CS 152 Computer Architecture and Engineering
CS 252 Graduate Computer Architecture
Midterm #1
February 26th, 2018
Professor Krste Asanovic

Name:_____

I am taking CS152 / CS252

This is a closed book, closed notes exam.

80 Minutes. 19 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	25 points	25 points	
Question 3	25 points	25 points	
Question 4	25 points	30 points	
Total	100 points	110 points	

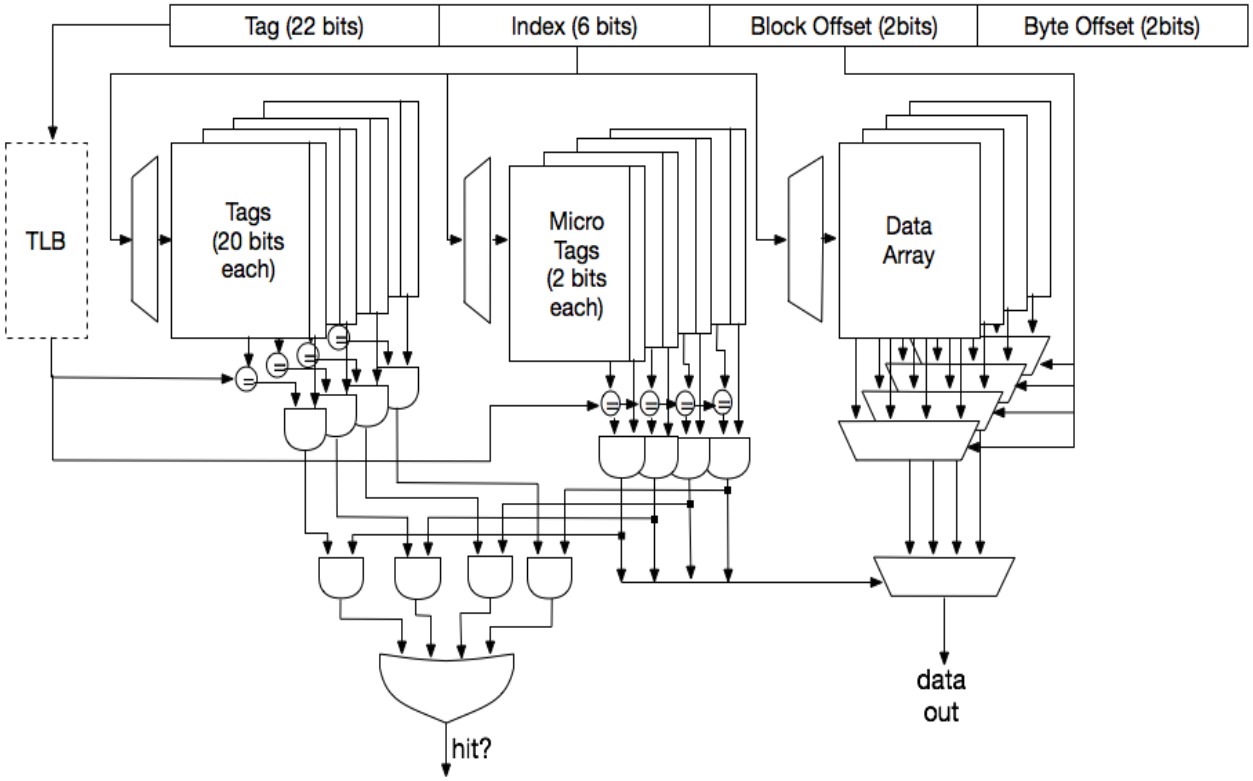
Name: _____

Question 1: Microtagged Cache [25 + 5 points]

In this problem, we explore *microtagging*, a technique to reduce the access time of set-associative caches. Recall that for associative caches, the tag check must be completed before load results are returned to the CPU, because the result of the tag check determines which cache way is selected. Consequently, the tag check is often on the critical path.

The time to perform the tag check (and, thus, way selection) is determined in large part by the size of the tags. We can speed up way selection by checking only a subset of the tag—called a microtag—and using the results of this comparison to select the appropriate cache way. Of course, the full tag check must also occur to determine if the cache access is a hit or a miss, but this comparison proceeds in parallel with way selection. We store the remainder of the full tags separately from the microtag array.

