

Computer Architecture Test 1

Question 1. (4 points) Indicate whether each of the following statements about programmed I/O and interrupt-driven I/O is True or False.

a)	programmed I/O is more appropriate for embedded systems than PCs (personal computers)	<input checked="" type="radio"/> True <input type="radio"/> False
b)	interrupt-driven I/O allows the CPU to perform useful work while the I/O is performed	<input checked="" type="radio"/> True <input type="radio"/> False
c)	programmed I/O allows the CPU to perform useful work while the I/O is performed	<input type="radio"/> True <input checked="" type="radio"/> False
d)	both use interrupts to indicate when the I/O is complete	<input type="radio"/> True <input checked="" type="radio"/> False

Question 2. (6 points) Indicate whether each of the following statements about interrupt-driven I/O and DMA (direct-memory access) is True or False.

a)	interrupt-driven I/O is more appropriate for block-oriented I/O devices than DMA	<input type="radio"/> True <input checked="" type="radio"/> False
b)	interrupt-driven I/O allows the CPU to perform useful work while the I/O is performed	<input checked="" type="radio"/> True <input type="radio"/> False
c)	DMA allows the CPU to perform useful work while the I/O is performed	<input checked="" type="radio"/> True <input type="radio"/> False
d)	both use interrupts to indicate when the I/O is complete	<input checked="" type="radio"/> True <input type="radio"/> False
e)	interrupt-driven I/O passes data through a CPU register during I/O	<input checked="" type="radio"/> True <input type="radio"/> False
f)	DMA passes data through a CPU register during I/O	<input type="radio"/> True <input checked="" type="radio"/> False

For questions 3 - 6, indicate all of the hardware support needed from the list (a-f) at the bottom of the page. (circle all the appropriate letter(s) (a-f) by the question)

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

Circle all that apply: a b c (d) (e) f

Question 4. (3 points) On a paged, multiprogrammed, multi-user computer system that uses memory-mapped I/O, indicate what hardware support for the operating system is needed to restrict a user program to its own main memory address space.

Circle all that apply: a b (c) d e f

Question 5. (3 points) On a paged, multiprogrammed, multi-user computer system that uses memory-mapped I/O, indicate what hardware support for the operating system is needed to restrict a user program from accessing other users' data files

Circle all that apply: a b (c) d e f

Question 6. (3 points) On a paged, multiprogrammed, multi-user computer system that uses I/O instructions, indicate what hardware support for the operating system is needed to restrict a user program from accessing other users' data files

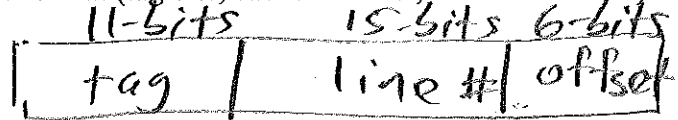
Circle all that apply: a (b) c d e f

Hardware support: (some may be used multiple times and some may not be used at all)

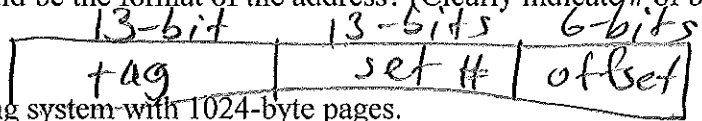
- a) privileged LOAD and STORE instructions that can only be executed by the CPU running in system mode
- b) privileged I/O instructions that can only be executed by the CPU running in system mode
- c) virtual-to-physical address translation that only maps to memory frames containing the process being executed
- d) CPU timer that traps/interrupts to the operating system when it expires
- e) privileged instruction to set the CPU timer that can only be executed by the CPU running in system mode
- f) privileged instruction to read the CPU timer that can only be executed by the CPU running in system mode

Question 7. (12 points) Suppose we have 32-bit memory addresses, a byte-addressable memory, and a 2 MB (2^{21} bytes) cache with 64 (2^6) bytes per block.

