# CMPE 110 Computer Architecture
# Fall 2015, Homework #2

Computer Engineering
UC Santa Cruz

October 19, 2015

**Name:** ______________________________

**Email:** ______________________________

**Submission Guidelines:**

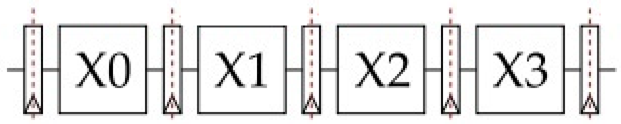
- This homework is due on Monday 11/2/15.
- The homework must be submitted to ecommons by 7:59am.
    - Anything later is a late submission
- **Please write your name and your UCSC email address**
- **The homework should be "readable" without too much effort**
    - The homework must be typed and submitted as a single file in PDF format
    - Please name your homework file cmpe110-hw2-yourcruzid.pdf
    - Please keep your responses coherent and organized or you may lose points
- Provide details on how to reach a solution. **An answer without explanation gets no credit. Clearly state all assumptions.**
- Points: 64 = 18 + 18 + 10 + 18

| **Question** | **Part A** | **Part B** | **Part C** | **Total** |
|---|---|---|---|---|
| **1** | | | | |
| **2** | | | | |
| **3** | - | - | - | |
| **4** | | | | |
| **Total** | | | | |

## Question 1. Pipelining Hazards (18 points)

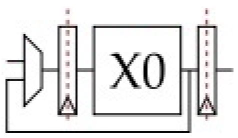
Consider the MIPS assembly code given below. We want to run this code on a 5-stage pipelined processor, with some modifications. The processor is a typical 5-stage pipeline (F-D-X-M-W), with the following exceptions:

- The multiplier block used to execute the mul instruction is pipelined into four stages:



  This means that a multiply instruction runs through the pipeline as follows: F-D-X0-X1-X2-X3-M-W and up to four multiply instructions maybe in-flight at a time. All other instruction types are blocked from the execute stage while any of the multiply stages are being used.

- The divider block used to execute the div instruction is iterative and takes four cycles:



  This means that a divide instruction runs through the pipeline as follows: F-D-X0-X0-X0-X0-M-W. All other instructions are blocked from the execute stage while a division is being done.

```
1      xor r0, r0, r0
2      addiu r1, r0, 10
3      j L1
4      loop: lw r3, 0(r2)
5      mul r4, r3, r3
6      mul r3, r3, r1
7      addiu r0, r0, 1
8      div r3, r4, r3
9      sw r3, 0(r2)
10     addiu r2, r2, 4
11 L1: bne r0, r1, -8
```

## Question 1.A Stalling for Structural Hazards (6 points)

Draw a pipeline diagram (table) showing the execution of the MIPS code through the first iteration of the loop, without bypassing. **Assume data hazards and structural hazards are resolved using only stalling. Assume the processor assumes branches are not taken, until they are resolved.** What is the CPI of the entire program?

## Question 1.B Bypassing for Data Hazards (6 points)

Draw a pipeline diagram similar to Part A, but **now assume the processor has data bypassing.** What is the CPI of the entire program?

## Question 1.C Compiler Optimization for Control Hazards (6 points)

Assume the compiler can optimize the code to further reduce stalls. **Assume the processor has data bypassing similar to Part B. Also assume now that the processor has a single branch delay slot.** This means that the instruction immediately following a jump or branch cannot be squashed. Re-order the code to improve the CPI of the entire program. Show the new ordering of the code and the corresponding pipeline diagram. What is the new CPI of the program?

