

### Question 1: Performance (25 points)

Consider the single-cycle and multi-cycle implementations discussed in lecture (for your reference the datapath for the single cycle is provided on the last page of this exam, the multi-cycle datapath and control are provided on pages 3 and 5, respectively). Given the functional unit latencies shown on the right, answer the following questions.

Func. Unit	Latency
Memory	4 ns
ALU	3 ns
Register File	2 ns

#### Part (a) (5 points)

What is the (minimum) cycle time for the single cycle implementation? Explain.

#### Part (b) (5 points)

What is the (minimum) cycle time for the multi-cycle implementation? Explain.

Below are two equivalent versions of C code for a loop that scales each element of a vector by a variable  $s$ . The version on the right uses array indexing, and the version on the left uses pointer arithmetic.

```
for (int i = 0 ; i < L ; ++ i)
    { D[i] = s * A[i] }
```

```
for (int i = 0 ; i < L ; ++ i)
    { *D = s * (*A); A++; D++; }
```

Below is MIPS assembly code that corresponds to the pointer arithmetic version:

```
loop:  lw      $t0, 0($a0)      # (*A)
      mul   $t0, $t0, $a3    # s * (*A)
      sw   $t0, 0($a1)      # *D = s * (*A)
      add  $a0, $a0, 4      # A ++
      add  $a1, $a1, 4      # D ++
      add  $t1, $t1, 1      # ++ I
      blt  $t1, $t2, loop   # branch to loop if (i<L)
```

#### Part (c) (5 points)

How many cycles would one iteration of this loop take to execute on the single cycle datapath?

#### Part (d) (10 points)

How many cycles would one iteration take on the the multicycle datapath? Assume the *mul* is as fast as an *add* and a *blt* is the same as a *beq*.

## Question 2: Multicycle Datapath (55 points)

Some ISA's have support for memory instructions with post-increment. These instructions perform both a load (or store) and an add to the address register. The value added to the address register is the size of the operand loaded/stored. For example:

```

lw+ $t0, 0($a0)    encodes    lw    $t0, 0($a0)
                                add    $a0, $a0, 4
and
sw+ $t0, 0($a1)    encodes    sw    $t0, 0($a1)
                                add    $a1, $a1, 4

```

These instructions are useful for loading/storing to values in arrays. For example, the code on the previous page could be reduced to:

```

loop:  lw+    $t0, 0($a0)        # (*A); A++
        mul   $t0, $t0, $a3     # s * (*A)
        sw+   $t0, 0($a1)       # *D = s * (*A); D ++
        add   $t1, $t1, 1       # ++ I
        blt   $t1, $t2, loop    # branch to loop if (i<L)

```

The lw+ and sw+ instructions have the same format i-type as lw and sw, shown below. Recall that in a load the rt field designates the destination register for the load.

```

lw+ rt, offset(rs) # R[rt] = Memory[R[rs]+offset]; R[rs] += 4;
sw+ rt, offset(rs) # Memory[R[rs]+offset] = R[rt]; R[rs] += 4;

