

Computer Architecture Test 1

Question 1. (3 points) Suppose we had a block transfer from an I/O device to memory. The block consists of 4096 words and one word can be transferred at a time. For each of the following, indicate the number of interrupts needed to transfer a block:

- a) programmed-I/O 0
- 3 b) interrupt-driven I/O 4096
- c) DMA (direct-memory access) 1

I process during I/O. When I/O completes, the I/O controller sends interrupt to CPU.

Question 2. (4 points) What is the main difference between programmed I/O and interrupt-driven I/O?

4 In programmed I/O the O.S. busy-waits wasting the CPU's time waiting for the I/O to complete. In interrupt-driven I/O, the O.S. turns CPU control over to another

Question 3. (4 points) On a paged, multiprogrammed, multi-user computer system that uses memory-mapped I/O, explain what hardware support for the operating system is needed to guard against infinite loops in user programs.

4 OS starts a countdown, CPU-timer before turning control over to user pgm. If timer expires, CPU traps to O.S. CPU has dual-mode (system + user) with CPU-timer instructions <sup>with</sup> privileged.

Question 4. (4 points) On a paged, multiprogrammed, multi-user computer system that uses memory-mapped I/O, explain what hardware support for the operating system is needed to restrict a user program to its own main memory address space.

4 Since a program can only access memory through virtual address translation using the page table. Access to the page table must be restricted.

Question 5. (4 points) On a paged, multiprogrammed, multi-user computer system that uses I/O instructions, explain what hardware support for the operating system is needed to restrict a user program from accessing other users' data files.

4 The CPU must have dual-mode (system, user) execution and the I/O instruction are privileged so only the CPU in system mode can execute them.

Question 6. (6 points) Why is the L1 cache on most processors split into two: I-cache (for instructions only) and D-cache (for data only)?

6 So both can operate simultaneously, i.e., fetch stage of pipeline can be accessing I-cache while "memory" stage can be accessing D-cache.

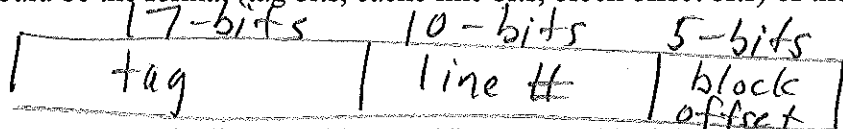
Question 7. (12 points) Suppose we have 32-bit memory addresses, a byte-addressable memory, and a 32 KB ( $2^{15}$  bytes) cache with 32 ( $2^5$ ) bytes per block.

a) How many total lines are in the cache?

$$\frac{2^{15}}{2^5} = 2^{10} \text{ lines}$$

b) If the cache is direct-mapped, how many cache lines could a specific memory block be mapped to? |

c) If the cache is direct-mapped, what would be the format (tag bits, cache line bits, block offset bits) of the address? (Clearly indicate the # of bits in each)

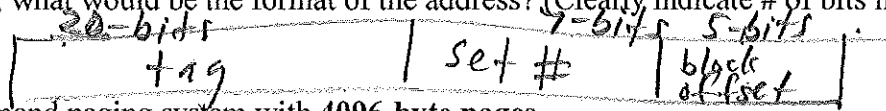


d) If the cache is 8-way set associative, how many cache lines could a specific memory block be mapped to? 8

e) If the cache is 8-way set associative, how many sets would there be?

$$\frac{2^{10}}{2^3} = 2^7$$

f) If the cache is 8-way set associative, what would be the format of the address? (Clearly indicate # of bits in each)

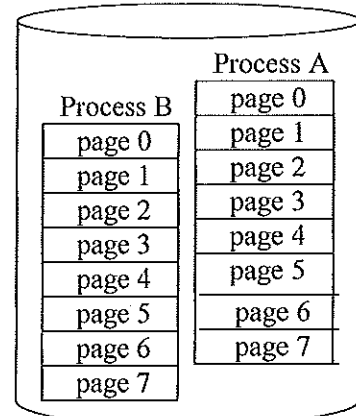


Question 8. (15 points) Consider a demand paging system with 4096-byte pages.

Process A Page Table

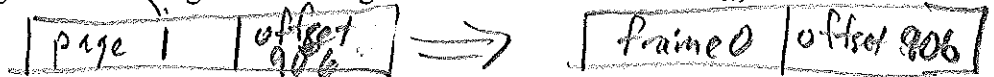
Frame #	Valid bit (Loaded In Memory)	Page Protection Bits
0	0	
1	1	
2	1	
3	1	
4	0	
5	0	
6	0	
7	0	

Frame #	Physical Memory
0	page 1 of A
1	page 2 of B
2	page 4 of A
3	page 1 of B
4	page 4 of B
5	page 2 of A
6	page 3 of A



a) Complete the above page table for Process A (ignore the "Page Protection Bits" column).

b) If process A is currently running and the CPU generates a logical/virtual address of  $5002_{10}$ , then what would be the corresponding physical address?



c) What is the TLB (translation-lookaside buffer) and why is it important for efficient operation of a paged, virtual memory system?