We will consider the impact of microtagging on a 4-way set-associative 4KiB data cache with 16-byte lines. Addresses are 32 bits. Microtags are 2 bits. A row in each tag memory array contains one tag and two status(valid and dirty) bits. Figure 1-1, below, shows the modified tag comparison and driver hardware in the microtagged cache:



Name: _____

Table 1-1 shows the delays for each cache component:

Component	Delay equation (ps)	Delay (ps)		
Decoder	20×(# of index bits) + 100	220		
Memory array	20×$\log_2$(# of rows) + 40×$\log_2$(# of bits in a row) + 100	Tag	Microtag	Data
		398	300	500
Comparator	20×(# of tag bits) + 50	Tag	Microtag	
		450	90	
4-to-1 MUX	50×$\log_2$N + 100	200		
2-input gate (AND)		50		
4-input gate (OR)		100		

Table 1-1: Cache component delays

A. **(4 points)** Which bits of each 22-bit full tag should be used as the 2-bit microtag? Explain the reason briefly.

B. **(5 points)** Assume data must be available in one cycle on a cache hit, while full tag checks do not need to complete in one cycle. What is the critical path and the cycle time?

C. **(5 points)** Assume the page size is 4 KiB. Can aliases occur in Figure 1? If not, explain why not. If aliases can happen, can you suggest an efficient solution to prevent them? Explain your reasoning carefully.

D. **(5 points)** How does the miss rate of this cache compare with a direct-mapped cache of the same capacity and line size? Explain your reasoning.

E. **(6 points)** We consider increasing the number of bits in microtags to 6 bits. How does this change the hit time, miss rate, and the miss penalty? Explain your reasoning carefully.

	Increase / Decrease / No effect?
Hit time	
Miss rate	
Miss penalty	

F. (CS252 only) +5 points You decide to implement way prediction for the instruction cache instead of microtags. Figure 5-1 shows how way prediction works.

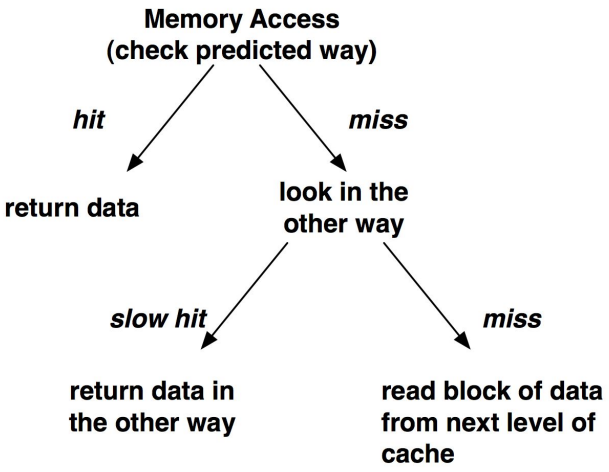


Figure 5-1: Way Prediction FSM

On a cache access, the prediction is used to route the data. If it is incorrect, there will be a delay as the correct way is accessed. If the desired data is not resident in the cache, it is like a normal cache miss.

For a given memory access, the way predictor chooses the most-recently-used way in the set. For a 4-way set-associative 4KiB instruction cache with 16-byte lines, under what scenarios do you expect this predictor to work well and under what scenarios do you expect the way predictor to mispredict?

Name: _____

Question 2: Superscalar In-order Processor with Microtagged Caches [25 points]

Your best friend, Klay Curry, designed a two-way superscalar in-order processor using the microtagged cache in Question 1 as follows:

