CS/ECE 552: Introduction to Computer Architecture
Department of Computer Sciences
University of Wisconsin-Madison

Prof. Karthikeyan Sankaralingam

Final Examination

May 14, 2009
Approximate Weight: 27.5%

2 hour 00 minutes

CLOSED BOOK. You can bring two-cheat sheets (two-sides 8.5 x 11 page).

Exam is **one-sided, total of 14 numbered pages**. Plan your time carefully.
One blank page included in the end for rough work. Also use left-side blank pages for rough work.

NAME: ____

Email: ____

Problem number	Maximum points	Actual points
1	**12**	
2	**12**	
3	**24**	
4	**12**	
5	**20**	
6	**40**	
7	**16**	
8	**20**	
Total	**156**	

Problem 1: Virtual Memory (4x3 = 12 points)

Consider a memory system with the following parameters. The virtual address is 64 bits (v63-v0 where v0 is least significant). The physical address is 42 bits (p41-p0). The page size is 8 kilobytes. The translation lookaside buffer (TLB) is 64 entries, two-way set-associative and accessed *before* the cache. The cache is unified (i.e., instructions and data), 32 kilobytes, four-way set-associative, LRU replacement, and 64-byte blocks (lines). All addresses are byte addresses.

a) Consider a naïve page table that stores an entry for every page in the virtual address space. How many entries would the page table store? In each page table entry, which bits of the physical address would be stored?

b) What are techniques for making page tables much smaller than the naïve page table of part (a)?

c) For the TLB, which bits will be stored in tag and which will select the set (the index)?

Problem 2: Caches (4x3 = 12 points)

Consider the *same* memory system as Problem 1. The virtual address is 64 bits (v63-v0 where v0 is least significant). The physical address is 42 bits (p41-p0). The page size is 8 kilobytes. The translation lookaside buffer (TLB) is 64 entries, two-way set-associative and accessed *before* the cache. The cache is unified (i.e., instructions and data), 32 kilobytes, four-way set-associative, LRU replacement, and 64-byte blocks (lines). All addresses are byte addresses.

(a) For the cache, which bits will be stored in tag, which will select the set (the index), and which bit select the data within the block?

(b) Increasing cache block size usually improves a cache's miss ratio, but does not always improve performance (e.g., the effective access time for memory access). How is this possible?

(c) A write-buffer for the cached designed for our single-cycle in-order processor in the homework can never make any difference to performance. Justify or explain why this statement is wrong.

Problem 3: Caches (4x6 = 24 points)

Consider a cache with the following characteristics:

1. 32-byte blocks
2. 6-way set associative
3. 1024 sets total
4. 47-bit addresses
5. Byte-addressable cache

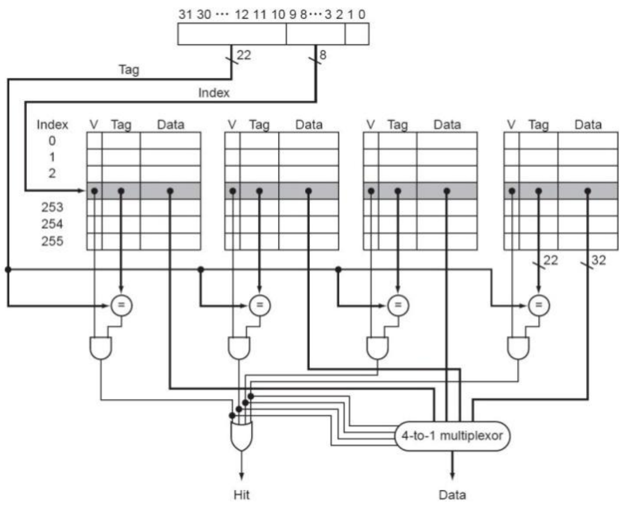
a) How many bytes of data storage are there?

b) Show the break-down of the 47-bit address in terms of word-select (or offset) within the cache block, index bits and tag bits.

c) What is the total number of bits needed to implement the cache?

d) How is the index to the cache computed i.e. what bits are used and what logical or arithmetic operation is applied on the bits?

e) Draw a high-level cache organization diagram for this cache. Use the reference diagram shown alongside for a 4-way associate cache with 256 sets.



f) Consider a 48-byte cache line, show the word-select/offset bits, index bits, and tags bits. How is the offset and index computed?

Problem 4: Arithmetic (4+8 = 12 points)

a) What are *denormalized* floating point numbers? What is their purpose?

b) Consider two single precision floating point numbers a and b. Consider the computation below:

   c = (a-b)/a

   c is the relative difference between a and b. What is the maximum relative difference that can be expressed with single precision floating point? Provide an equation or describe how you would approach this question.

