COMP 230: Computer Architecture and Organization **December 03, 2018**

Exam 3 (Practice)

Instructions:

- **This practice exam is meant to give you the flavor of questions that will be asked on the exam. Do not expect the real exam to be the same questions with only the numbers or MIPS instructions changed.**
- The final exam is cumulative, so there are too many topics to be represented on any reasonably-sized practice exam. Expect that there will be topics on the final exam that are barely covered or not covered at all in this practice exam. For example, virtual memory is only briefly covered on this practice exam...
- Your real exam will be open book and notes. You are allowed to use calculators but not laptops, cell phones, or other electronics.
- If you do not show your work, do not expect partial credit for incorrect answers.
- If you believe a problem is incorrectly or incompletely specified, make a reasonable assumption and solve the problem. The assumption should not result in a trivial solution.
- In all cases, clearly state any assumptions you make in your answers.

Name	

Part	Description	**Points Possible**	**Grade**
1	Multiple choice	15	
2	Short Answer	25	
3	Arithmetic	20	
4	Pipelines	20	
5	Memory Hierarchy	40	
6	Parallelism	10	
Total		130	

1 Multiple Choice (15 points)

1. [3 points] Processor A and Processor B have the same ISA and clock speed. What additional metric(s) can help you decide which processor is faster?

   (a) instructions per second

   (b) IPC

   (c) Either A or B

   (d) None of the above

   Your Answer C

2. [3 points] Processor A has twice the clock speed (half the clock period) of processor B; they both have the same ISA and compiler. What information do you need to decide which processor is faster?

   (a) The number of instructions executed (*dynamic* instruction count)

   (b) IPC

   (c) Both A and B

   (d) We already have enough information to decide.

   Your Answer B

3. [3 points] Which of the following techniques would *not* reduce the number of conflict misses your cache experiences?

   (a) Increasing the block size (keeping capacity fixed)

   (b) Increasing the cache capacity

   (c) Increasing the cache associativity

   (d) All of the above reduce conflict misses

   Your Answer A

4. [3 points] Which of the following does a cache designer *not* have control over?

   (a) The number of bits in the offset

   (b) The number of bits in the page offset

   (c) The number of bits in the index

   (d) The associativity of the cache

   Your Answer B

5. [3 points] x86 is a very complicated ISA that is quite difficult to implement efficiently. How does Intel reduce this complexity in their implementations?

   (a) They use a really long pipeline with no forwarding

   (b) They decrease the clock speed

   (c) They don't; they just hire really good chip designers and pay them a lot of money

   (d) During execution they translate complex instructions into simpler instructions

   Your Answer D

2 Short Answer (25 points)

6. [5 points] You are told that processor A has a CPI of 1.5 when running program 1. When you run program 2 on processor A, the CPI is 2.0. What might account for the difference?

Program 2 must have a different instruction mix that is harder for processor A to execute. There are many examples that can cause this — more dependencies between instructions (data hazards), worse memory access pattern (cache misses), or use of more multi-cycle instructions (e.g. floating point).

7. [5 points] Do TLB misses and page faults occur independently of one another? Explain.

No, a TLB miss does not necessarily mean a page fault. But if you had a page fault, you must have also had a TLB miss. This is because if the page is not in memory it must not have been accessed in a while. But the TLB is much smaller than the page table, so the entry must also not be in the TLB.

8. [5 points] With regard to multiple-issue processors, give one reason why static scheduling is better than dynamic scheduling. *Only your first answer will be graded.*

Simpler hardware.

9. [5 points] The MIPS ISA is relatively simple to implement with a pipeline by design — all ALU operations work only on register operands. Give one way in which our typical

$$\text{Fetch} \rightarrow \text{Decode} \rightarrow \text{Execute} \rightarrow \text{Memory} \rightarrow \text{Write-back}$$

pipeline must change if want to perform ALU operations directly on memory operands.

The current pipeline cannot operate directly on memory because we only access memory *after* we've already used the ALU. So we'll need to swap the Execute and Memory stages to make this work. Though this complicates many other instructions which rely on having Execute before Memory...

10. [5 points] Give two reasons to use virtual memory.

Stops concurrent processes from accidentally or maliciously interfering with each other by changing memory.

Gives each process it's own private view of memory, which makes things like linking and loading simpler.

Let's us avoid restrictions on the amount of memory on the machine — we can just page to disk (or SSD) when we need to (though this is incredibly slow when we have to page).

3 Arithmetic (20 points)

11. [10 points] Write MIPS assembly code to for a procedure `vmult` that takes in three pointers (to arrays of integers, all the same size) and an integer (the size of the arrays). This procedure should multiply each element of the first array by the corresponding element of the second array, and put the result into the corresponding element in the third array. In other words, if A, B, and C represent our arrays, our procedure sets $C[i] = A[i] * B[i]$ for all valid i.

    This procedure should return an integer. If any of the multiplications overflow a 32-bit register, the procedure should immediately return 0. Otherwise, return 1.