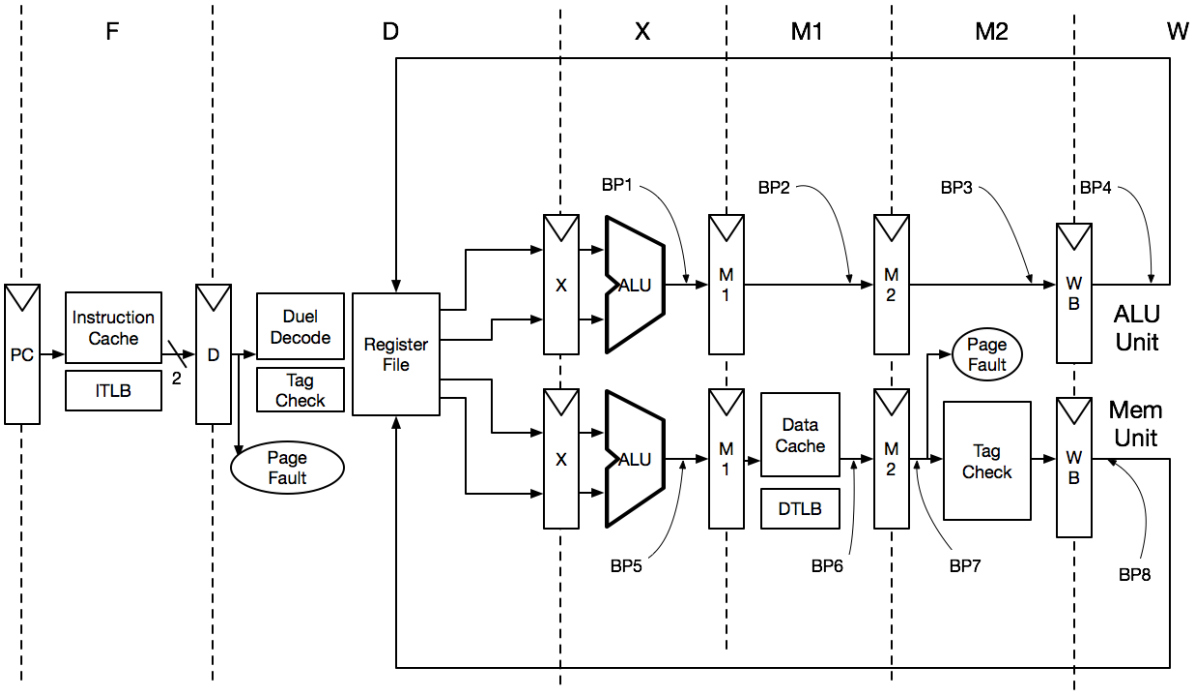


Figure 2-1: Two-way superscalar in-order pipeline with microtagged caches

Two instructions are fetched every cycle and both instructions are issued at the same time when there are no pipeline hazards. If either instruction cannot move forward, the following instructions are also stalled. Thus, writebacks are always in-order.

There are two functional units available in this pipeline: the ALU unit and the MEM unit. ALU and branch instructions can be issued to either unit.

On the other hand, memory instructions can only be issued to the MEM unit. Memory addresses are calculated by the ALU in the X stage and the memory is accessed in the following M1 stage. Note that caches can be read in one cycle because full tag checks are postponed until the M2 stage. Stores can only be written at the end of the M2 stage after full tags are checked. Ignore structural hazards between loads and stores in the data cache array.

Name: _____

Assume there is perfect branch prediction. Also, **the pipeline is fully bypassed.** Each bypass source has been numbered in the figure (BP1 ~ BP8). Bypass paths connect their sources to the inputs of the X registers.

In this question, assume there are no cache misses and no exceptions other than page faults. Page faults from instruction accesses are detected in the D stage and page faults from data accesses are detected in the M2 stage.

A. **(5 points)** Explain how page-fault exceptions can be made precise in this pipeline.

B. **(10 points)** Now, Klay Curry is benchmarking their design with a simple integer vector-vector add:

```
# for (i = 0 ; i < N ; i++)
#   c[i] = a[i] + b[i]
Loop:    lw x2, 0(x1)           # load a[i]
         lw x4, 0(x3)           # load b[i]
         add x5, x2, x4         # c[i] = a[i] + b[i]
         sw x5, 0(x6)           # store c[i]
         addi x1, x1, 4         # bump pointer
         addi x3, x3, 4         # bump pointer
         addi x6, x6, 4         # bump pointer
         addi x7, x7, 1         # i++
         bne x7, x8, Loop       # x8 holds N
```

Figure 2-2: Code snippet for vector-vector add

Fill out the pipeline diagram (Figure 2-3) when you execute the assembly code in Figure 2-2. Specify which bypasses are used for each instruction. Note a single instruction might use more than one bypass path. What is the CPI for this code?

