# CS146 Computer Architecture

Fall 2017

Midterm Exam

This exam is worth a total of 100 points.
Note the point breakdown below and budget your time wisely.

To maximize partial credit, show your work and state any assumptions explicitly.

| | |
|---|---|
| 1 | /35 |
| 2 | /20 |
| 3 | /10 |
| 4 | /12 |
| 5 | /12 |
| 6 | /10 |
| Total | /100 |

Name:

## 1. Multiple Choice and Short Answer (35pts)

1.1 (10 points) Amdahl's Law

a. (5 points) Consider a workload where 50% of the execution time consists of multimedia processing for which the MMX instruction set extensions might be helpful. According to Amdahl's law, what is the maximum speedup that can be achieved by implementing them?

b. (5 points) Now, say that you work at Intel and the MMX designers claim that multimedia code sequences will see a 3.5 times (3.5X) speedup by using the MMX extensions. What is the fraction of the execution time that must be multimedia code in order to achieve an overall speedup of 1.8X?

1.2 (9 points) CISC vs. RISC

When pipelined microprocessors were first becoming more common (early to mid 80's) designers believed that RISC instruction sets were easier to pipeline because…?
(Please give 3 reasons)

1.3 (5 points) In spite of this, high-performance pipelined implementations of CISC instruction sets have been successfully built; various VAX and x86 implementations are examples including Intel's P6-Microarchitecture discussed in class. The most effective/common implementation strategy used by these machines has been to:

a. pipeline the CISC instructions, despite their wide variability in instruction execution times, and use elaborate memory disambiguation techniques to avoid stalls due to memory address calculations.

b. Pipeline the CISC instructions and handle the extra structural hazards using aggressive scoreboarding techniques.

c. Use link-time techniques to break CISC instructions into small RISC-like operations that are more easily pipelined

d. Use run-time techniques to break CISC instructions into easily-pipelined RISC-like operations

1.4 (6 points) Pipelining Limits

We have seen how pipelining improves the instruction throughput increasing effective performance. Machines with deeper pipelines perform less work per pipestage but have more “in-flight” instructions processing at the same time allowing instructions to complete at a higher rate. In class we discussed several reasons why the effectiveness of deeply pipelined machines can be limited – too much pipelining can be detrimental. Describe two reasons here.

1.5 (6 points) Limits of Loop Unrolling

We have seen how loop unrolling can significantly improve performance by removing loop overhead and providing a better opportunity for the compiler to generate an efficient static schedule. In class we discussed several reasons why loop unrolling cannot be performed indefinitely – too much loop unrolling can limit performance. Describe two reasons here.

## 2. Pipelining (20 Points)

For this question, consider the code segment below. Assume that full bypassing/forwarding has been implemented. Assume that the initial value of register R23 is much bigger than the initial value of register R20. Assume that all memory references hit in the caches and TLBs. **Assume that both load-use hazards and branch delay slots are hidden using delay slots.** You may-not reorder instructions to fill such slots, but if a subsequent instruction is independent and is properly positioned, you may assume that it fills the slot. Otherwise, fill slots with additional no-ops as needed.

```
LOOP:     lw    R10, X(R20)
          lw    R11, Y(R20)
          subu  R10, R10, R11
          sw    Z(R20), R10
          addiu R20, R20, 4
          subu  R5, R23, R20
          bnez  R5, LOOP
          nop                  ; 1 delay slot
```

a. (5 points) On the grid page at the end of the exam, draw a pipeline diagram of 2 iterations of its execution on a standard 5-stage MIPS pipeline. (You may want to turn it horizontally). Assume that the branch is resolved using an ID control point. In the box below, write the total number of cycles required to complete 2 iterations of the loop.

| Cycles = |  |
|---|---|

b. (15 points) On the second grid page that follows, draw a pipeline diagram of 2 iterations of the loop on the pipeline below. Note that the loop has a single branch delay slot nop included – you may need to add more. You can not assume anything about the program's register usage before or after this code segment. Fill in the boxes below.

| | |
|---|---|
| Pipeline Branch Delay = | |
| Pipeline Load Delay = | |
| Cycles = | |

Pipeline:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IF1 | IF2 | ID | RF | EX1 | EX2 | M1 | M2 | WB | | | | | |
| | IF1 | IF2 | ID | RF | EX1 | EX2 | M1 | M2 | WB | | | | |
| | | IF1 | IF2 | ID | RF | EX1 | EX2 | M1 | M2 | WB | | | |
| | | | IF1 | IF2 | ID | RF | EX1 | EX2 | M1 | M2 | WB | | |
| | | | | IF1 | IF2 | ID | RF | EX1 | EX2 | M1 | M2 | WB | |
| | | | | | IF1 | IF2 | ID | RF | EX1 | EX2 | M1 | M2 | WB |