    Here is one possible solution:

```
vmult:    add   $t0,  $zero,  $zero                # initialize i
loop:     beq   $t0,  $a3,    success
          sll   $t1,  $t0,    2       # bytes past start of arrays
          add   $t2,  $a0,    $t1       # prepare address of A[i]
          lw    $t2,  0($t2)
          add   $t3,  $a1,    $t1                # address of B[i]
          lw    $t3,  0($t3)
          mult  $t2,  $t3
          mfhi  $v0
          bne   $v0,  $zero,  failure
          add   $t4,  $a2,    $t1                # address of C[i]
          mflo  $v0
          sw    $v0,  0($t4)
          addi  $t0,  $t0,    1
          j     loop
failure:  add   $v0,  $zero,  $zero
          jr    $ra
success:  addi  $v0,  $zero,  1
          jr    $ra
```

12. Consider an 8-bit *fixed* point representation for fractional numbers. The first 3 bits represent the (signed) integer part, while the last 5 bits represent the fractional component.

    (a) [3 points] How many different objects can be represented?

    $2^8 = 256$

    (b) [3 points] How many different *real* numbers are represented in this format?

    Unlike floating point, all the objects we represent are real numbers, so again it's $2^8 = 256$.

    (c) [4 points] What is the smallest positive (and non-zero) number that can be represented? Give the number as a base-10 fraction.

    It will be $000.00001_2 = \frac{1}{32}$.

4 Pipelines (20 points)

13. [10 points] Figure 1 shows the datapath for our MIPS pipeline, minus the instruction fetch stage. I've shown some forwarding wires in color, allowing us to remove some stalls. Below each stage is an instruction currently in progress in that stage.

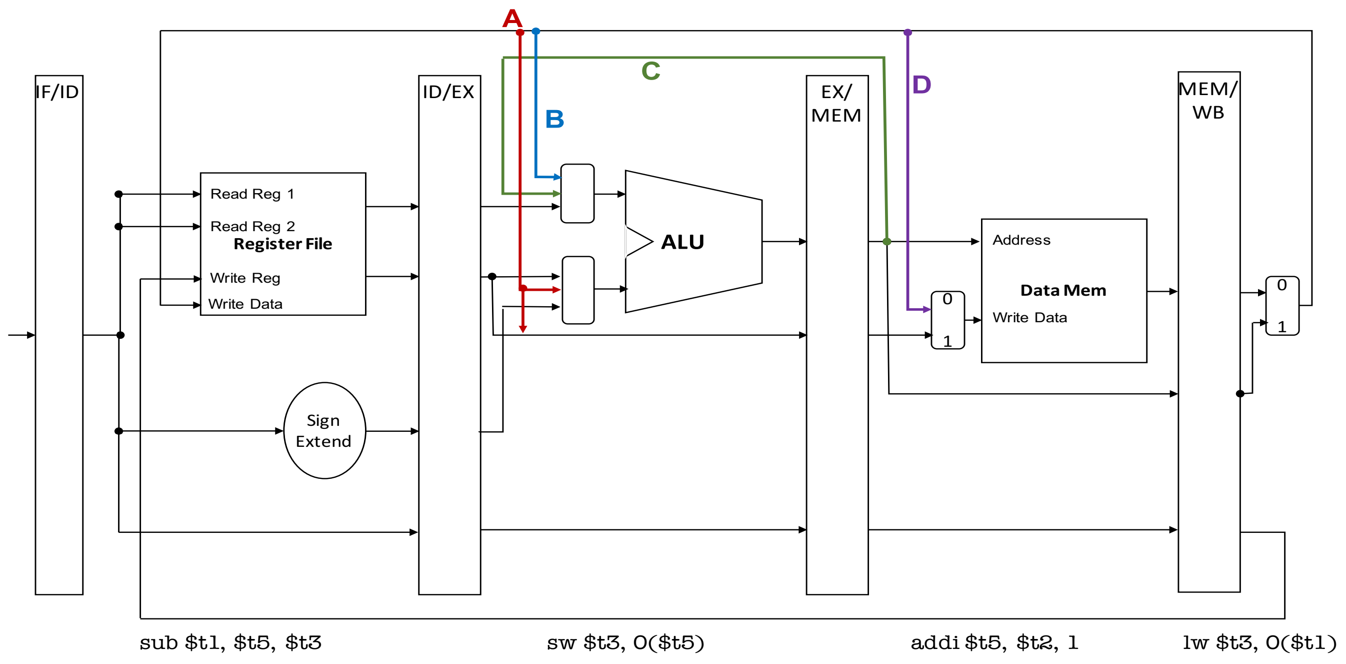

Figure 1: MIPS pipeline stages with forwarding.

For each input specified below, state where the instruction gets its data — one of the forwarding lines (write the letter of the line shown in figure 1), or the register file (write RF). Assume that the in-progress instructions did not require forwarding of results from earlier instructions (i.e., instructions before the `lw`).