(F: Fetch, D: Decode, X: Execute, M1: Memory, M2: Tag Check, W: writeback)

Instructions	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	Bypass?
lw x2, 0(x1)	F	D	X	M1	M2	W																No
lw x4, 0(x3)	F	D	D	X	M1	M2	W															No
add x5, x2, x4																						
sw x5, 0(x6)																						
addi x1, x1, 4																						
addi x3, x3, 4																						
addi x6, x6, 4																						
addi x7, x7, 1																						
bne x7, x8,Loop																						
lw x2, 0(x1)																						
lw x4, 0(x3)																						

Figure 2-3: Pipeline diagram for Question 2. B

Name: _____

C. **(CS152 only) (10 points)** Now, Klay Curry has a compiler that unrolls loops and reschedule instructions. Assume the loop operates on an even number of elements in the vectors.

```
Loop:   lw x2, 0(x1)          # load a[i]
        addi x1, x1, 8        # bump pointer a
        lw x4, 0(x3)          # load b[i]
        addi x3, x3, 8        # bump pointer b
        lw x8, -4(x1)         # load a[i+1]
        add x5, x2, x4        # c[i] = a[i] + b[i]
        lw x9, -4(x3)         # load b[i+1]
        addi x7, x7, 2        # i+=2
        sw x5, 0(x6)          # store c[i]
        addi x10, x8, x9      # c[i+1] = a[i+1] + b[i+1]
        sw x10, 4(x6)         # store c[i+1]
        add x6, x6, 8         # bump pointer c
        bne x7, x11, Loop     # x11 holds N
```

Figure 2-4: Loop unrolling and rescheduling

Fill out the pipeline diagram (Figure 2.5). Also, specify what bypasses are used for each instruction. What is the CPI with this optimization? What is the speedup over Figure 2-2? (F: Fetch, D: Decode, X: Execute, M1: memory access, M2: tag check, W: writeback)

Instructions	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	Bypass?
lw x2, 0(x1)	F	D	X	M1	M2	W																No
addi x1, x1, 8	F	D	X	M1	M2	W																No
lw x4, 0(x3)																						
addi x3, x3, 8																						
lw x8, -4(x1)																						
add x5, x2, x4																						
lw x9, -4(x1)																						
addi x7, x7, 2																						
sw x5, 0(x6)																						
addi x10, x8, x9																						
sw x10, 8(x6)																						
add x6, x6, 8																						
bne x7, x11, Loop																						
lw x2, 0(x1)																						
addi x1, x1, 8																						

Figure 2-5: Pipeline diagram for Question 2. 3)

Name: _____

D. **(CS252 only)** Draymond Durant suggests a new architecture (Figure 2-6) to efficiently execute the following floating-point vector-vector add (Figure 2-7).

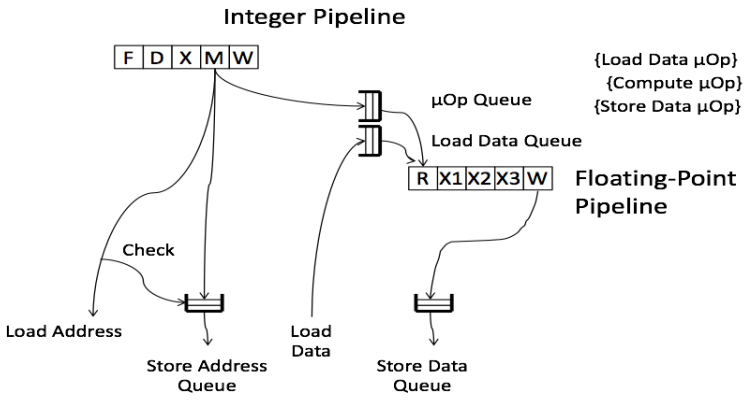


Figure 2-6: Simple decoupled machine

```
Loop:    fld f0, 0(x1)          # load a[i]
         fld f1, 0(x2)          # load b[i]
         fadd f2, f0, f1        # c[i] = a[i] + b[i]
         fsd f2, 0(x3)          # store c[i]
         addi x1, x1, 4         # bump pointer
         addi x2, x2, 4         # bump pointer
         addi x3, x3, 4         # bump pointer
         addi x4, x4, 1         # i++
         bne x4, x5, Loop       # x5 holds N
```

Figure 2-7: Floating-point vector vector add

In Figure 2-6, all instructions flow through the integer pipeline. However, floating-point instructions issue microops (load data, compute, or store data) to the floating-point pipeline. Instructions following a floating-point instruction can execute without waiting for the microop to complete.

