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	

Name: _____

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	4	7	8	FMUL.D	r2	r0	r1
I_4	2	8	13	14	FADD.D	r3	f2	r2
I_5	4	5	8	15	FLD	r0	x1	
I_6	5	14	19	20	FADD.D	r1	r3	r0

-1.5 if cycle time row incorrect
-1 if register row incorrect

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 → p8 → p9 → p8 → p3	p1	<x5>	p	p7 x
x2	p4 → p5	p2	<x6>	p	p8 x
x3	p6 → p7	p3	<x1>	p	p9 x
x4		p4	<x2>	~~p~~	p0
x5	p1	p5	<x2>	p	p4
x6	p2	p6	<x3>	~~p~~	p6
x7		p7	<x3>	p	p9
x8		p8	<x1>	~~p~~	p8
x9		p9	<x1>	~~p~~	
…		…			

ROB										
valid	complete	exception	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
~~v~~	c		lw	p	p3			x2	p4	p5
~~v~~	c		add	p	p5	p	p1	x3	p6	p7
~~v~~	c		beq	p	p5	p	p2			
~~v~~	c		addi	p	p3			x1	p3	p8
~~v~~	c	x	sw	p	p7	p	p8			
~~v~~	c		addi	p	p8			x1	p8	p9

-1 for each incorrect rename table entry
-0.25 for each incorrect phys reg file entry or markout
-1 for each incorrect free list markout or addition
-0.5 if valid not marked out in first ROB row
-1 for each incorrect ROB row (other than first)

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?

This is generally not a good design for a high-performance core, because the exception rollback procedure is quite expensive. It's fine for exceptions, since they don't happen very frequently, but branch mispredictions are more common, so going through the full rollback procedure can be quite expensive. Most OoO cores save snapshots for each branch so that the state can be restored quickly on mispredict.

Name: _____

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

`for (int i = 0; i < N; i++) {`
`  int b = A[i];`
`  if (b >= 5)`
`  c += b;`
`  if (b < 4)`
`  c -= b;`
`}`
	`  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`

Name: _____

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	10	taken	not taken
	bge x2, x5, skip2	00	not taken	taken
	bnez x4, loop	10	taken	taken
i=2	blt x2 x5, skip1	01	not taken	taken
	bge x2, x5, skip2	01	not taken	not taken
	bnez x4, loop	11	taken	taken
i=3	blt x2 x5, skip1	10	taken	not taken
	bge x2, x5, skip2	00	not taken	taken
	bnez x4, loop	11	taken	not taken

(0.3 point for each row, +0.3 all corect)

What is the prediction accuracy for each branch? What is the prediction accuracy overall? (0.5 points for each)
blt: 0/4
bge: 2/4
bnez: 2/4
overall: 4/12

Name: _____

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	1	10	01	not taken	not taken
	bge x2, x5, skip2	0	01	00	not taken	taken
	bnez x4, loop	1	10	01	not taken	taken
i=2	blt x2 x5, skip1	1	10	00	not taken	taken
	bge x2, x5, skip2	1	10	00	not taken	not taken
	bnez x4, loop	0	10	10	taken	taken
i=3	blt x2 x5, skip1	1	10	01	not taken	not taken
	bge x2, x5, skip2	0	10	00	taken	taken
	bnez x4, loop	1	11	10	taken	not taken

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

blt: 2/4
bge: 3/4
bne: 1/4
Overall: 6/12

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?

The bge instruction will benefit the most, because it is correlated with the preceding blt instruction. It will usually go in the opposite direction of the blt instruction.

(-3 points for wrong reason, -2 points for wrong answer)

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

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

BTB can be placed in the fetch stage for quick target address predictions, but should have a small number of entries and can't cover lots of branches.

BHT is placed in the decode stage only for branch direction prediction. It has a large number of entries to cover a large number of branches, but cannot avoid pipeline bubbles.

Name: _____

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	yes	0xBADE2140
5	0x1000	yes	0xF00D3ABC
8	0x2000	no	
13	0x1000	no	
16	0x1000	yes	0xDEADBEEF
18	0x3000	no	

Table 3.3 Load Queue

-1 for each incorrect “Can execute” entry
-1 for each incorrect “Value” entry

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?

Loads 13 and load 18 can now be speculatively issued. Load 13 has the speculative value 0xF00D3ABC. Load 18 has the speculative value 0x1234ABCD.

-3 if incorrect numbers identified
-2 if incorrect value given

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?