## Question 2. Datapath Bypassing (18 points)

Assume in a given emerging technology, the execute logic is significantly slower than the memory delay as compared to the standard CMOS technology used in modern processors. In this situation, the execute stage of the standard five-stage pipeline might be significantly longer than the other stages, and as a consequence, we might want to split this stage into two pipeline stages. In this problem we will be investigating the implications of using a two-cycle pipelined integer ALU. Our new pipelined processor will have the following six stages:

- F - instruction fetch
- D - decode and read registers
- X0 - first half of the ALU operation
- X1 - second half of the ALU operation
- M - data memory read/write
- W - write registers

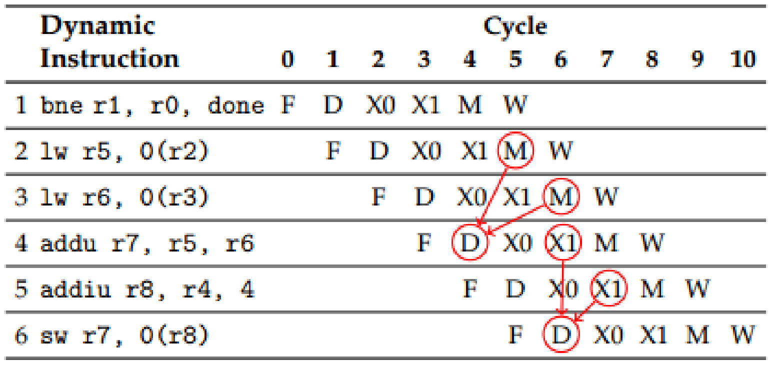
The figure on the next page shows the datapath of the new six-stage datapath. Notice that this datapath currently does not allow any data bypassing. Also notice that conditional branches are resolved by the end of the X1 stage. Assume that this instruction set does not include a branch delay slot.

A simple code sequence is shown below and illustrates the read-after-write (RAW) data dependencies present given the current pipeline assuming the branch is not taken. An arrow indicates a RAW dependency, and since some of these arrows point backwards they create data hazards. Please note the backwards arrow representing the RAW hazard between the X1 stage of the addiu instruction and the D stage of the sw instruction. The result of the addiu is not ready until the end of the X1 stage, but the sw instruction needs the store data at the beginning of the X0 stage. In this microarchitecture, the store addresses and store data must both be ready at the beginning of the X0 stage.

