

SOLUTIONS

CS 152 Computer Architecture and Engineering

CS 252 Graduate Computer Architecture

Midterm #2

April 11th, 2018

Professor Krste Asanovic

Name: _____

I am taking CS152 / CS252

This is a closed book, closed notes exam.

80 Minutes. 20 pages.

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Please write your name on every page in the exam.
- You must not discuss an exam's contents with other students who have not taken the exam. If you have inadvertently been exposed to an exam prior to taking it, you must tell the instructor or TA.
- You will receive no credit for selecting multiple-choice answers without giving explanations if the instructions ask you to explain your choice.

	CS152	CS252	Your Points
Question 1	25 points	30 points	
Question 2	15 points	15 points	
Question 3	20 points	20 points	
Question 4	15 points	15 points	
Question 5	15 points	15 points	
Question 6	10 points	15 points	
Total	100 points	110 points	

Question 1: Out-of-Order Execution [25 + 5 points]

Question 1.A (10 points)

For this question, we want to schedule the following code on an out-of-order core.

```
fld f0, 0(x1)
fld f1, 8(x1)
fmul.d f0, f0, f1
fadd.d f2, f2, f0
fld f0, 16(x1)
fadd.d f2, f2, f0
```

The processor is a single-issue core with a data-in-ROB design. The ROB has four entries. Instructions can commit one cycle after writeback, and ROB entries can be reused one cycle after commit. Instructions that depend on others can issue one cycle after the instruction it depends on writes back. Loads and stores take two cycles each, floating-point multiplies take three cycles, and floating-point adds take five cycles. All functional units are fully pipelined.

Fill out the table with the cycles at which instructions enter the ROB, issue to the functional units, write back to the ROB, and commit. Also fill out the new register names for each instruction. Use r0-r3 for the four ROB entries. If the instruction producing a source register had already committed before the dependent instruction enters the ROB, use the architectural register name.

Remember that instructions must enter the ROB and commit in order. On each cycle, only one instruction can enter the ROB, one can issue, one can write back, and one can commit.

	Time				Instruction			
	Enter ROB	Issue	WB	Commit	OP	Dest	Src1	Src2
I ₁	-1	0	2	3	FLD	r0	x1	
I ₂	0	1	3	4	FLD	r1	x1	
I ₃	1				FMUL.D			
I ₄					FADD.D			
I ₅					FLD			
I ₆					FADD.D			

Name: _____

Question 1.B (15 points)

We execute the following program on an out-of-order core with a unified physical register file. The “away” target of the branch is in some unrelated part of the code. The branch predictor initially predicts that the branch is not taken. But after all six instructions complete, the branch resolves to taken and the store word has an exception. Show the state of the pipeline after all the mispredicts and exceptions are handled. That is, after precise architectural state has been restored and the correct branch target is about to be decoded. Assume that mispredicts and exceptions use the same rollback procedure and that the free list is in FIFO order. We have already completed the first instruction for you.

lw x2, 0(x1)	
add x3, x2, x5	
beq x2, x6, away	mispredict
addi x1, x1, 8	
sw x3, 0(x1)	exception
addi x1, x1, 8	

Name: _____

Rename Table		Physical Register File			Free List
x0		p0			p5
x1	p3	p1	<x5>	p	p7
x2	p4 → p5	p2	<x6>	p	p8
x3	p6	p3	<x1>	p	p9
x4		p4	<x2>		p0
x5	p1	p5	<x2>	p	p4
x6	p2	p6	<x3>		
x7		p7			
x8		p8			
x9		p9			
...		...			

ROB										
valid	complete	exception	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
	c		lw	p	p3			x2	p4	p5

Name: _____

Question 1.C (252 only, 5 points)

In the previous question, mispredictions and exceptions used the same procedure for restoring precise state. Is this an acceptable design for a high-performance core? Why or why not? If not, what is an alternative way of handling mispredictions?

Question 2: Branch Prediction [20 points]

For the following question, we are interested in the performance of branches when executing the following code. For each part, assume that $N = 4$ and the array A has the values $\{1, 7, 2, 5\}$.

C code	RISC-V Assembly
<pre> for (int i = 0; i < N; i++) { int b = A[i]; if (b >= 5) c += b; if (b < 4) c -= b; } </pre>	<pre> li x1, A li x4, N loop: lw x2, 0(x1) li x5, 5 blt x2, x5, skip1 add x3, x3, x2 skip1: li x5, 4 bge x2, x5, skip2 sub x3, x3, x2 skip2: addi x1, x1, 4 addi x4, x4, -1 bnez x4, loop </pre>

Question 2.A (CS152 Only, 5 points)

Fill out the table to show what the predictions will be if the branch predictor is a BHT indexed by PC with two-bit counters. If the most-significant bit of a counter is 1, the predictor predicts taken. Otherwise it predicts not taken. The counters are initialized to weakly not-taken (01). The Counter column in the table shows the state of the counter before the branch is executed. Assume that the BHT is large enough that no aliasing of instruction addresses will occur.