Problem 5: Short answer (4x5 = 20 points)

a) How does software access a device, given that most instruction set architectures do not have explicit I/O instructions? How do most systems prevent user code from directly accessing devices (potentially interfering with other users)?

b) What functions does the operating system provide in accessing I/O?

c) What is the difference between an interrupt and exception?

d) In a multiprocessor environment with write-back caches explain why cache coherence is required and outline an example. Explain why write-through caches will not solve the problem.

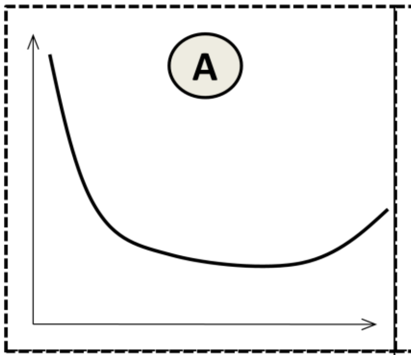
e) In a RAID-10 array with 20 disks, how many failures can be tolerated and under what conditions?

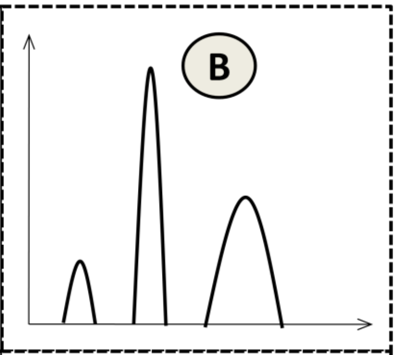
Problem 6: Qualitative analysis (4x10 = 40 points)

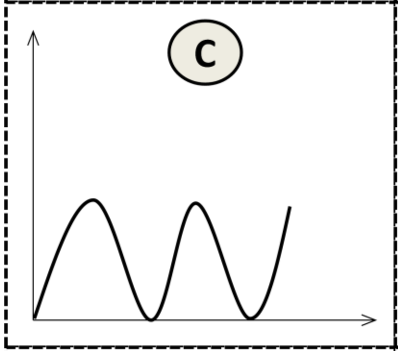
On the left are 10 descriptions of graphs describing different trends labeled a-j. On the right are 10 graphs. Match each description with a graph that shows the trend and fill in the table below. You can reuse graphs and not all graphs are used at least once.

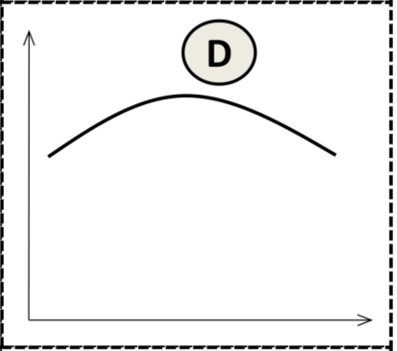
1. Cache miss penalty vs. block size (on X-axis)
2. Cache hit rate vs. block size
3. AMAT vs. block size
4. Cache access time vs. cache size
5. Execution time of program vs. cache size (single level of cache)
6. Pipeline latency (in cycles) vs. pipeline depth (# of stages)
7. Probability of reference vs. Address range (over entire execution of program)
8. Maximum speedup vs. fraction of program that can be speedup using a given technique
9. MTTF for a disk array vs. number of disks (no redundancy or RAID)
10. Peace of mind vs. number of days to verify cache ☺

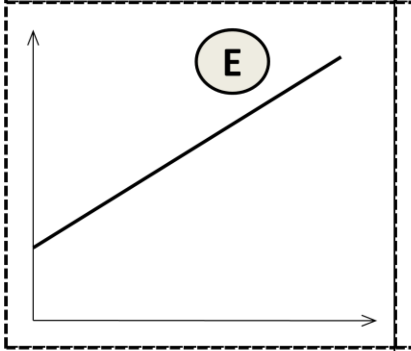
DESCRIPTION	GRAPH NUMBER	DESCRIPTION	GRAPH NUMBER
1		6	
2		7	
3		8	
4		9	
5		10	

