# COE 301 / ICS 233
# Computer Organization

# Final Exam – Spring 2017

Friday, May 19, 2017

7:30 – 10 AM

Computer Engineering Department

College of Computer Sciences & Engineering

King Fahd University of Petroleum & Minerals

Student Name: ______________________________

Student ID: ______________________________

| | | | |
|---|---|---|---|
| Q1 | / 14 | Q2 | / 16 |
| Q3 | / 15 | Q4 | / 18 |
| Q5 | / 10 | Q6 | / 20 |
| Q7 | / 10 | | |
| Total | / 103 | | |

Notes:

- This is a closed book and closed notes exam.
- Mobile phones and tablets are not allowed and should be switched off.
- Using unauthorized information on an exam is a severe violation of academic honesty. Detected cases will receive a failing grade.

## Question 1: True or False

(14 pts) **Explain the reason why it is true or false for a full mark**.

**a)** (2 pts) A pipelined datapath must have separate instruction and data memories because the format of instructions is different from the format of data.

**b)** (2 pts) Allowing ALU instructions to write back their result in the 4th stage rather than the 5th stage, improves the performance of a MIPS 5-stage pipeline.

**c)** (2 pts) In the MIPS 5-stage pipeline, some but not all RAW data hazards can be eliminated by forwarding.

**d)** (2 pts) Name dependences such as Write-After-Read and Write-After-Write do not cause any hazard in the 5-stage pipeline and do not require special handling.

**e)** (2 pts) A directly-mapped cache does not need to store tags because it is directly indexed. However, a fully-associative cache requires tags because it is not indexed.

**f)** (2 pts) When comparing a set-associative with a directly-mapped cache with the same capacity (data size), the set-associative cache decreases the cache miss rate but increases the hit time.

**g)** (2 pts) Each block in a write-through cache has a Modified bit to indicate whether the block is modified or not.

## Question 2: Single-Cycle / Multi-Cycle / Pipelined Performance

(16 pts) Compare the performance of a **single-cycle** / **multi-cycle** / **pipelined** processor. The delay times are as follows:

Instruction memory access time = 500 ps
Instruction Decode and Register read = 300 ps
ALU delay = 300 ps
Data memory access time = 500 ps
Register write = 200 ps

Ignore the other delays in the multiplexers, wires, etc. Assume the following instruction mix: 35% ALU, 20% load, 10% store, 25% branch, and 10% jump.

**a)** (5 pts) Compute the delay for each instruction class for the **single-cycle** processor.

| Instruction Class | Instruction Memory | Decode and Register Read | ALU | Data Memory | Write Back | Total Delay |
|---|---|---|---|---|---|---|
| ALU | | | | | | |
| Load | | | | | | |
| Store | | | | | | |
| Branch | | | | | | |
| Jump | | | | | | |

**b)** (2 pts) Compute the clock cycle for the **single-cycle** processor.

**c)** (2 pts) Compute the clock cycle for the **multi-cycle** processor.

**d)** (3 pts) Compute the average CPI for the **multi-cycle** processor.

**e)** (2 pts) What is the speedup factor of the **multi-cycle** over the **single-cycle** processor?

**f)** (2 pts) What is the speedup factor of the **pipelined** over the **single-cycle** processor?

## Question 3: Performance of a MIPS program

(15 pts) The following code fragment processes two single-precision floating-point arrays *A* and *B*, and modifies the array *B*. Each array consists of **500** single-precision floats. The addresses of *A* and *B* are stored in **$a0** and **$a1** respectively. Note that there is an outer loop and an inner loop in the MIPS assembly language code.

```
    li      $a2, 500          # $a2 = 500
    move    $t0, $a0          # $t0 = address of array A
    move    $t4, $zero        # $t4 = 0
    mtc1    $zero, $f4        # $f4 = 0 (move to coprocessor 1)

outer:                        # outer loop
    lwc1    $f0, ($t0)        # load single-precision float
    add.s   $f4, $f4, $f0
    move    $t1, $a1          # $t1 = address of Array B
    move    $t5, $zero        # $t5 = 0

inner:                        # inner loop
    lwc1    $f1, ($t1)        # load single-precision float
    mul.s   $f2, $f1, $f0
    div.s   $f3, $f2, $f4
    swc1    $f3, ($t1)        # store single-precision float
    addi    $t1, $t1, 4
    addi    $t5, $t5, 1
    bne     $t5, $a2, inner   # end of inner loop

    addi    $t0, $t0, 4
    addi    $t4, $t4, 1
    bne     $t4, $a2, outer   # end of outer loop
```

The above code is executed on a processor with a **2 GHz** clock that requires the following number of cycles for each instruction:

| Instruction | Cycles |
|---|---|
| li, move, mtc1, addi | 3 |
| lwc1, swc1 | 5 |
| add.s, mul.s | 10 |
| div.s | 20 |
| bne | 4 |

**a)** (3 pts) Count the total number of instructions executed by the above code fragment, including those executed outside the outer loop.