F	D	X	M		WB
		F1	F2	F3	WB

Figure 2-8: Traditional pipeline with floating-point units

Name: _____

(CS252 only) (5 points) Will the floating-point vector-vector-add execute more efficiently in the decoupled machine in Figure 2-6 or in the traditional pipeline in Figure 2-8? Explain your reasoning.

(CS252 only) (5 points) Can you make the new architecture handle page-fault exceptions precisely? Explain.

Name: _____

Question 3: Micro-Programming [25 points]

For this problem, you will implement a new copy-if-non-zero (CPN) instruction. The new instruction has the following format.

CPN (rd), (rs1), rs2

This instruction will copy a word from the address given by register rs1 to the address given by register rd if the value in register rs2 is non-zero.

if Reg[rs2] != 0 then Mem[Reg[rd]] ← Mem[Reg[rs1]]

Fill out the table on the next page with the microcode for CPN. Use don't cares (*) for fields where it is safe to use don't cares. Study the hardware description well, and make sure all your microinstructions are legal.

Please comment your code clearly. If the pseudo-code for a line does not fit in the space provided, or if you have additional comments, you may write in the margins as long as you do it neatly.

Finally, make sure that the instruction fetches the next instruction (i.e., by doing a microbranch to FETCH0 as discussed).

State	PseudoCode	ldIR	Reg Sel	Reg Wr	en Reg	ldA	ldB	ALUOp	en ALU	ld MA	Mem Wr	en Mem	Imm Sel	en Imm	uBr	Next State
FETCH0:	MA ← PC; A ← PC	*	PC	0	1	1	*	*	0	1	*	0	*	0	N	*
	IR ← Mem	1	*	*	0	0	*	*	0	0	0	1	*	0	N	*
	PC ← A+4	0	PC	1	1	0	*	INC_A_4	1	*	*	0	*	0	D	*
. . .																
NOP0:	microbranch back to FETCH0	0	*	*	0	*	*	*	0	*	*	0	*	0	J	FETCH0

Name: _____

Name: _____

Question 4: Virtual Memory [25 + 5 points]

A. **(15 points)** The table on the next page shows the contents of a portion of physical memory used for page tables. Assume the system uses 64-bit words, 16-byte pages, three-level page tables, and a fully associative two-entry TLB with LRU eviction. Each stage of the page table uses a single bit index. At the beginning, the TLB is empty and the free pages list contains page numbers 0x16, 0x5, 0x18, 0x12, and 0x19 in order from first-to-be-allocated to last-to-be-allocated. For the following virtual memory address trace, indicate whether the access results in a TLB hit, a page table hit, or a page fault, and give the translated physical address. Fill out the memory table and the TLB with its final state. Assume that the page table base register is set to 0 and TLB fills in from left to right. The entries in the page table are the full physical addresses of the start of the page, **not just the PPN** .

Virtual Address	TLB hit/page hit/page fault	Physical Address
0x58	Page hit	0x178
0x10		
0x50		
0x60		
0x18		

Name: _____

Memory

Addr	Contents (Phys Addr)
0x00	0x020
0x08	0x040
0x10	
0x18	
0x20	0x070
0x28	0x090
0x30	
0x38	
0x40	0x080
0x48	
0x50	
0x58	
0x60	
0x68	
0x70	0x110
0x78	
0x80	0x150
0x88	0x170
0x90	
0x98	0x130

TLB

VPN		
PPN		

B. **(5 points)** You are asked to design a virtually indexed, physically tagged cache. A page is 4096 bytes. The cache must have 128 lines of 64 bytes each. What associativity must the cache have in order for there to be no aliasing?

C. **(5 points)** Assume the cache is direct-mapped and an inclusive L2 is used to detect aliasing. If the L2 detects an alias for the physical address 0x80001468, which sets in the L1 could contain the aliased entry? The sets are indexed starting from zero. Give your answers in decimal.

D. **(CS252 only) (5 points)** What are the advantages and the disadvantages of hashed page tables?