```

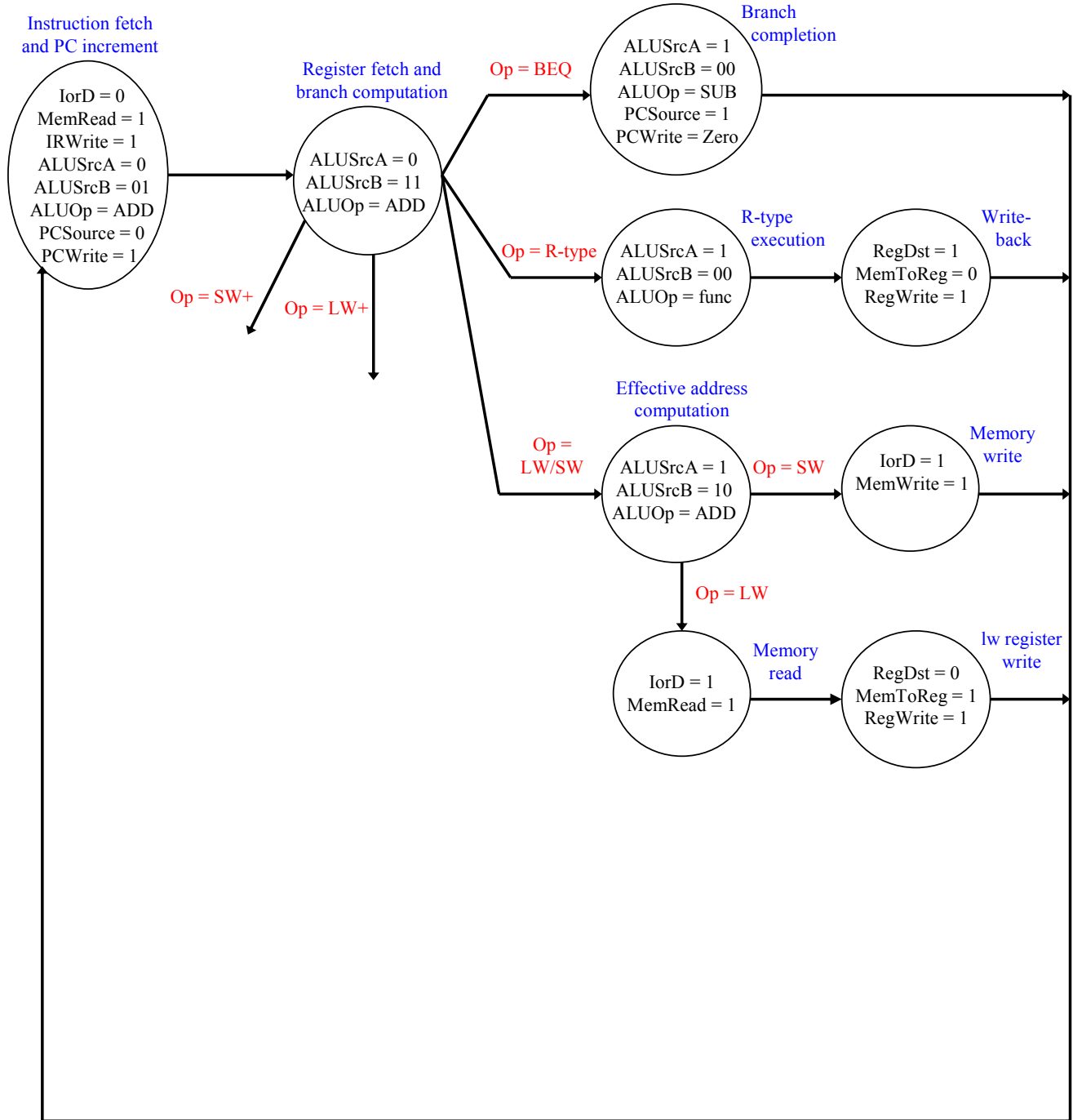
Field	op	rs	rt	offset
Bits	31-26	25-21	20-16	15-0



**This page was intentionally left blank.**

**Part (b) (25 points)**

Complete this finite state machine diagram for the **lw+** and **sw+** instruction. Control values not shown in each stage are assumed to be 0. Remember to account for any control signals that you added or modified in the previous part of the question!



**Part (c) (10 points)**

What is the average CPI of the new version of code on your modified datapath. You can leave your result as an expression. State any assumptions.

**Part (d) (10 points)**

A fellow engineer proposes spending two cycles for each memory access, to allow the clock period to be shortened. This would add 2 cycles (one for each memory access) to the execution of each load and store and one cycle to each non-memory instruction. Would this improve the execution time of the second version of the loop (found on page 1)? Show your work.

### Question 3: Arithmetic (20 points)

The original reason for Booth's algorithm was to reduce the number of operations by avoiding operations when there were string of 0s and 1s. Revise the algorithm presented in class to look at 3 bits at a time and compute the product two bits at a time.