**b)** (6 pts) How many cycles does it take to execute the above code?

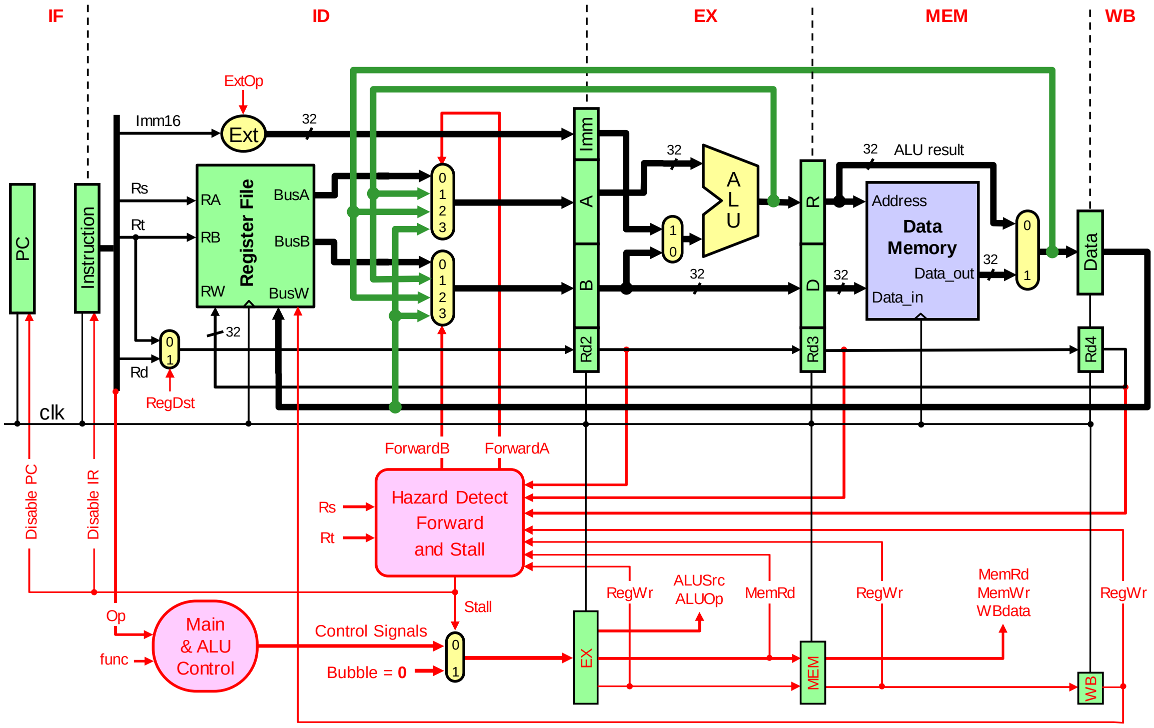
**c)** (2 pts) What is the execution time in nanoseconds?

**d)** (2 pts) What is the average CPI for the above code?

**e)** (2 pts) What is the MIPS rate for the above code?

## Question 4: MIPS 5-Stage Pipeline

(18 pts) The following is a 5-stage pipeline for the MIPS processor that implements a subset of the MIPS instruction set, and that supports **forwarding** and **pipeline stall**.




Consider the following MIPS code that is being executed on the above 5-stage pipeline:

```
I1: lw    $t1, 8($t0)
I2: addi  $t2, $t1, 2
I3: or    $t3, $t1, $t2
I4: lw    $t4, -4($t3)
I5: sub   $t5, $t3, $t4
```

**a)** (8 pts) Complete the following table showing the timing of the above code on the 5-stage pipeline MIPS processor (IF, ID, EX, MEM, WB). Draw an arrow showing forwarding between the stage that provides the data and the stage that receives the data. Show all stall cycles by placing an X in the box to represent a stall cycle.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I1: LW | | | | | | | | | | | | | | | |
| I2: ADDI | | | | | | | | | | | | | | | |
| I3: OR | | | | | | | | | | | | | | | |
| I4: LW | | | | | | | | | | | | | | | |
| I5: SUB | | | | | | | | | | | | | | | |

**b)** (10 pts) Complete the following table pertaining to the control signals needed for the execution of the following MIPS code on the 5-stage pipeline MIPS processor **<u>during the first 9 clock cycles</u>**. Use **?** for any control signal that is **unknown** during a given clock cycle and **X** for a **don't care**. Assume that instruction **I1:LW** is being fetched during **clock cycle 1**. The control signals needed in the first clock cycle are shown as an example.

```
I1: lw     $t1, 8($t0)
I2: addi   $t2, $t1, 2
I3: or     $t3, $t1, $t2
I4: lw     $t4, -4($t3)
I5: sub    $t5, $t3, $t4
```