(i) `$t2` of the `addi` instruction RF

(ii) `$t3` of the `sw` instruction A

(iii) `$t5` of the `sw` instruction C

(iv) `$t5` of the `sub` instruction B

(v) `$t3` of the `sub` instruction RF

14. [10 points] Give a series of instructions that would require a pipeline stall, i.e., the forwarding shown above would not help resolve the dependency.

Any load-use dependency works here, since the result is needed at the ALU, but the previous instruction is still loading the value from memory at this point. For example:

```
lw    $t0,   4($s0)
add   $t2,    $t0,   $t1
```

5 Memory Hierarchy (40 points)

15. [13 points] Complete the following trace of a 2-way set associative cache with 2B cache lines and an LRU replacement policy. The LRU column stores which way was least recently accessed.

Pay Attention: For an access on row *n*, first state *on row n* whether the access results in a hit or miss. If a miss occurs, say which type (capacity, conflict, or cold). Then, if the access changes the contents of the cache, show that change *on row n+1*. Update only the fields in the cache that change.

The accesses *before* the question started were the addresses of the data: N, U, G, L, in that order.

Set 0					Set 1					Cache access			
way 0			way 1		way 0			way 1					
tag	data	LRU	tag	data	tag	data	LRU	tag	data	addr	data	H or M	M type
011	M N	0	101	U V	010	K L	1	001	G H	00001	B	M	cold
000	A B	1								10111	X	M	cold
							0	101	W X	00111	H	M	conflict
					001	G H	1			01101	N	M	capacity

Data in memory:

address	data
00000	A
00001	B
00010	C
00011	D
00100	E
00101	F
00110	G
00111	H
01000	I
01001	J
01010	K
01011	L
01100	M
01101	N
01110	O
01111	P
10000	Q
10001	R
10010	S
10011	T
10100	U
10101	V
10110	W
10111	X
11000	Y
11001	Z

16. [12 points] The following table gives the parameters for a number of different caches. For each cache, fill in the missing fields in the table. Let m be the number of physical address bits, C be the total number of bytes in the cache, B be the block size (in bytes), E be the associativity, S be the number of cache sets, t be the number of tag bits, s be the number of set index bits, and b the number of block offset bits.

m	C	B	E	S	t	s	b
32	1024	4	4	64	24	6	2
32	1024	4	256	1	30	0	2
32	1024	8	1	128	22	7	3
32	1024	8	128	1	29	0	3
32	1024	32	1	32	22	5	5
32	1024	32	4	8	24	3	5

17. Consider a web server that records a log of each access to a particular web page. Each entry uses a structure as shown on the left of figure 2, and each entry is later processed with the function described on the right of figure 2.

```
struct entry {
  int srcIP; // remote IP address
  char URL[60]; // request URL (e.g.
      "home.html")
  int refTime; // time of request
  char browser[60]; // client browser name
};
entry log[NUM_ENTRIES];
```

```
/* Looks through the entries in log,
 * considering only the accesses to
 * the server before time.
 * Puts srcIP of the last
 * 10 into a new array and returns it.
 **/
int* latest10_hits(entry *log, int time) {
  ...
}
```

Figure 2: Web server code.

(a) [5 points] Given a cache with 64-byte blocks, how many cache misses does `latest10_hits` incur for each entry, in the worst case? Assume each entry is accessed exactly once.

Two, since our block size is only 64 bytes but each log entry requires 128 bytes. When accessing `srcIP`, the entire block is pulled in, which contains `URL` even though we don't care about it. Then when we access `refTime` we'll see another cache miss.

(b) [10 points] How can you reorganize the `entry` data structure to reduce cache misses? Show your new structure definition code. (You do NOT need to calculate the number of cache misses with this change.)

Simply rearranging the entries works:

```
struct entry {
  int srcIP;
  int refTime;
  char URL[60];
  char browser[60];
};
```

Now when `latest10_hits` looks at `log[0].srcIP` it will incur a cache miss, which will bring in the whole block. But since `refTime` is right next to it, it is included in the block, hence an access to `log[0].refTime` is a hit. Only part of `URL` fits in this block ($64 - 4 - 4 = 56$ bytes of it); the rest of it will be in the next block of memory. But `latest10_hits` does not look at or care about `URL`, so we'll never access this; it's never loaded into cache. The same goes for `browser`.

6 Parallelism (10 points)

18. [10 points] Your coworker has developed a new (sequential) algorithm for matrix multiplication, but it is not fast enough. You are able to parallelize $\frac{3}{4}$ of its execution time, independent of the size of the input matrices. How many cores do you need to achieve a 3x speedup over a single-core execution?

From our application of Amdahl's Law to parallel speedup, we know that

$$\text{Speedup} = \frac{1}{(1 - F_{\text{par}}) + \frac{F_{\text{par}}}{P}}$$

So we have

$$3 = \frac{1}{(1 - \frac{3}{4}) + \frac{3}{4P}}$$

$$\implies \frac{1}{4} + \frac{3}{4P} = \frac{1}{3}$$

$$\implies \frac{3}{4} = \frac{P}{12}$$

$$\implies P = \boxed{9}$$