

Compare the performance of a **single-cycle processor** and a **multi-cycle processor**. The delay times are as follows:

Instruction memory access time = 300 ps

Data memory access time = 300 ps

Instruction Decode and Register read = 200 ps

Register write = 100 ps

ALU delay = 200 ps

Ignore the other delays in the multiplexers, wires, etc. Assume the following instruction mix: **50% ALU, 10% load, 10% store, 10% branch, and 20% jump.**

a) **(4 points)** Compute the delay for each instruction class and the clock cycle for the single-cycle processor.

| Instruction Class | Instruction Memory | Decode and Register Read | ALU | Data Memory | Write Back | Total Delay |
|-------------------|--------------------|--------------------------|--------|-------------|------------|-------------|
| ALU | 300 ps | 200 ps | 200 ps | | 100 ps | 800 ps |
| Load | 300 ps | 200 ps | 200 ps | 300 ps | 100 ps | 1100 ps |
| Store | 300 ps | 200 ps | 200 ps | 300 ps | | 1000 ps |
| Branch | 300 ps | 200 ps | 200 ps | | | 700 ps |
| Jump | 300 ps | 200 ps | | | | 500 ps |

Clock cycle for the single-cycle processor = **1100 ps = 1.1 ns**

b) **(3 points)** Compute the **clock cycle** and the **average CPI** for the multi-cycle processor.

Clock cycle for the multi-cycle processor = $\max(300, 200, 100) = \mathbf{300}$

CPI for the multi-cycle processor = $(0.5 \times 4) + (0.1 \times 5) + (0.1 \times 4) + (0.1 \times 3) + (0.2 \times 2) = \mathbf{3.6}$

c) **(2 points)** Determine quantitatively if there is a speedup when using the multi-cycle processor.

Speedup = $(1100 \times 1) / (300 \times 3.6) = \mathbf{1.0185}$

- d) Assume that the **load** and the **store** instructions have been modified to use register-indirect addressing with a zero offset. Hence, there is NO need for the ALU to compute the memory address. The **load** and **store** instructions will have the following format, where **Rs** is the register that contains the memory address.

LW Rt, (Rs) # No immediate constant used
SW Rt, (Rs) # No immediate constant used

Find the following:

- i. (3 points) The **clock cycle** and the **average CPI** for the modified multi-cycle processor.

Clock cycle for the modified multi-cycle processor = $\max(300, 200, 100) = 300$

CPI for the modified multi-cycle processor = $(0.5 \times 4) + (0.1 \times 4) + (0.1 \times 3) + (0.1 \times 3) + (0.2 \times 2) = 3.4$

- ii. (2 points) Determine quantitatively if there is a speedup when using the modified multi-cycle processor instead of the original multi-cycle processor.

Speedup of modified multi-cycle processor over original multi-cycle processor = $3.6/3.4 = 1.0588$