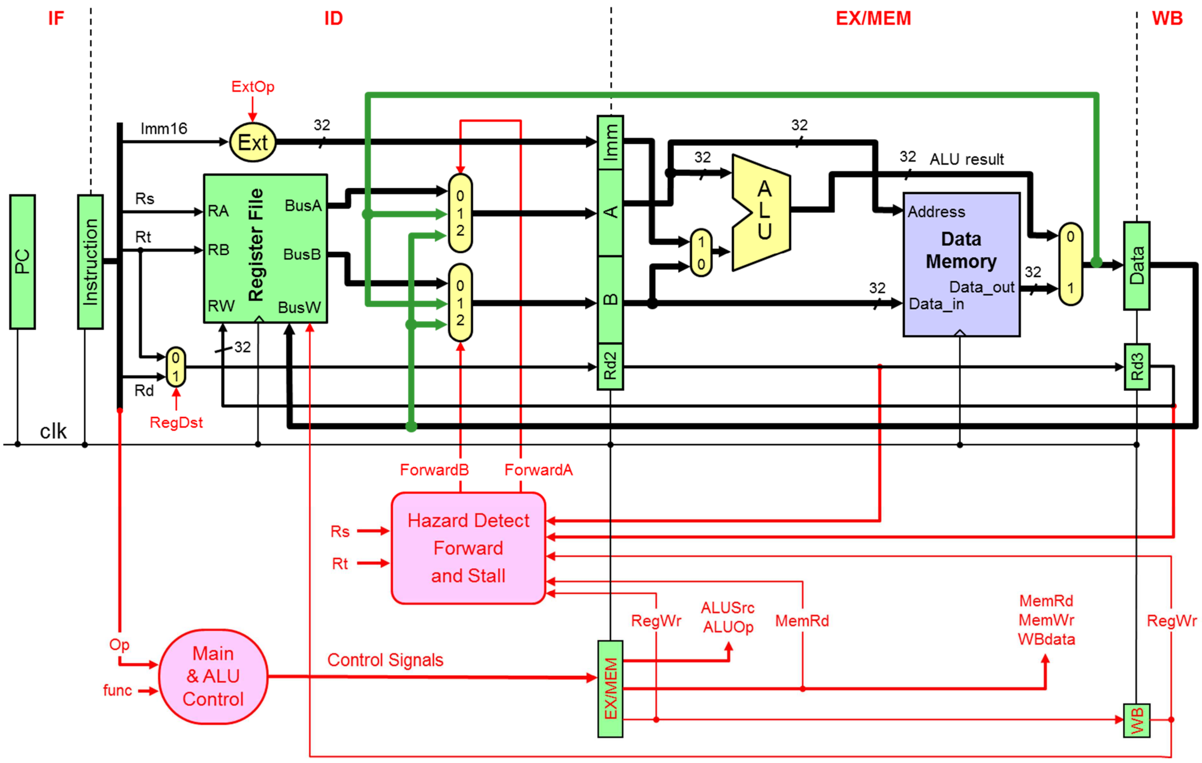
| | IF | | ID | | | | | EX | | | | MEM | | | | WB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clock Cycle | Disable PC (1=disable) | Disable IR (1=disable) | RegDst (0=Rt, 1=Rd) | ExtOp(0=zero,1=sign) | ForwardA | ForwardB | Stall | EX.RegWr | ALUSrc (0=B, 1=Imm) | ALUOp (add, sub, …) | EX.MemRd | MEM.RegWr | MEM.MemRd | MEM.MemWr | MEM.WBdata (0=ALU,1=MEM) | WB.RegWr |
| 1 | 0 | 0 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| 2 | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | |

## Question 5: MIPS 4-Stage Pipeline

(10 pts) Given that the **load** and **store** instructions use **register-indirect addressing**, and always have a **zero offset**, 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. The modified pipeline MIPS processor eliminates the load delay and the RAW data hazard due to a load instruction.




**a)** (4 pts) Provide the **control signals values** needed for executing the **LW** instruction.
**b)** (6 pts) Write the **if-statement** that will be used for generating the **ForwardA** signal.

## Question 6: Cache Memory

(20 pts) Consider a **2-way set-associative cache** with **2048 blocks**, where each block has a size of **64 bytes**.

**a)** (4 pts) Compute the number of sets and the cache data size (do NOT include the valid, modified, and tag bits).

**b)** (3 pts) If the memory address consists of 32 bits, find the number of tag bits, index bits, and block offset bits.

**c)** (3 pts) Given that the cache is a write-back cache, compute the total number of bits required to store the valid, modified, and tag bits in the cache.

**d)** (10 pts) 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 FIFO.

| Address | Tag | Index | Way | Hit / Miss |
|---|---|---|---|---|
| 0x001F3A70 | | | | |
| 0x002E3A74 | | | | |
| 0x001F3A80 | | | | |
| 0x001F3A78 | | | | |
| 0x002E3A78 | | | | |
| 0x007F3A80 | | | | |
| 0x001F3A84 | | | | |
| 0x001F3A74 | | | | |

## Question 7: Cache Performance

(10 pts) A processor runs at 3.0 GHz and has a CPI=1.5 for a perfect cache (i.e. without including the stall cycles due to cache misses). Assume that load and store instructions are 25% of the instructions. The processor has an I-cache with a 5% miss rate and a D-cache with 4% miss rate. The hit time is 1 clock cycle 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 40 ns.

**a)** (5 pts) Compute the number of stall cycles per instruction.

**b)** (2 pts) Compute the overall CPI.

**c)** (3 pts) Compute the average memory access time (AMAT) in ns.