TLB is a small, fully-associate cache in the CPU containing page-table entries (PTE). If the PTE is found in the TLB, then we avoid going to the slow page table.

Question 9. (8 points) Each page table entry can contain "Page Protection Bits" (as shown in the above question) to allow pages to be protected differently (e.g., page 1 can be shared with all processes). Segmentation is an alternative virtual memory scheme to paging. Why is protection in segmentation "better" than protection in paging?

Segments split the program into logical "chunks" (e.g. main pgm, run-time stack, etc.) for each segment.

Question 10. (12 points) Complete the table by indicating the number of branch penalties (assume 5-stage pipeline as discussed in lecture and homework #3)

High-level Code	Assembly/Machine Language	No BPB	1-bit BPB	2-bit BPB
for r := 0 to 100 do	LOAD_IMMEDIATE R3, #0 FOR_R: BGT R3, #100, END_FOR_R →	22	2	2
for c = 0 to 500 do	LOAD_IMMEDIATE R4, #0 FOR_C: BGT R4, #500, END_FOR_C →	2x100	2+4x99	2x100
:	...			
:	ADD R4, R4, #1 JUMP FOR_C →	1x100x 500	1	1
end for c	END_FOR_C:			
end for r	ADD R3, R3, #1 JUMP FOR_R → END_FOR_R:	1x100	1	1

Question 11. (10 points) Assume the same 5-stage pipeline discussed in class. Recall that:

- arithmetic instructions, e.g., "ADD R3, R2, R1" register R3 receives the result of adding registers R2 and R1
- load instruction, e.g., "LOAD R6, 16(R3)" loads R6 from the memory address specified by 16 + content in R3
- store instruction, e.g., "STORE R3, 8(R5)" stores R3 to the memory address specified by 8 + content of R5

a. What would the timing be **without** bypass-signal paths/forwarding (use "stalls" to solve the data hazard)?

		Time →																					
#	Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	ADD R3, R2, R1	F	D	E	M	W																	
2	SUB R4, R3, R5		F	-	-	-	D	E	M	W													
3	LOAD R6, 16(R3)						F	D	E	M	W												
4	MUL R3, R6, R2							F	-	-	-	D	E	M	W								
5	STORE R3, 8(R5)											F	-	-	-	E	M	W					

b. What would the timing be **with** bypass-signal/forwarding paths? (You might not need all 22 cycles)

		Time →																					
#	Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	ADD R3, R2, R1	F	D	E	M	W																	
2	SUB R4, R3, R5		F	D	E	M	W																
3	LOAD R6, 16(R3)			F	D	E	M	W															
4	MUL R3, R6, R2				F	D	-	E	M	W													
5	STORE R3, 8(R5)					F	-	D	E	M	W												

c. Draw arrows in the above table (part b) indicating all forwarding.

Question 12. (9 points) Superscalar processors with out-of-order execution introduce new data-dependencies: WAW (write-after-write) and WAR (write-after-read).

a) Using the above code indicate an example of WAW: Instructions # 1 and # 4 have a WAW on R 3

b) Using the above code indicate an example of WAR: Instructions # 3 and # 4 have a WAR on R 3

c) Explain how register renaming enables a superscalar processor to achieve a higher level of instruction-level parallelism (ILP) within a program.

Program doesn't need to wait for reg. to become free since a different register is used

Question 13. (9 points) Two approaches for designing a computer is CISC (Complex Instr. Set Computer - pre-1980) and RISC (Reduced Instruction Set Computer post 1985).

The architectural characteristics of CISC machines include:

- complex high-level-like AL instructions (e.g., VAX MATCHC string instr.) as well as simple AL instructions
- variable format machine-language instructions (e.g., the DEC VAX instructions ranged from 2 to 57 bytes)
- many addressing modes (e.g., the DEC VAX had 22 addressing modes with some doubly indirect)
- micro-programmed control unit to execute instructions using a variable number of clock cycles

The architectural characteristics of RISC machines include: (emphasis on optimizing instruction pipeline)

- limited and simple instruction set
- few and simple addressing modes -- only reg.-to-reg. operations (only Load & Store instrs. access memory)
- simple, fixed-length instruction formats
- large number of registers so compiler technology can optimize register usage
- hardwired control unit - circuit used to drive the fetch-execute cycle by generating control signals for the CPU

Why are complex instructions of CISC (Complex Instr. Set Computer) machines difficult to pipeline?

Ideally we want

- each pipeline stage to take the same time, so regardless of the instruction we want the fetch, decode, execute, memory, write, etc. to be consistent.

This is difficult with CISC because:

- variable format makes it difficult to consistently fetch and decode CISC instructions
- complex addressing modes of CISC cause operand fetches and writing to vary in time.
- complex instructions vary widely in their execution stage.