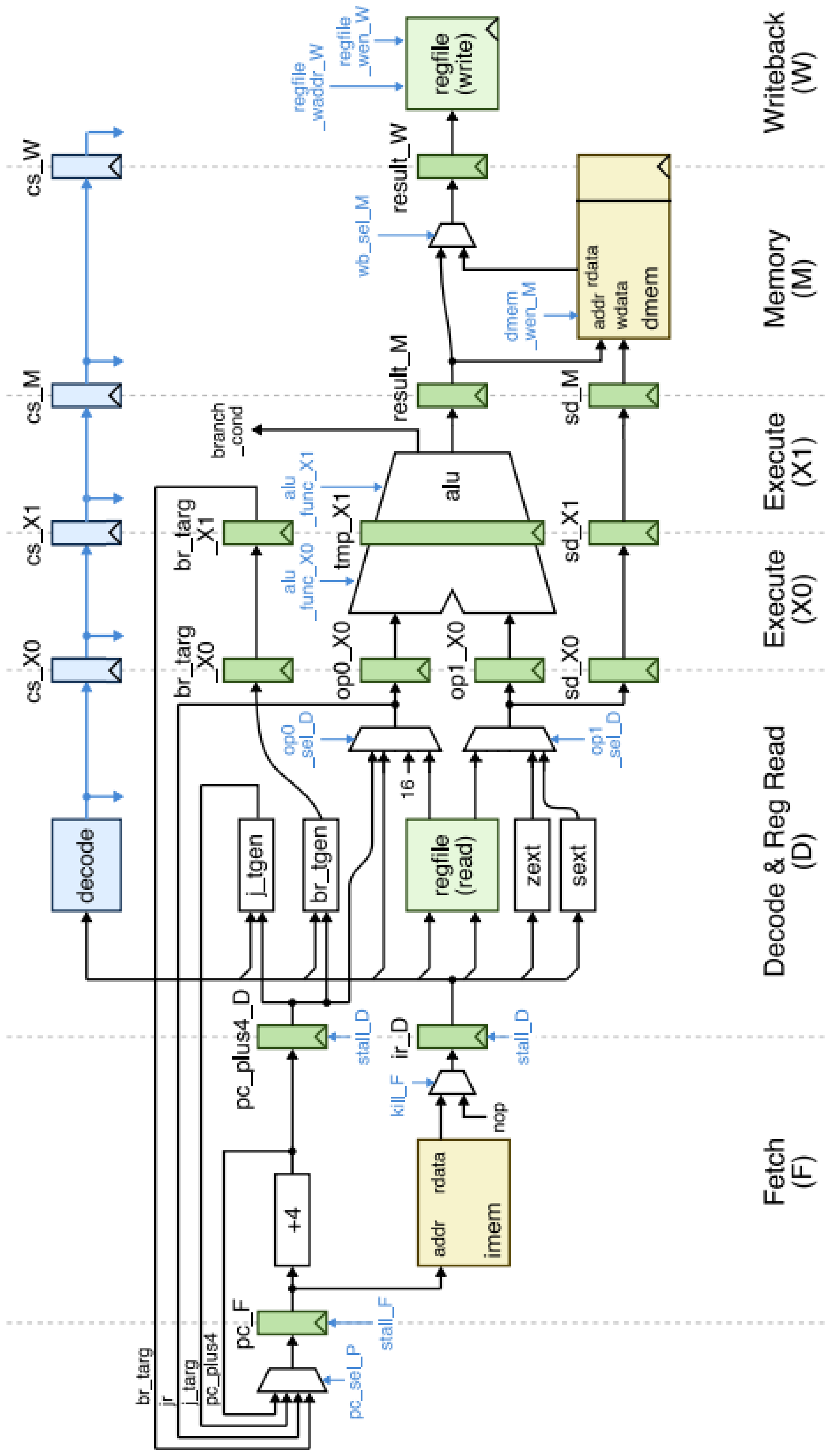
For all parts, assume the branch is not taken, and that the microarchitecture always speculatively executes the not taken path.

**Static Instr Sequence**

```
1     bne    r1, r0, done
2     lw     r5, 0(r2)
3     lw     r6, 0(r3)
4     addu   r7, r5, r6
5     addiu  r8, r4, 4
6     sw     r7, 0(r8)
7     ...
8  done:
9     addiu  r2, r2, 1
10    addiu  r8, r9, 1
```



| Dynamic Instruction | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bne r1, r0, done | F | D | X0 | X1 | M | W | | | | | |
| 2 lw r5, 0(r2) | | F | D | X0 | X1 | M | W | | | | |
| 3 lw r6, 0(r3) | | | F | D | X0 | X1 | M | W | | | |
| 4 addu r7, r5, r6 | | | | F | D | X0 | X1 | M | W | | |
| 5 addiu r8, r4, 4 | | | | | F | D | X0 | X1 | M | W | |
| 6 sw r7, 0(r8) | | | | | | F | D | X0 | X1 | M | W |



Fetch (F)
Decode & Reg Read (D)
Execute (X0)
Execute (X1)
Memory (M)
Writeback (W)
pc_F
stall_F
pc_sel_P
br_targ
jr
j_targ
pc_plus4
+4
imem
addr
rdata
kill_F
nop
pc_plus4_D
ir_D
stall_D
stall_D
decode
j_tgen
br_tgen
regfile (read)
zext
sext
16
op0_sel_D
op1_sel_D
cs_X0
cs_X1
cs_M
cs_W
br_targ_X0
br_targ_X1
op0_X0
op1_X0
sd_X0
alu_func_X0
alu_func_X1
tmp_X1
sd_X1
alu
branch_cond
result_M
sd_M
dmem_wen_M
dmem
addr
wdata
rdata
wb_sel_M
result_W
regfile_waddr_W
regfile_wen_W
regfile (write)

## Question 2.A Resolving Data Hazards by Stalling (6 points)

Assume we implement stall logic to correctly prevent RAW hazards. Draw a pipeline diagram similar to the one shown in the example on the previous pages that shows which instructions have to stall in which stages. Your pipeline diagram should execute the same instructions as in the example code. Either draw the microarchitectural RAW dependencies as in the example or list them clearly. Assume that the Register File can read new values in the Registers on the same cycle they are written.