IF1: Begin Instruction Fetch
IF2: Complete Instruction Fetch
ID: Instruction Decode
RF: Register Fetch
EX1: ALU operation execution begins. Branch target calculation finishes. Memory address calculation. Branch condition resolution calculation begins.
EX2: Branch condition resolution finishes. Finish ALU ops. (But branch and memory address calculations finish in a single cycle).
M1: First part of memory access, TLB access.
M2: Second part of memory access, Data sent to memory for stores OR returned from memory for loads.
WB: Write back results to register file

## 3. Multimedia ISAs and Conditional MOVs (10 Points)

Absolute value is expressed as `A = abs(B)`. In high-level code:

```
If (B<0) {A=-B;} else {A=B;)
```

In MIPS-style code this would look something like the following (R2 = B, R1 = A):

```
        BLTZ R2,THEN:  ; Check if R2 < 0, Jump to Then
        ADDI R1,R2,0   ;  R1 = R2 + 0; Else Clause
        JUMP END:      ; Skip over Then Clause
THEN:   SUBI R1,0,R2   ;  R1 = 0 – R2; Then Clause
END:
```

In class, we have seen that conditional branches are detrimental to performance and we have seen two methods to remove conditional branches.

a. (5 points) Using the saturating arithmetic features of a multimedia ISA code the absolute value function without using any branch or jump instructions. You can perform the computation with the following six instructions (you may not need all of them). You can perform the absolute value operation on subwords (ie. don't worry about shifting or extracting).

```
HADD, HADD,us, HADD,ss, HSUB, HSUB,us, HSUB,ss
```

Here `HADD,us` uses unsigned saturating arithmetic and `HADD,ss` uses saturating arithmetic.

b. (5 points) Using Conditional Move operations code the absolute value function without using any branch or jump instructions. In this problem just perform the absolute value operation on a 64-bit integer value. You may use CMOV's of the form:

```
CMOVGTZ      R1, R2, R3    // if (R1 > 0) R2 = R3
CMOVLTZ      R1, R2, R3    // if (R1 < 0) R2 = R3
CMOVEQZ      R1, R2, R3    // if (R1 ==0) R2 = R3
```

## 4. Branch Prediction (12 Points)

The following series of branch outcomes occurs for a single branch in a program. (T means the branch is taken, N means the branch is not taken).

T T T N T N T T T N T N T

a. (4 points) Assume that we are trying to predict this sequence with a BHT using a 1-bit counter. The counters of the BHT are initialized to the N state. Which of the branches would be mispredicted? Use the following table. You may assume that this is the only branch in the program.

| Predictor State Before Prediction | Branch Outcome | Mis-Prediction? |
|---|---|---|
| N | T | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