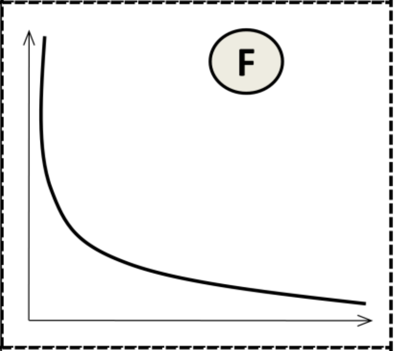


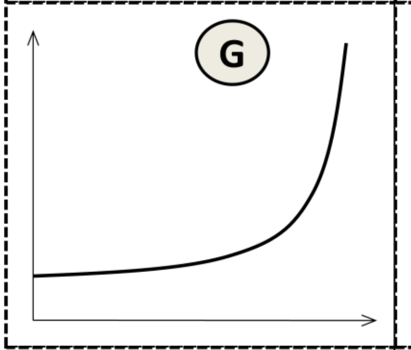


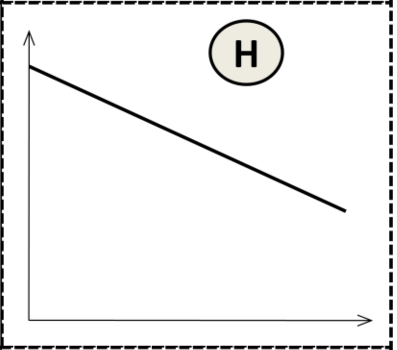


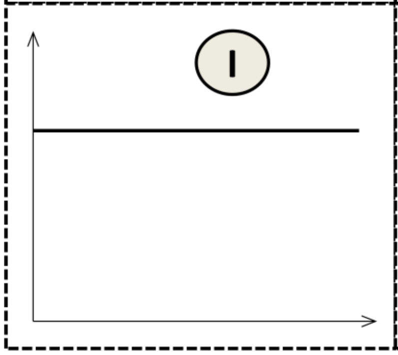


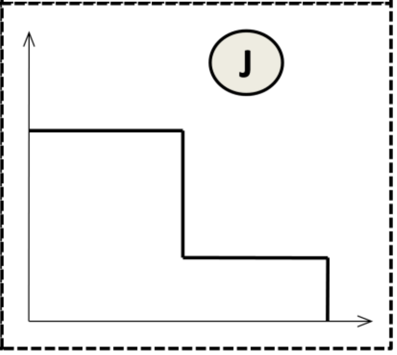












Problem 7: VM+Caches (6+10 = 16 points)

Part 1: Consider the following three caches designs and match them to the ordering of accesses to the TLB and the cache.

a) Virtually Indexed Virtually Tagged	Access TLB, then access cache
b) Physically Indexed Physically Tagged	Access Cache, access TLB only on cache miss
c) Virtually Indexed Physically Tagged	Access TLB and Cache in parallel

Part 2: Circle the legal combinations of cache organization and cache sizes in the list below. If necessary, assume a 40-bit virtual address and 32-bit physical address.

a) **VIVT, 8kb virtual page size, 32-byte cache blocks, 32kb direct mapped cache**
b) **PIPT, 16kb virtual page size, 32-byte cache blocks, 16kb direct mapped cache**
c) **VIPT, 8kb virtual page size, 32-byte cache blocks, 32kb direct mapped cache**
d) **VIPT, 8kb virtual page size, 32-byte cache blocks, 32kb 4-way set-associate cache**
e) **VIPT, 8kb virtual page size, 32-byte cache blocks, 32kb 16-way set-associate cache**