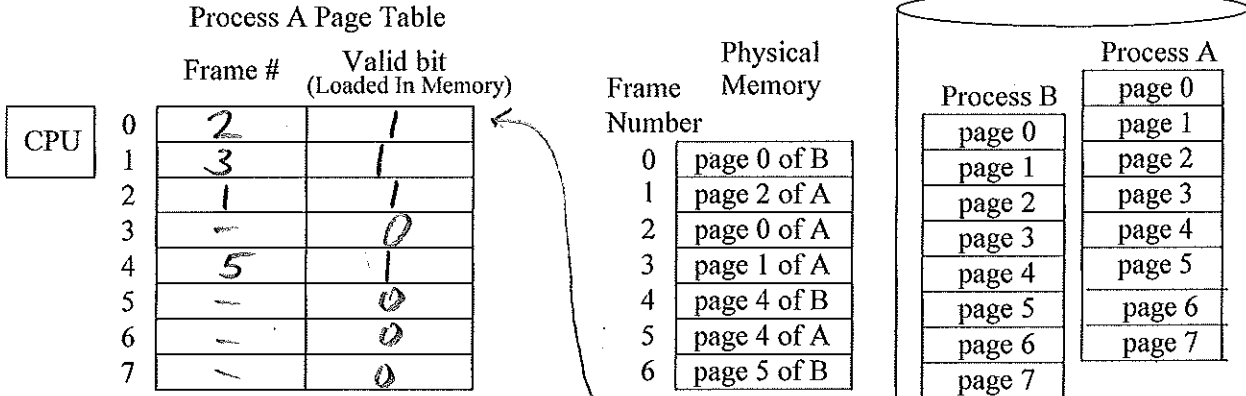
- 2 a) How many total lines are in the cache? $\frac{2^{21}}{2^6} = 2^{15}$ lines
- 2 b) If the cache is direct-mapped, how many cache lines could a specific memory block be mapped to? 1
- 2 c) If the cache is direct-mapped, what would be the format (tag bits, cache line bits, block offset bits) of the address? (Clearly indicate the # of bits in each)



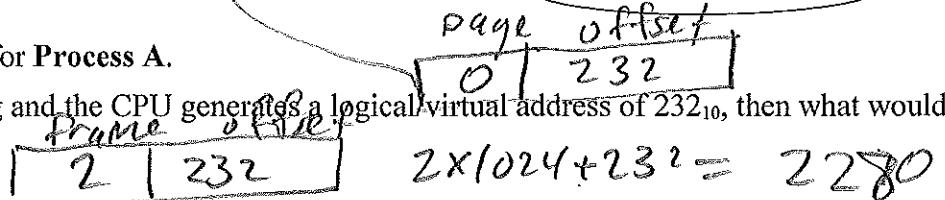
- 2 d) If the cache is 4-way set associative, how many cache lines could a specific memory block be mapped to? 4
- 2 e) If the cache is 4-way set associative, how many sets would there be? $\frac{2^{15}}{4} = 2^{13}$ sets
- 2 f) If the cache is 4-way set associative, what would be the format of the address? (Clearly indicate # of bits in each)



Question 8. (15 points) Consider a demand paging system with 1024-byte pages.



- 5 a) Complete the above page table for Process A.
- 5 b) If process A is currently running and the CPU generates a logical virtual address of 232_{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?

5 Cache of Page table entries in the CPU. It speeds virtual to physical addr. translation if the page table entry can be found in TLB and avoid access slow page table in memory.

Question 9. (8 points) There are many similarities between the cache-memory (RAM) level and memory-disk level of the memory hierarchy (i.e., memory acts as a cache of pages for the disk), but there are also important differences. A cache miss (i.e., access to a memory block not loaded into cache) stalls the running program temporarily, but a page fault (i.e., access to a page not loaded into memory) causes the running program to turnover the CPU to another program. Why are these cases treated differently by the computer system?

8 On cache miss, memory block just needs to be loaded $\approx 50\text{ ns}$.
 On page fault, page must be read from much slower disk $\approx 10,000,000\text{ ns}$, so we turn CPU over to another process.

Question 10. (9 points)

High-level for-loop	Assembly/Machine Language
for i := 0 to 100 do	LOAD_IMMEDIATE R3, #0
:	LOAD_IMMEDIATE R4, #100
:	FOR: BGT, R3, R4, END_FOR
:	:
end for	B FOR
	END_FOR:

If the above “for-loop” is executed on a pipelined computer with a branch-prediction buffer (BPB) with two-bits to dynamically predict the branch outcome, indicate whether each of the following statements is True or False.