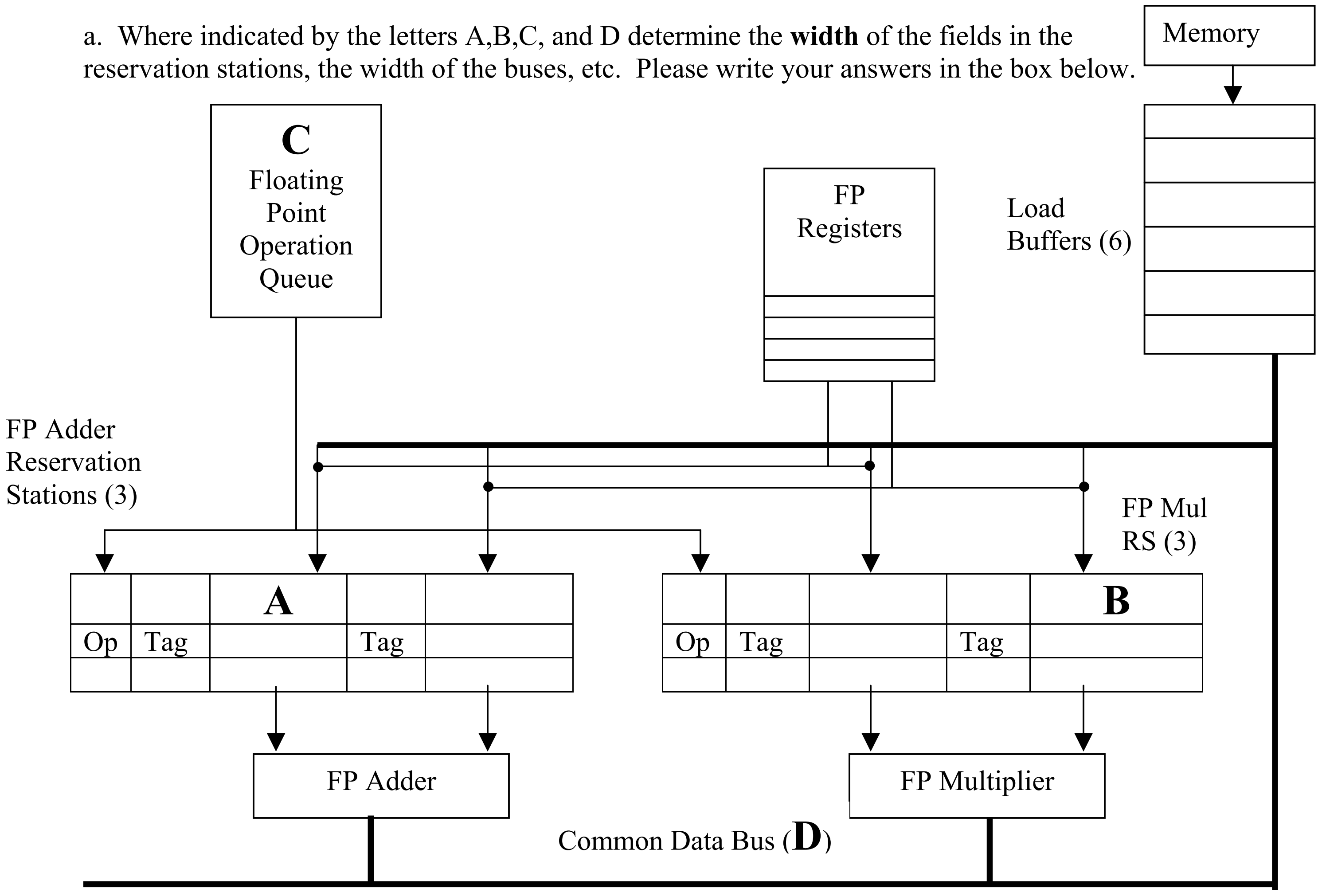
b. (8 points) Draw the state-transition diagram for a BHT scheme using 2-bit saturating counters. Repeat this exercise with a 2-bit saturating counter initialized to Weakly-Not-Taken.

| Predictor State Before Prediction | Branch Outcome | Mis-Prediction? |
|---|---|---|
| W-N | T | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## 5. Tomasulo's Algorithm (12 pts)

The drawing below depicts the basic structure of Tomasulo's algorithm as described in the textbook and during class. This is the version **without** a reorder buffer – all renaming occurs in the reservation stations.

a. Where indicated by the letters A,B,C, and D determine the **width** of the fields in the reservation stations, the width of the buses, etc. Please write your answers in the box below.



| | | | |
|---|---|---|---|
| A= | | C= | |
| B= | | D= | |

b. Now consider the three reservation stations associated with the FP adder. How many (and what size) comparators are needed in order to determine when relevant values are being broadcast on the Common Data Bus? Why?

## 6. SuperScalar Microarchitectures (10 Points)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Register | Execute | Write | | | |
| Fetch | Transit | Map | Queue | Register | Address | Cache1 | Cache2 | Write | |
| | | | | Register | FP1 | FP2 | FP3 | FP4 | Write |

The Alpha 21264 processor is designed to issue 4 integer (2 of which may be load/store) and 2 FP instructions per cycle, and its pipeline diagram is shown above. (The shaded region indicates where instruction queueing/reordering may occur). All instructions execute in the first four stages, and then the pipeline is different for integer ops (top row), memory ops (second row), or floating point opts (third row).

a. (5 points) Since the Alpha instruction set is similar to MIPS (RISC, load/store, etc), how many register read and write ports would one expect to need, to avoid structural hazards, in a straightforward implementation of the integer register file?

b. (5 points) Since that number of ports is too difficult to implement, the chip designers used a trick instead. They divided the physical registers of the machine into two clusters. A group of functional units are associated with each clusters. Values written to registers in one cluster will eventually propagate to the other cluster, but there will be extra delay. In words explain how this implementation choice affects both the machine's instruction dispatch/issue unit as well as compiler strategies for this chip.