

**Problem 1: Short Answers (35 points)**

a) (2 points) Assuming that there are enough bits so that overflow is not a problem, how would you multiply a binary number by two?

- 1) Shift all bits one bit to the left
- 2) Shift all bits one bit to the right
- 3) Shift all bits two bits to the left
- 4) Shift all bits two bits to the right

b) (2 points) Single-bit multiplication is equivalent to which of the following gates? (2 points)

- 1) OR
- 2) AND
- 3) XOR
- 4) NAND

c) (5 points) There are three different types of data hazards, RAW, WAR, and WAW. Define them, giving a short code sequence to illustrate each, and describe how a 5-stage pipeline removes them.

d) (4 points) What are control hazards? Name and explain two different techniques for getting rid of them.

Name \_\_\_\_\_  
Student ID \_\_\_\_\_

Page 2

e) (4 points) Give a simple definition of precise interrupts/exceptions:

f) (4 points) Consider the following claim:

“Having a larger number of processor registers makes it possible to reduce the number of memory accesses needed to perform complex tasks”.

Devise a simple computational task to show the validity of this statement for a processor that has four registers as opposed to a processor that has only 2 registers.

g) (2 points) Convert the decimal numbers +26 and +19 into a *6-bit* binary two's-complement representation.

h) (2 points) Next, show the two's-complement representation of  $-26$  and  $-19$ .

i) (2 points) Show the two's-complement addition for  $(-26) + (-19)$ , and clearly mark your 6-bit result.

j) (2 points) Was there an overflow? Explain briefly.

k) (2 points) Which 6-bit two's-complement number *cannot* be negated correctly, and what is its decimal value?

l) (4 points) Define of synchronous and asynchronous busses. Give an advantage of using a synchronous bus. Give an advantage for using an asynchronous bus.

**Problem 2: Performance (20 points, 4 each)**

- a) Ryan and Gang decide to compare two computers which implement the x86 instruction set (the long standing computer instruction set family defined by Intel). To do this, they each take a chess playing program, written in C, and compile it on their respective machines and then calculate the instruction processing rate. Ryan's machine executes the program at 100 million instructions per second, and Gang's machine achieves 120 million instructions per second. Precisely state the relative performance result for these two systems, based on these measurements.
- b) Give one specific reason why this performance comparison might be flawed
- c) Now, Tom comes along and runs the same program on his computer, an Apple Power Macintosh, which uses the PowerPC processor. Tom's machine runs the program at 130 million instructions per second. How should we compare the performance of Tom's machine with Ryan and Gang's?

Name \_\_\_\_\_  
Student ID \_\_\_\_\_

Page 5

- d) Consider a multi-cycle implementation of the MIPS instruction set in which the mix across instruction classes is as shown below:

Instruction Dynamic Frequency (by count)

R-class (arithmetic/logic) 65%

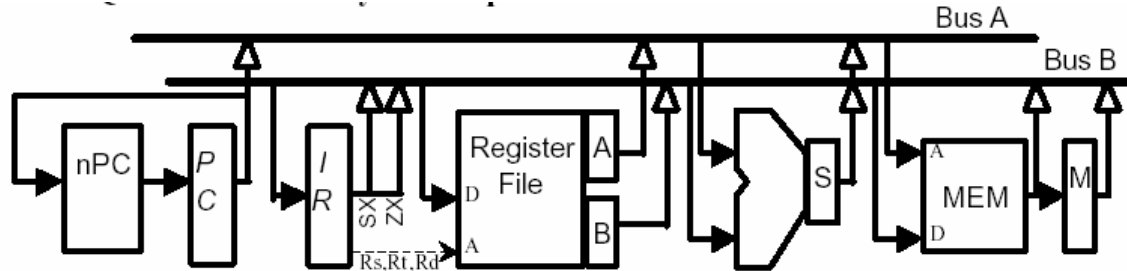
Memory (load/store) 15%

Branch (branch and jump) 20%

If R-class instructions on average take 4 cycles, memory instructions 6 cycles, and branch instructions 3 cycles, compute the CPI (cycles per instruction) for this machine.

- e) Based on Amdahl's law, what is the maximum performance improvement achievable for the system described in part (d) possible by improving the performance of branch instructions?