Load 18 was mis-speculated. We should squash the load and all following instructions. Load 13 is fine because the store to the same address comes after it in program order.
-3 if load 18 not identified as misspeculated
-2 if recovery method not correct or not mentioned
-2 if load 13 identified as misspeculated

Name: _____

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.

Name: _____

A. **(5 points)** Schedule instructions for the VLIW machine in Table 4-1 without loop unrolling and software pipelining.

Label	ALU1	ALU2	MEM	FPU1	FPU2
loop:		addi a1,a1,8	fld f1,0(a1)		
				fmax.d f0,f0,f1	flt.d t1,f0,f1
		beqz t1,skip			
		addi s0,t0,0			
skip:	addi t0,t0,1	bltu t0,a0,loop			

Table 4-1: VLIW Scheduling without Optimizations

-1 for each mistake

Name: _____

B. **(10 points)** Schedule instructions for the VLIW machine with software pipelining but without loop unrolling in Table 4-2 including the prologue and the epilogue. You do not need to find the optimal scheduling.

Label	ALU1	ALU2	MEM	FPU1	FPU2
	addi t0,t0,1	addi a1,a1,8	fld f1,0(a1)		
				fmax.d f0,f0,f1	flt.d t1,f0,f1
	addi t0,t0,1	addi a1,a1,8	fld f1,0(a1)		
loop:		beqz t1,skip1		fmax.d f0,f0,f1	flt.d t1,f0,f1
		addi s0,t0,-2			
skip1:	addi t0,t0,1	addi a1,a1,8	fld f1,0(a1)		
	bltu t0,a0,loop				
		beqz t1,skip2		fmax.d f0,f0,f1	flt.d t1,f0,f1
		addi s0,t0,-2			
skip2:					
		beqz t1,skip3			
		addi s0,t0,-1			
skip3:					

Table 4-2: VLIW Scheduling with Software Pipelining

+1 prologue, +1 correct prologue
+1 epilogue, +1 correct epilogue
+1 loop, +1 correct i-2, +1 correct i-1, +1 correct i
+2 all correct

Name: _____

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:     setvl t0, a0 (2 pt)_____ # set VL for loop, t0 = VL

          # load A[i+20] (3 pts)

          add t1, a1, 160_____

          vld v2, 0(t1)_____

          _____

          # load B[2*i], load B[2*i+1] (5 pts)

          li, t2, 16_____

          vlds v3, 0(a2), t2_____

          vlds v4, 8(a2), t2_____

          _____

          _____

          # Compute A[i+20] + B[2*i]*B[2*i+1] (3 pts)

          vmadd v0, v3, v4, v2 _____

          (-2 with more than one instruction)__

          _____

          _____
```

Name: _____

```
# Store A[i]  (2 pts)
vst v0, 0(a1) _____
_____
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
```

B. ***(CS 252 Only)*** We have a vector machine where 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?

3 (functional units) / 8 (= MAXVL / #lanes)

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

3 (function units) / 2 (= MAXVL / #lanes)

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

Nothing as functional units are fully pipelined.

Name: _____

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

`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++;`
`}`
	`// 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`

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.

8 threads

Two instructions between fld and the fadd that depends on it. If load takes 16 cycles, then you need 16/2 = 8 threads.

2 points if right number and right explanation
1 point if wrong number but right explanation

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.

5 threads

At steady state, we can run five instructions before the fadd dependent on the load without blocking. We have to cover 16 - 2 = 14 cycles of latency, which would require $\lceil 14/5 \rceil + 1 = 4$ threads. However, the fdiv is dependent on the result of the fadd right before it, so we still need 5 threads to cover all latencies.

4 points for correct answer and correct explanation
3 points if answer is 4 and correct explanation
2 points if correct answer but wrong explanation

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?

The last addi instruction before the branch can be moved up before the second fadd instruction. This will increase the number of instructions between the fld and fadd to 3 so that we only need ⌈16/3⌉ = 6 threads.

-1 if wrong reordering
-1 if wrong 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?

No, you cannot reduce the number of threads. Moving the addi to between the two fadd instructions doesn't change the number of threads needed to cover the load latency. Moving it to between between the fadd and fdiv changes the threads needed to cover that latency to ⌈3/2⌉+1 = 3, but then you need ⌈14/4⌉ + 1 = 5 threads to cover the load latency.

2 points for correct answer
1 point for plausible reordering

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	Duplicated
Fetch Unit	Shared
Rename Table	Duplicated
Physical Register File	Shared
Issue Window	Shared
Functional Units	Shared
ROB	Shared