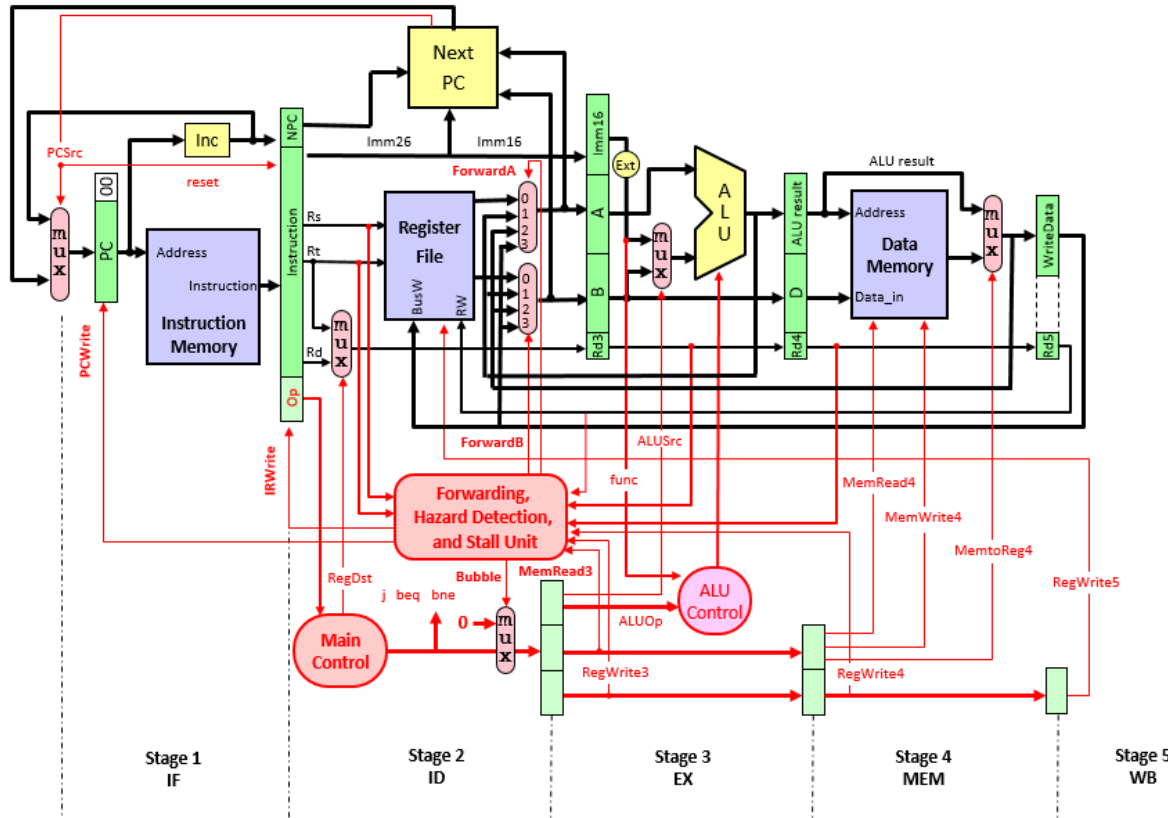
- e) (3 points) Assume that a program has the following instruction mix: **50%** ALU, **10%** load, **10%** store, **10%** branch, and **20%** jump. We want to improve the execution time of the program on the **original single-cycle processor** by a factor of 1.25. Assume that the execution time for the **Jump** instructions was enhanced to run 4 times faster. Determine quantitatively the speedup factor needed for the **Branch** instructions execution time to achieve an overall speedup of 1.25 for the program execution time.

$$Speedup = 1.25 = \frac{1}{\frac{0.2}{4} + \frac{0.1}{s_{Branch}} + (1 - 0.2 - 0.1)}$$

$$\Rightarrow s_{Branch} = \frac{0.1}{0.05} = 2$$

(Q2)

(i) Consider the 5-stage pipelined CPU design given below.



a) (4 points) Show the conditions that will be used for generating the ForwardA signals.

The conditions that will be used for generating the ForwardA signals

If $((Rs \neq 0) \text{ and } (Rs == Rd3) \text{ and } (RegWrite3))$ ForwardA $\leftarrow 1$
 Else if $((Rs \neq 0) \text{ and } (Rs == Rd4) \text{ and } (RegWrite4))$ ForwardA $\leftarrow 2$
 Else if $((Rs \neq 0) \text{ and } (Rs == Rd5) \text{ and } (RegWrite5))$ ForwardA $\leftarrow 3$
 Else ForwardA $\leftarrow 0$

b) (5 points) Show the control signals that will be used for stalling the pipeline due to data hazards due to load instruction along with their conditions. Show the necessary changes that need to be done to the design.

Condition for Stalling the pipeline due to Load Instruction:

if $((MemRead3 == 1) // \text{Detect Load in EX stage}$
 and $(ForwardA==1 \text{ or } ForwardB==1))$ Stall // RAW Hazard

OR:

if (MemRead3 == 1)
and (Rd3 ≠ 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.

- c) (2 points) Show the control signals that will be used for handling control hazards. Show the necessary changes that need to be done to the design.

When PCSrc=1, reset=1 and the content of IR register will be reset to 0 to make the fetched instruction a NOP.