**Problem 3: Multicycle Datapath (40 points)**



The datapath above forms a multicycle processor which uses two time- multiplexed buses for communication rather than point-to-point connections and muxes. SX and ZX is the sign and zero extended immediate.

a) (20 points) For this datapath draw a FSM (with bubbles and arcs) for Fetch, Decode and the operations ADDI and LW.

b) (20 points) Fill out the microprogram table below to implement this FSM.  
**SrcA** and **SrcB** fields specify which signals will be assigned to BusA and BusB  
**WrtA** and **WrtB** fields specify what components are receiving inputs from the busses.  
 The **SrcX** and **WrtX** fields can be any one of the state registers (**IR, A, B, S, M**), the register file (**RegFile**), or memory (**Mem**).  
**Sequence** specifies the function of a jump counter (**next** for next instruction, **fetch** for go to 00 or **dispatch**).

| $\mu$ Address | Instruction | SrcA | SrcB | ALUOp | WrtA | WrtB | Sequence |
|---------------|-------------|------|------|-------|------|------|----------|
| 00            | Fetch       | PC   | Mem  |       |      | IR   | Next     |
| 01            | Decode      |      |      |       |      |      |          |
| 02            | ADDI        |      |      |       |      |      |          |
| 03            |             |      |      |       |      |      |          |
| 04            | LW          |      |      |       |      |      |          |
| 05            |             |      |      |       |      |      |          |
| 06            |             |      |      |       |      |      |          |
| 07            |             |      |      |       |      |      |          |
| 08            |             |      |      |       |      |      |          |

**Problem 4: Memory Hierarchy (50 points)**

- a) (5 points) Assume that we have a 32-bit processor (with 32-bit words) and that this processor is byte-addressed (i.e. addresses specify bytes). Suppose that it has a 512-byte cache that is two-way set-associative, has 4-word cache lines, and uses LRU replacement. Split the 32-bit address into “tag”, “index”, and “cache-line offset” pieces. Which address bits comprise each piece? Remember: 1 word = 4 bytes

**Tag:**

**Index:**

**Cache-line offset:**

- b) (5 points) How many sets does this cache have?
- c) (20 points) Draw a block diagram for this cache. Show a 32-bit address coming into the diagram and a 32-bit data result and “Hit” signal coming out. Include all of the comparators in the system and any muxes as well. Include the data storage memories (indexed by the “Index”), the tag matching logic, and any muxes. You can indicate RAM with a simple block, but make sure to label address widths and data widths. Make sure to label the function of various blocks and the width of any buses.

Name \_\_\_\_\_  
Student ID \_\_\_\_\_

This page was intentionally left black

d) (15 points) Below is a series of memory read references set to the cache from part (a). Assume that the cache is initially empty and classify each memory references as a hit or a miss. Identify each miss as either **compulsory, conflict, or capacity**. One example is shown. *Hint: start by splitting the address into components. Show your work.*

| Address | Hit/Miss? | Miss Type? |
|---------|-----------|------------|
| 0x300   | Miss      | Compulsory |
| 0x1BC   |           |            |
| 0x206   |           |            |
| 0x109   |           |            |
| 0x308   |           |            |
| 0x1A1   |           |            |
| 0x1B1   |           |            |
| 0x2AE   |           |            |
| 0x3B2   |           |            |
| 0x10C   |           |            |
| 0x205   |           |            |
| 0x301   |           |            |
| 0x3AE   |           |            |
| 0x1A8   |           |            |
| 0x3A1   |           |            |
| 0x1BA   |           |            |