	Instruction	Counter	Prediction	Actual
i=0	blt x2 x5, skip1	01	not taken	taken
	bge x2, x5, skip2	01	not taken	not taken
	bnez x4, loop	01	not taken	taken
i=1	blt x2 x5, skip1			not taken
	bge x2, x5, skip2			taken
	bnez x4, loop			taken
i=2	blt x2 x5, skip1			taken
	bge x2, x5, skip2			not taken
	bnez x4, loop			taken
i=3	blt x2 x5, skip1			not taken
	bge x2, x5, skip2			taken
	bnez x4, loop			not taken

What is the prediction accuracy for each branch? What is the prediction accuracy overall?

blt:

bge:

bnez:

overall:

Question 2.B (5 points)

Now assume we change the branch predictor to a BHT indexed by PC and a single bit of global history. Assume the global history is initialized to 0, the counters are initialized to 01 (weakly not-taken), and there is no aliasing. The Counter columns in the table show the state of the counters before the branch is executed. Fill out the table with the predictions. (3 points for table, 2 points for accuracy)

	Instruction	Global History	Counter 0	Counter 1	Prediction	Actual
i=0	blt x2 x5, skip1	0	01	01	not taken	taken
	bge x2, x5, skip2	1	01	01	not taken	not taken
	bnez x4, loop	0	01	01	not taken	taken
i=1	blt x2 x5, skip1					not taken
	bge x2, x5, skip2					taken
	bnez x4, loop					taken
i=2	blt x2 x5, skip1					taken
	bge x2, x5, skip2					not taken
	bnez x4, loop					taken
i=3	blt x2 x5, skip1					not taken
	bge x2, x5, skip2					taken
	bnez x4, loop					not taken

What is the prediction accuracy for each branch? What is the prediction accuracy overall?

blt:

bge:

bne:

Overall:

Name: _____

Question 2.C (5 points)

If you run this code with a large array containing uniformly randomly distributed values, which branch do you expect to get the most benefit from global history and why?

Question 2.D (CS 252 Only, 5 points)

Explain the motivation for using both BHT and BTB branch-prediction structures in the same implementation.

Question 3: Load/Store Units [20 points]

Question 3.A (10 points)

Table 3.1 shows the current state of the store queue in an out-of-order processor. The instruction number indicates the order of instructions in the program, with lower numbers being earlier in program order. Table 3.2 shows the values stored in the data cache.

Assume that all stores and loads are for the full 32-bit word and aligned to 32 bits. The processor uses conservative out-of-order load/store execution. For each of the following loads, can the load be completed under this model? If so, what value does it read? Fill out Table 3.3 with your answers.

Instruction Number	Address	Value
3	0x1000	0xF00D3ABC
6	0x2000	Unknown
11	Unknown	Unknown
15	0x1000	0xDEADBEEF
17	Unknown	Unknown

Table 3.1 Store Queue

Address	Value
0x1000	0xAACCBDAF
0x2000	0xBADE2140
0x3000	0x1234ABCD

Table 3.2 Data Cache

Name: _____

Instruction Number	Address	Can execute?	Value
4	0x2000		
5	0x1000		
8	0x2000		
13	0x1000		
16	0x1000		
18	0x3000		

Table 3.3 Load Queue

Question 3.B (5 points)

Now assume that the processor has address speculation and assumes that unknown addresses in the store queue will be different from addresses of pending loads. Which of the loads that couldn't be executed in Question 3.A can now be speculatively issued? Give the instruction numbers. What are their speculative values?

Question 3.C (5 points)

After the loads are speculatively issued, store 11 turns out to have address 0x3000 and store 17 turns out to have address 0x1000. Which speculative loads were mistakenly issued in Question 3.B? How do we recover from mis-speculation?