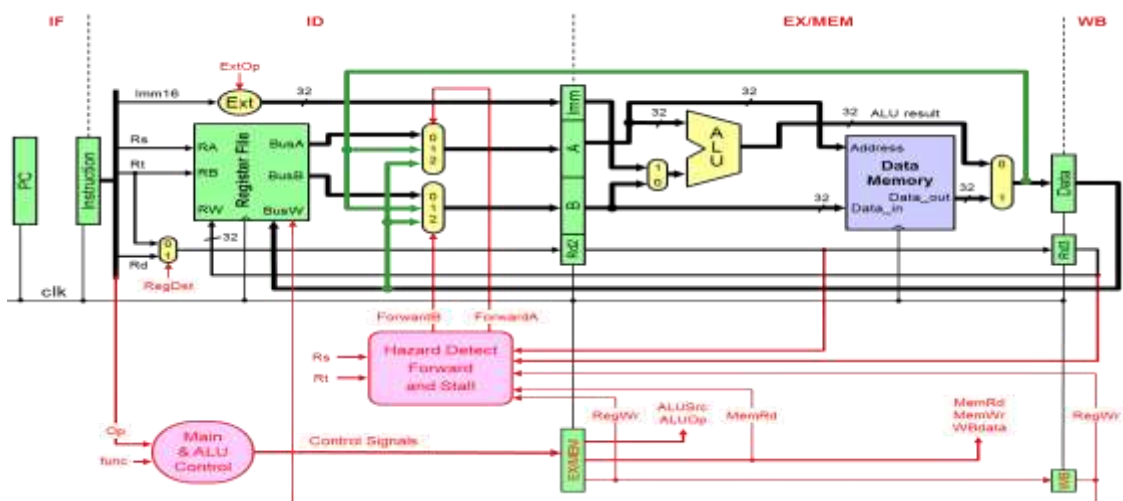
PCSrc=1 same as [(beq and Z) or (bne and Z') or J]

- (ii) (2 points) Suppose that the **load** and **store** instructions are modified to use **register-indirect addressing**, without using an offset. Thus, there is no need for the ALU to compute the memory address. The **load** and **store** instructions will have the following format, where **Rs** is the register that contains the memory address.

LW Rt, (Rs) # No immediate constant used

SW Rt, (Rs) # No immediate constant used

Accordingly, the MIPS pipeline can be modified to have only 4 stages: IF, ID, EX/MEM, and WB, as shown below. Discuss whether data hazards due to LW instruction will require stalling the pipeline or not. If not, explain how such data hazards will be handled.



No pipeline stall is required in this design as the data can be forwarded from the output of the MUX in EX/MEM stage to the ID stage.

(iii) (8 Points) Consider the following MIPS assembly language code:

```

I1:   ORI   $s0, $0, 10
I2:   SLL   $s0, $s0, 4
I3:   SW    $s1, ($s0)
I4:   LW    $s2, 4($s0)
I5:   SUB   $s3, $s2, $s0
I6:   SW    $s2, 4($s3)
    
```

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) supporting **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: ORI | IF | ID | EX | - | WB | | | | | | | | | | |
| I2: SLL | | IF | ID | EX | - | WB | | | | | | | | | |
| I3: SW | | | IF | ID | EX | M | - | | | | | | | | |
| I4: LW | | | | IF | ID | EX | M | WB | | | | | | | |
| I5: SUB | | | | | IF | X | ID | EX | - | WB | | | | | |
| I6: SW | | | | | | | IF | ID | EX | M | - | | | | |

Total number of clock cycles to execute the code = 10

[13 Points]**(Q3)**

- (i) **(4 Points)** Explain how Branch Target Buffer is used to achieve zero delay for a jump or a taken branch.

The Branch Target Buffer (BTB) is used in the IF stage. It uses the lower bits of the PC to index the BTB. Each BTB entry stores Branch/Jump address, Target Address and prediction bits. If the PC value matches the stored Branch/Jump address and prediction bits indicate that the branch should be taken, then the PC is updated using the target address stored in the BTB. This will result in zero delay for jump instructions and for branch instructions in case the prediction is correct.