Use page on left for more space for this problem

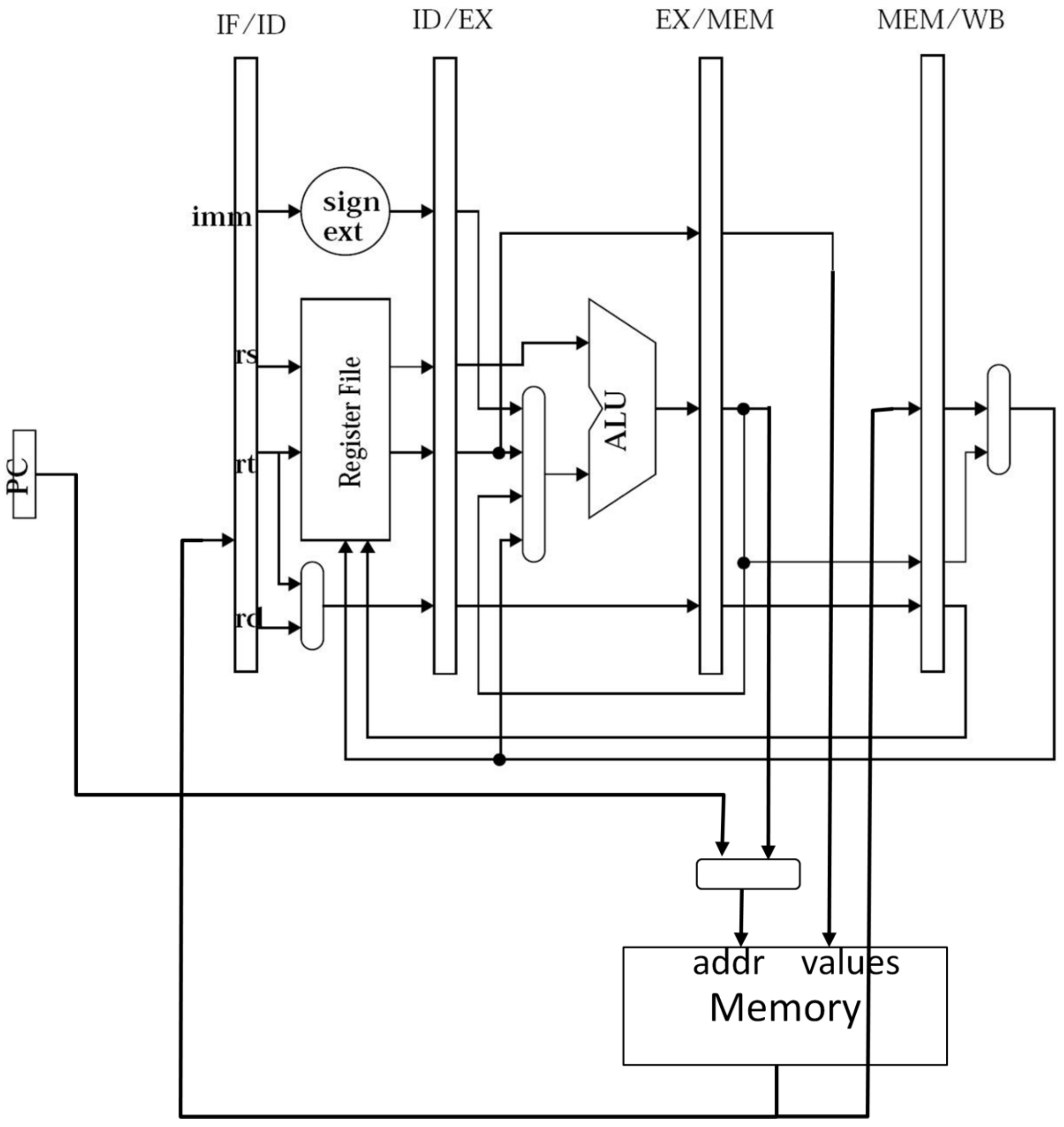
Problem 8: (20 points): Pipeline similar to what you saw in Exam1. This is not a trick question!

High performance datapaths use bypass paths (also known as data forwarding logic) to reduce pipeline stalls. However, bypass paths are relatively expensive especially in some wire constrained technologies. To reduce the cost (and potential cycle time impact), some architecture have explored omitting some of the possible bypass paths. Consider the datapath illustrated below (note that the PC update logic and all control logic is intentionally omitted).
This pipelined datapath is similar to the one in the book, buy only has bypass paths on one side of the ALU.
Also it has one memory only, which has one port, so that any cycle either an instruction can be fetched or a ld/store can access memory. Assume the mux enforces fairness, so that instruction-fetch and datamemory access are never starved and get equal priority by alternating.
Assume that the register file internally bypasses the value, so that if register $i is read and written in the same cycle, then the read returns the new value. Assume that the control logic bypasses the data as soon as possible using the given forwarding data paths, and stalls decode otherwise. You may NOT add additional data paths.

In this problem, you will look at how a program snippet performs on this pipeline. Recall that R-format

instructions have the form:	`opcode rd, rs, rt`
and I-format instructions have the form	`opcode rt, imm(rs)`
or	`opcode rt, rs, imm`

Use the table on the next page to show how the given instruction sequence flows through the pipeline and where stalls are necessary to resolve hazards.


IF/ID
ID/EX
EX/MEM
MEM/WB
imm
sign ext
rs
rt
rd
Register File
ALU
PC
addr
values
Memory

	Cycle																			
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
add $1, $2, $3	F	D	X	M	W															
sub $4, $5, $1		F	D																	
or $6, $1, $4																				
and $7, $4, $6																				
lw $9, 4($7)																				
add $1, $9, $2																				
sw $1, 4($7)																				

Consider the code and pipeline above. Show the execution of this code on the pipeline above.
Use the letters, F, D, X, M, and W.

For each cycle where a stall occurs explain why.

Scratch sheet