Recall the table for 2 bits (A is the multiplicand, B is the multiplier):

$A_i$	$A_{i-1}$	Operation
0	0	Do nothing
0	1	Add B
1	0	Subtract B
1	1	Do nothing

Fill in the following table to determine the 2-bit Booth encoding. Assume that you have both the multiplicand and 2 x the multiplicand already in registers. Explain the reason for the operation on each line. Two of the cases are given for you.

Current Bits		Previous Bit	Operation	Reason
$A_{i+1}$	$A_i$	$A_{i-1}$		
0	0	0	None	Middle of a string of 0s
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1	None	Middle of a string of 1s

**Question 4: Caches (50 points)**

**Part (a) (20 points)** You are given a 16-byte cache (initially empty) and the sequence of memory accesses (byte loads) shown in the table. For each of the following cache configurations explain whether it can produce this sequence. If so, what is the result of the last access (shown as ???) for each of the matching sequences?

address	hit/miss
0	miss
12	miss
1	hit
17	miss
15	hit
3	miss
7	???

1. 4-byte blocks, 2-way set-associative

2. 4-byte blocks, direct-mapped

3. 8-byte blocks, direct-mapped

4. 4-byte blocks, fully associative

**Part (b) (30 points)**

$$\text{AMAT} = \text{Hit time} + (\text{Miss rate} \cdot \text{Miss penalty})$$
$$\text{Memory stall cycles} = \text{Memory accesses} \cdot \text{miss rate} \cdot \text{miss penalty}$$

Assume you have a processor with a 16KB, 4-way set-associative (i.e., each set consists of 4 blocks) data cache with 32-byte blocks. Here, a “KB” is  $2^{10}$  bytes.

**Part 1 (5 points)**

How many total blocks are in the cache? How many sets are there?

**Part 2 (5 points)**

Assuming that memory is byte addressable and addresses are 35-bits long, give the number of bits required for each of the following fields:

Cache Tag	
Index	
Byte Select	

**Part 3 (5 points)**

What is the total size of the cache, including the valid, tag and data fields? Give an exact answer, in either bits or bytes.

Assume that the cache communicates with main memory via a 64-bit bus that can perform one transfer every 10 cycles. Main memory itself is 64-bits wide and has a 10-cycle access time. Memory accesses and bus transfers may be overlapped.

**Part 4 (5 points)**

What is the miss penalty for the cache? In other words, how long does it take to send a request to main memory and to receive an entire cache block?

**Part 5 (5 points)**

If the cache has a 95% hit rate and a one-cycle hit time, what is the average memory access time?

**Part 6 (5 points)**

If we run a program which consists of 30% load/store instructions, what is the average number of memory stall cycles per instruction?

## Question 5: Pipelining, Dependencies and Forwarding (35 points)

### Part (a) (5 points)

Show or list all of the dependencies in the following code fragment. Make sure that you clearly indicate which instructions *and* registers are involved in each dependency.

add \$8, \$5, \$5

sub \$8, \$8, \$10

lw \$6, 4(\$8)

add \$4, \$6, \$8

### Part (b) (30 points; 2 points each)

The pipelined datapath on the next page shows the fifth cycle of executing this program. Fill in the correct datapath values for the *fifteen* question mark symbols ? in the diagram. There is one ? in the IF stage, three in the ID stage, eight in EX, two in MEM, and one in WB.

- Assume that registers initially contain their number plus 100: \$6 contains 106, \$8 contains 108, etc.
- Write your values directly on the diagram, but please write clearly.
- Use decimal notation. You may write 'X' for any values that cannot be determined.



## Miscellaneous Information

### Performance

1. Formula for computing the CPU time of a program P running on a machine X:

$$CPU\ time_{X,P} = \text{Number of instructions executed}_{P,X} \times CPI_{X,P} \times \text{Clock cycle time}_X$$

2. CPI is the average number of clock cycles per instruction:

$$CPI = \text{Number of cycles needed} / \text{Number of instructions executed}$$

3. Speedup is a metric for relative performance of 2 executions:

$$\begin{aligned} \text{Speedup} &= \text{Performance after improvement} / \text{Performance before improvement} \\ &= \text{Execution time before improvement} / \text{Execution time after improvement} \end{aligned}$$

### Single Cycle Datapath