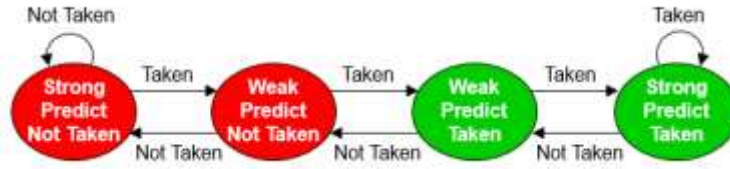
- (ii) A sequence of two branches are shown in the first column of the table below with their respective PC value, branch target address, and next PC. These branches are executed in four passes (pass 1 to 4). The actual branch outcomes of each branch in each pass are shown where T represents a branch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

| Two 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 |
| 0xA000 Beq --, (target=0xEBC0) 0xA004 | T | T | T | NT |
| 0xB000 Beq --, (target=0x0008) 0xB004 | T | NT | T | NT |

- a) **(3 points)** Fill in the BTB entries for PC, target and initial prediction (T for all) for the two branches above. Fill in the prediction in BTB table **after** each pass (1 to 4) by assuming a 1-bit prediction. Then, compute the probability of correct prediction.

| BTB | | | BTB Prediction just after pass k (k = 1 to 4) | | | |
|---|--------|--------------------|--|-------------------|-------------------|-------------------|
| PC | Target | Prediction initial | Prediction pass 1 | Prediction pass 2 | Prediction pass 3 | Prediction pass 4 |
| 0xA000 | 0xEBC0 | T | T | T | T | NT |
| 0xB000 | 0x0008 | T | T | NT | T | NT |
| Probability of Correct Prediction = $(3+1)/8 = 0.5$ | | | | | | |

b) (3 points) Repeat the question above 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 that the predictor is initialized to T1 (weak predict taken). Then, compute the probability of correct prediction.



| 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 |
| 0xA000 | 0xEBC0 | T1 | T | T | T | T1 |
| 0xB000 | 0x0008 | T1 | T | T1 | T | T1 |
| Probability of Correct Prediction = $(3+2)/8 = 0.625$ | | | | | | |

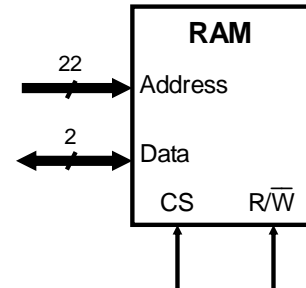
c) (3 points) Assume that a correctly predicted branch incurs zero stalls and a mis-predicted branch incurs 2 stalls. Assume the CPI = 1.4 when all the predictions are correct. Determine the CPI for using 2-bit predictors if 18% of the instructions are branches.

Average number of stalls per instruction = $0.18 \times (1 - 0.625) \times 2 = 0.135$
 CPI = $1.4 + 0.135 = 1.535$

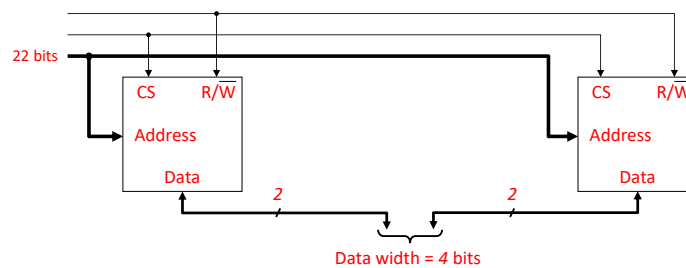
[24 Points]

(Q4)

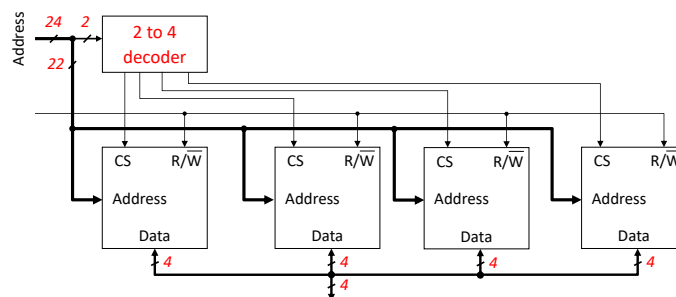
- (i) (4 points) Given a $4M \times 2$ memory block as shown below. Use this block to implement a $16M \times 4$ memory block.