## Question 2.B Implementing Data Bypassing (6 points)

How would you modify the given datapath diagram from the previous page to implement bypassing to reduce the stalls from the pipeline diagram in Part A? Either describe your changes clearly using labels given in the original diagram or draw your changes in a diagram. Is it possible for bypassing to remove stalling due to RAW hazards completely? Why or why not?

## Question 2.C Bypassing Execution (6 points)

Draw a pipeline diagram similar to the one shown in the example above that shows the behavior of the pipeline after implementing bypassing in Part B. Your pipeline diagram should execute the same instructions as in the example code. Either draw the microarchitectural RAW dependencies as in the example or list them clearly.

## Question 3. Branch Prediction (10 points)

Assume the following instruction mix for a 5-stage MIPS pipeline:

| **Instruction Type** | **Proportion** |
|---|---|
| Load | 12% |
| Store | 13% |
| Arithmetic | 50% |
| Branch | 25% |

- The processor's base CPI = 1.
- We use "always branch not taken" scheme for branch prediction.
- 55% of the branch are taken.
- 30% of the load instructions are immediately followed by an instruction that uses the loaded value.
- There are no other stalls in the pipeline.
- Branch misprediction has a 4 cycle penalty.
- Stalling due to a load instruction has a 2 cycle overhead.

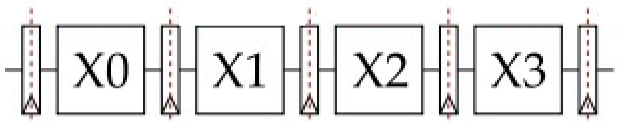
Calculate the CPI of this pipeline.

## Question 4. Out-of-Order Execution (18 points)

**This question is based on the same pipeline and code as question 1.**

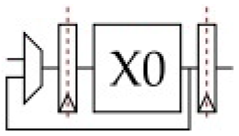
Consider the MIPS assembly code given below. We want to run this code on a 5-stage pipelined processor, with some modifications. The processor is a typical 5-stage pipeline (F-D-X-M-W), with the following exceptions:

- The multiplier block used to execute the mul instruction is pipelined into four stages:

  This means that a multiply instruction runs through the pipeline as follows: F-D-X0-X1-X2-X3-M-W and up to four multiply instructions maybe in-flight at a time.

- The divider block used to execute the div instruction is iterative and takes four cycles:

  This means that a divide instruction runs through the pipeline as follows: F-D-X0-X0-X0-X0-M-W.

- This processor supports out-of-order execution of instructions.

```
1       xor r0, r0, r0
2       addiu r1, r0, 10
3       j L1
4       loop: lw r3, 0(r2)
5       mul r4, r3, r3
6       mul r3, r3, r1
7       addiu r0, r0, 1
8       div r3, r4, r3
9       sw r3, 0(r2)
10      addiu r2, r2, 4
11 L1:  bne r0, r1, -8
```

## Question 4.A Out-of-Order with no Bypassing (6 points)

Draw a pipeline diagram (table) showing the out-of-order execution of the MIPS code through the first iteration of the loop, without bypassing. **Assume data hazards and structural hazards are resolved using only stalling. Assume the processor assumes branches are not taken, until they are resolved.**

## Question 4.B Out-of-Order with Bypassing (6 points)

Draw a pipeline diagram similar to Part A, but **now assume the processor has data bypassing.**

## Question 4.C Program Latency (6 points)

Determine the number of cycles it takes to execute all iterations of the loop for both the scenario in Part A and the scenario in Part B. Justify your answer.