a)	The BPB correctly predicts NOT TAKEN for the conditional branch (BGT) instruction for all but the first and last iteration of the loop.	<input checked="" type="radio"/> True <input type="radio"/> False
b)	The BPB correctly predicts TAKEN for the conditional branch (BGT) instruction for all but the last iteration of the loop.	<input type="radio"/> True <input checked="" type="radio"/> False
c)	The BPB correctly predicts NOT TAKEN for the unconditional branch (B FOR) instruction for all but the last iteration of the loop.	<input type="radio"/> True <input checked="" type="radio"/> False
d)	The BPB correctly predicts TAKEN for the unconditional branch (B) instruction for all but the first iteration of the loop.	<input checked="" type="radio"/> True <input type="radio"/> False
e)	As every instruction is fetched (F stage), the BPB is checked to see if the instruction is a known branch instruction and if it is what its prediction is (TAKEN or NOT TAKEN).	<input checked="" type="radio"/> True <input type="radio"/> False

f) WITHOUT a branch-prediction buffer (BPB) on our 5-stage pipeline what would be the total branch penalty for:

Conditional BGT instruction = $2 \times 1 = 2$ Unconditional B FOR instruction = $1 \times 101 = 101$
penalty + 1

Question 11. (15 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 R4, 16(R3)” loads R4 from the memory address specified by 16 + content in R3
- store instruction, e.g., “STORE R4, 8(R3)” stores R4 to the memory address specified by 8 + content of R3

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

Instructions	Time →																					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
ADD R3, R2, R1	F	D	E	M	W																	
LOAD R4, 16(R5)		F	D	E	M	W																
SUB R2, R6, R3			F	-	-	D	E	M	W													
LOAD R7, 8(R2)						F	-	-	-	D	E	M	W									
MUL R6, R7, R4										F	-	-	-	D	E	M	W					
STORE R6, 4(R5)														F	-	-	-	D	E	M	W	

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

Instructions	Time →																					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
ADD R3, R2, R1	F	D	E	M	W																	
LOAD R4, 16(R5)		F	D	E	M	W																
SUB R2, R6, R3			F	D	E	M	W															
LOAD R7, 8(R2)				F	D	E	M	W														
MUL R6, R7, R4					F	D	-	E	M	W												
STORE R6, 4(R5)						F	-	D	E	M	W											

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

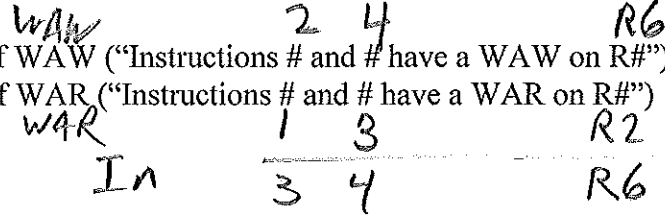
Question 12. (12 points) Superscalar processors with out-of-order execution introduce new data-dependencies:

- WAW: write-after-write
- WAR: write-after-read

- 4 a) Using the below code indicate an example of WAW ("Instructions # and # have a WAW on R#")
 4 b) Using the below code indicate an example of WAR ("Instructions # and # have a WAR on R#")

```

Instruction 1: DIV    R8, R2, R1
Instruction 2: MUL    R6, R4, R8
Instruction 3: ADD    R2, R6, R7
Instruction 4: SUB    R6, R2, R4
    
```



c) Rewrite the above code using register renaming to remove the WAW and WAR dependencies. You can just use letters (A, B, C, etc.) for new registers that you introduce when removing these dependencies.

```

Instruction 1: DIV    R8, R2, R1
Instruction 2: MUL    R6, R4, R8
Instruction 3: ADD    A, R6, R7
Instruction 4: SUB    B, A, R4
    
```

YOU HAVE A CHOICE FOR THE LAST QUESTION 13! DON'T DO BOTH ONLY ONE.

Question 13. (7 points) Explain how register renaming (e.g., question 12 above) enables a superscalar processor to achieve a higher level of instruction-level parallelism (ILP) within a program.

Reg. renaming "breaks" the dependency, so these instructions can be executed out of order.

Question 13. (7 points) The Intel x86 family of processors (including the Pentium IV discussed in class) starting in the early 70's. Since the idea of RISC had not been thought of yet, the x86 instruction set is a CISC (complex instruction set computer) design. Explain how the more modern Intel processors (like the Pentium IV, and later) in this family are able to execute x86 CISC programs and still take advantage of RISC ideas like pipelining and superscalar.

The CISC x86 instructions are dynamically translated at execution time to RISC type micro-operations that can be pipelined & "superscalared". The results of the out-of-order RISC execution is retired in CISC x86 order.