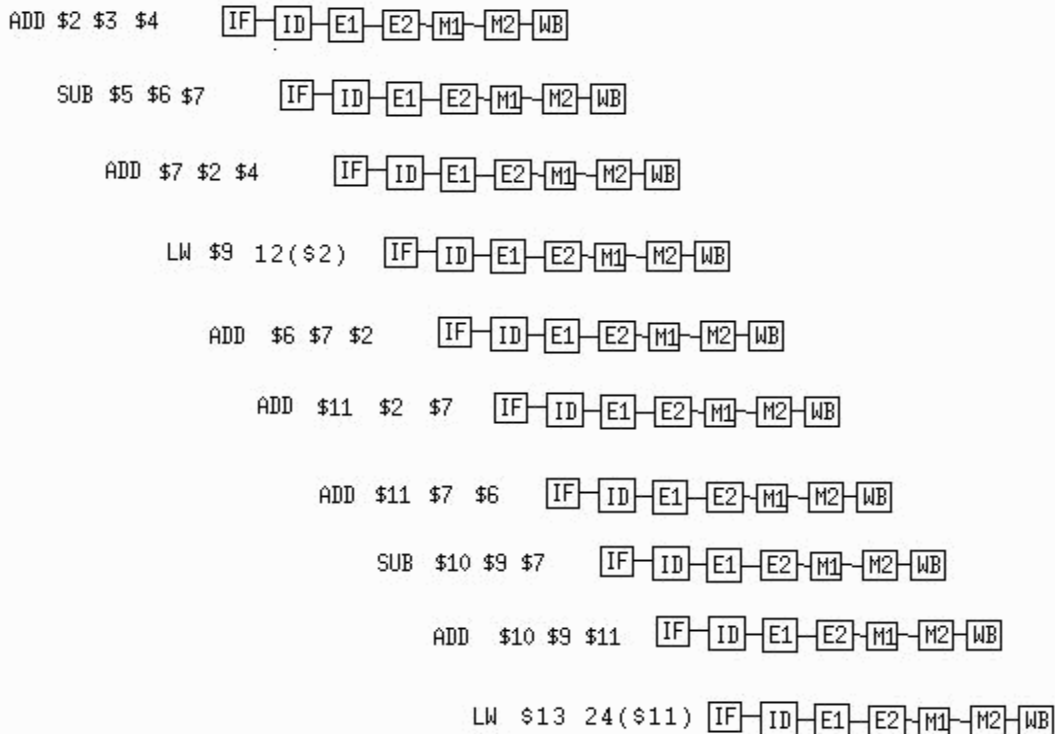
Name \_\_\_\_\_  
Student ID \_\_\_\_\_

e) (5 points) Calculate the miss rate and hit rate.

**Problem 5: Pipelining, Datapath Hazards, Forwarding (25 points)**

You work for some CPU company. The marketing department has decided that, even though the flagship CPU has the best performance on the market, it is difficult to sell because it has a low clock rate. The other engineers have decided that the best way to increase clock rate is to split the standard, MIPS-style 5-stage pipeline into a 7-stage pipeline. This will be accomplished by splitting the EXE stage into EXE1 and EXE2. Likewise for the MEM stage, there is now MEM1 and MEM2. Assume you cannot access any useful data in the position between the newly split stages, and you cannot forward into the middle of 2 EXE or MEM stages. The new pipeline, repeated a few times, is shown below.

This, of course, expands the number of forwarding paths. As a new designer, you get the job of designing the forwarding scheme. You decide to first do some test cases. a) (15 points) On the pipelines shown below, draw in the forwarding paths that are exercised in each cycle. The instruction shown to the left of each pipeline is the instruction that is issued (enters the IF stage) in the cycle where the IF stage resides in the picture, e.g. the first ADD is issued in cycle 1, the first SUB is issued in cycle 2, etc. Show only the forwarding paths that are used by the code shown, not all the possible forwarding paths in the processor. It is possible that stalls will be required to resolve the hazard, which you will enumerate in part b). Assume that the register file can do a write before a read in the same cycle.



b) (10 points) In the course of working through the above instructions, and the forwarding paths that are needed, you realized that there are still some cases where no amount of forwarding can resolve the dependencies, the data is simply not ready by the time the next instruction needs it. The only way to resolve the hazard is to insert a bubble into the pipeline, stalling the next instruction, which needs the data, while allowing the current instruction that is using the data, to continue processing. You have come up with some instruction sequences with these problems, and now you have to figure out how many bubbles need to be inserted in each case. One bubble is equivalent to stalling for one clock cycle. The cases of interest are shown in the table below. Fill in the total number of bubbles that need to be inserted for the block of code shown, in order to resolve all hazards.

| Instruction Sequence                                      | Total Number of Bubbles |
|---|-------------------------|
| ADD \$2, \$3, \$4<br>SUB \$7, \$2, \$3                    |                         |
| LW \$5, 24(\$6)<br>ADD \$9, \$8, \$5<br>SUB \$6, \$9, \$5 |                         |
| LW \$5, 12(\$13)<br>LW \$5, 16(\$8)                       |                         |
| LW \$4, 32(\$3)<br>LW \$5, 32(\$4)<br>LW \$6, 32(\$5)     |                         |
| ADD \$7, \$4, \$2<br>LW \$4, 0(\$7)                       |                         |
| SW \$7, 0(\$12)<br>XOR \$2, \$7, \$4                      |                         |