First, we design a block of RAM of size $4M \times 4$ as follows:



Then, we design a block of RAM of size $16M \times 4$ using the block above as follows:



- (ii) (2 points) Consider a 64K byte direct-mapped cache that uses 32-byte blocks and write through policy. Compute the total number of **bits** required to store the **data**, **valid**, and **tag** bits in the cache.

Total Data bits = $64 \times 1024 \times 8 = 524,288$ bits

Total Valid bits = # of blocks = 2048 bits

Total Tag bits = # of blocks \times Tag bits = $2048 \times (32 - 11 - 5)$ bits = 32,768 bits

Total bits = Total (Data + Valid + Tag) Bits = **559,104 bits**

- (iii) (3 points) If the memory address consists of 32 bits, and a 64K byte 2-way set associative cache with 32 bytes cache blocks is used. Find the number of **tag** bits, **index** bits, and **offset** bits.

Block offset bits = **5 bits**

Index bits = $\log_2[64K/(2 \times 32)] =$ **10 bits**

Tab bits = $32 - 10 - 5 =$ **17 bits**

- (iv) (8 points) Given a 2-way set associative cache that uses a 32-bits memory address divided into 4 bits of offset, 8 bits of index, and 20 bits of tag. Starting with an empty cache, show the **tag**, **index**, and **way** (block 0 or 1) for each address and indicate whether a **hit** or a **miss**. The replacement policy is LRU.

| Address | Tag | Index | Way | Hit / Miss |
|------------|---------|-------|-----|------------|
| 0x001F3A70 | 0x001F3 | 0xA7 | 0 | Miss |
| 0x001F3A74 | 0x001F3 | 0xA7 | 0 | Hit |
| 0x002F3A78 | 0x002F3 | 0xA7 | 1 | Miss |
| 0x002F3C88 | 0x002F3 | 0xC8 | 0 | Miss |
| 0x001F3A78 | 0x001F3 | 0xA7 | 0 | Hit |
| 0x003F3A70 | 0x003F3 | 0xA7 | 1 | Miss |
| 0x002F3C80 | 0x002F3 | 0xC8 | 0 | Hit |
| 0x003F3C84 | 0x003F3 | 0xC8 | 1 | Miss |

- (v) A processor runs at 4.0 GHz and has a CPI = 2 for a perfect cache. Assume that load and store instructions are 15% of the instructions. The processor has an I-cache with a 5% miss rate and a D-cache with 6% miss rate. The hit time is 2 clock cycles for both caches. Assume that the time required to transfer a block of data from the main memory to the cache, i.e. miss penalty, is 30 ns.

- a) (2 points) Compute the number of stall cycles per instruction.

Combined misses per instruction = $5\% + 15\% \times 6\% = 0.059$

Miss Penalty = $30 \text{ ns} \times 4 \text{ GHz} = 120 \text{ cycles}$

Memory stall cycles per instruction = $0.059 \times 120 =$ **7.08 cycles**

- b) (1 point) Compute the overall CPI.

Overall CPI = $2.0 + 7.08 =$ **9.08 cycles**

c) (4 points) Compute the average memory access time (AMAT) in ns.

$$\begin{aligned} \text{AMAT(IC)} &= \text{Hit time(IC)} + \text{Miss rate(IC)} \times \text{Miss penalty} \\ &= (2 \text{ cycles} / 4 \text{ GHz}) + 0.05 \times 30 \text{ ns} = 2 \text{ ns} \end{aligned}$$

$$\begin{aligned} \text{AMAT(DC)} &= \text{Hit time(DC)} + \text{Miss rate(DC)} \times \text{Miss penalty} \\ &= (2 \text{ cycles} / 4 \text{ GHz}) + 0.06 \times 30 \text{ ns} = 2.3 \text{ ns} \end{aligned}$$

$$\begin{aligned} \text{AMAT} &= 1/(1+P_{LS}) \times \text{AMAT(IC)} + P_{LS}/(1+P_{LS}) \times \text{AMAT(DC)} \\ &= 1/(1+0.15) \times 2 \text{ ns} + 0.15/(1+0.15) \times 2.3 \text{ ns} = \mathbf{2.04 \text{ ns}} \end{aligned}$$