Question 4: VLIW Machines [15 points]

In this problem, we consider the execution of a code segment on a VLIW processor. The code we consider is the IMAX kernel, which finds the maximum value and its index in the list.

```
for (i = 0 ; i < N ; i++) {
  if (max < l[i]) {
    idx = i;
    max = l[i];
  }
}
```

```
# t0: i, s0: idx, f0: max, a0: N, a1: pointer of l[i]
loop:  fld f1, 0(a1)          # load l[i]
      flt.d t1, f0, f1      # set if max < l[i]
      fmax.d f0, f0, f1     # max = max < l[i] ? l[i] : max
      beqz t1, skip        # if max >= l[i], jump to skip
      addi s0, t0, 0       # update idx
skip:  addi a1, a1, 8        # bump l
      addi t0, t0, 1       # increment i
      bltu t0, a0, loop    # loop
```

Now we have a VLIW machine with five execution units:

- two ALU units, latency one cycle, also used for branch operations.
- one memory unit, latency two cycles, fully pipelined, each unit can perform either a store or a load.
- two FPU units, latency three cycles, fully pipelined, both can perform `flt.d` and `fmax.d`.

Assume there are no exceptions during the execution.

Question 5: Vector Machines [15 points]

A. *(CS152 Only, 15 points)* In this problem, we will vectorize the following code with the RISC-V Vector ISA:

```
double A[N+20], B[2*N];
...
for (i=0; i<N; i++) {
    A[i] = A[i+20] + B[2*i]*B[2*i+1];
}
```

Fill out the blanks below (the code spans to the next page).

```
# a0: N, a1: A pointer, a2: B pointer
# v0, v2-v7: 64-bit float vector
# v1: 8 bit int vector for mask
```

```
loop: _____ # set VL for loop, t0 = VL
```

```
# load A[i+20]
```

```
_____
_____
_____
```

```
# load B[2*i], load B[2*i+1]
```

```
_____
_____
_____
_____
```

```
# Compute A[i+20] + B[2*i]*B[2*i+1]
```

```
_____
_____
_____
_____
```

Name: _____

Store A[i]

```
sll t2, t0, 3
add a1, a1, t2          # bump A
sll t3, t0, 4
add a2, a2, t3          # bump B
sub a0, a0, t0          # decrement N
bnez a0, loop           # loop
```


Name: _____

B. (CS 252 Only) We have a vector machine where $\text{MAXVL} = 64$. There are three vector functional units (a multiply unit, an add unit, and a load/store unit) each with 8 lanes and each unit (multiply, add, and load/store) is fully pipelined with a 6-cycle latency.

i) **(5 points)** What is the minimum vector instruction bandwidth (vector instructions issued per clock cycle) to keep all the functional units busy?

ii) **(5 points)** What if the MAXVL was 16?

iii) **(5 points)** How does the minimum vector instruction bandwidth change if the pipeline latency increases to 8 cycles?

Question 6: Multithreading [10 + 5 points]

In this question, we consider a program that computes a running average across a long stream.

C code	Assembly
<pre>float *stream = ...; float *stream_end = ...; float avg = 0.0f; float total = 0.0, n = 0.0; while (stream != stream_end) { total += *stream; n += 1.0; avg = total / n; stream++; }</pre>	<pre>// x1 = stream, x2 = stream_end // f0 = avg, f1 = total // f2 = n, f3 = 1.0 loop: flw f4, 0(x1) fadd.s f2, f2, f3 fadd.s f1, f1, f4 fdiv.s f0, f1, f2 addi x1, x1, 4 bne x1, x2, loop</pre>

We run this on a multithreaded in-order core with no data cache, perfect branch prediction, and no threading overhead. The latencies of each type of instruction are as follows.

Load/store	16 cycles
Float to integer conversion	2 cycles
Floating-point addition	5 cycles
Floating-point multiply	3 cycles
Floating-point division	7 cycles
Integer operations	1 cycle

Name: _____

Question 6.A (2 points)

How many threads do we need to run without pipeline stalls if the processor switches to another thread every cycle using fixed round-robin scheduling?
Please show your work.

Question 6.B (4 points)

How many threads do we need to run without stalling if the processor only switches threads when the next instruction will stall due to a data dependency?
Show your work.

Name: _____

Question 6.C (2 points)

In the case where we switch threads every cycle, can we reduce the number of threads needed to run without pipeline stalls by reordering instructions? How can we do this and what is the new number of threads?

Question 6.D (2 points)

In the case where we switch only if there is an unmet data dependency, can we reduce the number of threads needed to run without stalls by reordering instructions? How can we reorder the instructions and what is the new number of threads?

Name: _____

Question 6.E (CS 252 Only, 5 points)

For each following resources, indicate whether or not they are shared in an SMT processor.

Program Counter	
Fetch Unit	
Rename Table	
Physical Register File	
Issue Window	
Functional Units	
ROB	