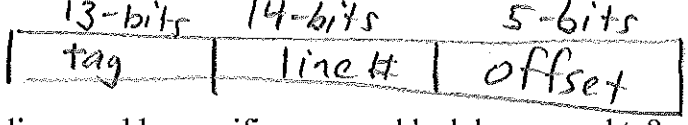
Computer Architecture Sample Test 1

Question 1. Suppose we have 32-bit memory addresses, a byte-addressable memory, and a 512 KB (2^{19} bytes) cache with 32 (2^5) bytes per block.

a) How many total lines are in the cache? $\frac{2^{19}}{2^5} = 2^{14}$ lines

b) If the cache is direct-mapped, how many cache lines could a specific memory block be mapped to? 1

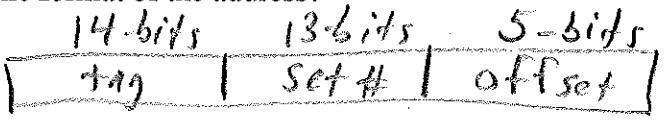
c) If the cache is direct-mapped, what would be the format (tag bits, cache line bits, block offset bits) of the address? (Clearly indicate the number of bits in each)



d) If the cache is 2-way set associative, how many cache lines could a specific memory block be mapped to? 2 sets of size 2

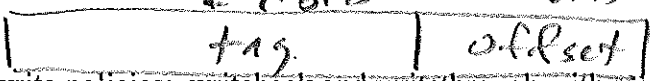
e) If the cache is 2-way set associative, how many sets would there be? $\frac{2^{14}}{2^1} = 2^{13}$ sets

f) If the cache is 2-way set associative, what would be the format of the address?



g) If the cache is fully-associative, how many cache lines could a specific memory block be mapped to? 2^{13} (any)

h) If the cache is fully-associative, what would be the format of the address?



Question 2. What is the difference between the cache write policies: writeback and writethrough with respect to each of the following:

a) the relative frequency of memory accesses

write back requires less memory accesses since we let it be out-of-date and only update the cache copy on a write.

b) the ability of a multiple CPU system to keep caches coherency ← Not on this test

write through helps since

Question 3. Assume that a program wants to copy a 500 word block of data from an I/O device to memory. Complete the following table for each I/O technique.

	Programmed I/O	Interrupt-driven I/O	Direct-memory Access
Total number of interrupts for whole block transfer	0	500	1
Total number of READ I/O commands issued by the CPU to the I/O module	500	500	1 for whole block

Question 7. You are to assume the same 5-stage pipeline discussed in class when answering these questions. Assume that the first register in an arithmetic operation is the destination register, e.g., in "ADD R3, R2, R1" register R3 receives the result of adding registers R2 and R1.

a. What would the timing be **without** bypass-signal paths/forwarding (use "stalls" to solve the data hazard)? (This code might require more or less that 15 cycles)

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

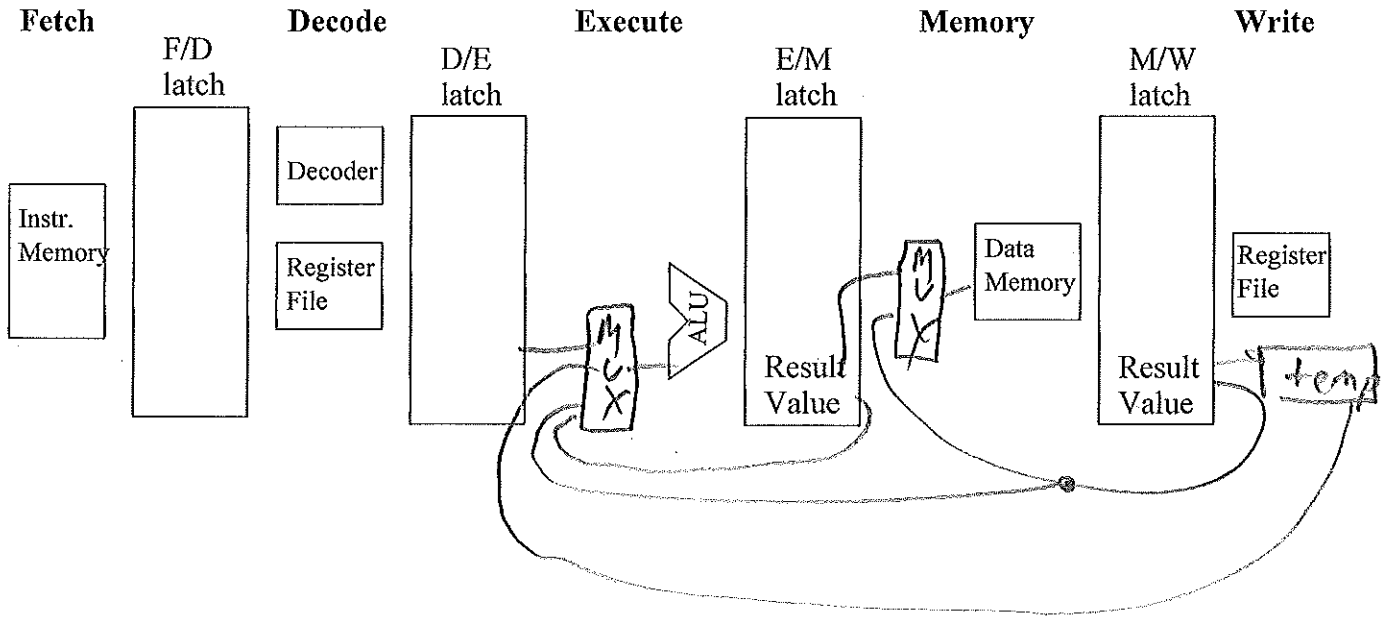
(Assume that a register **cannot** be written and the new value read in the same stage.)

b. What would the timing be **with** bypass-signal paths?
(This code might require more that 15 cycles)

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

(Assume that a register **cannot** be written and the new value read in the same stage.)

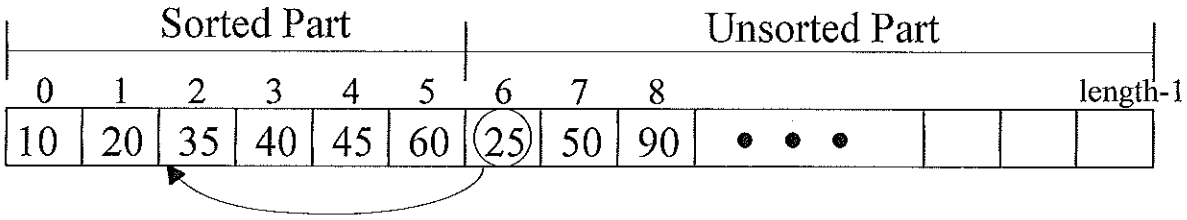
c. Draw ALL the bypass-signal paths needed for the above example.



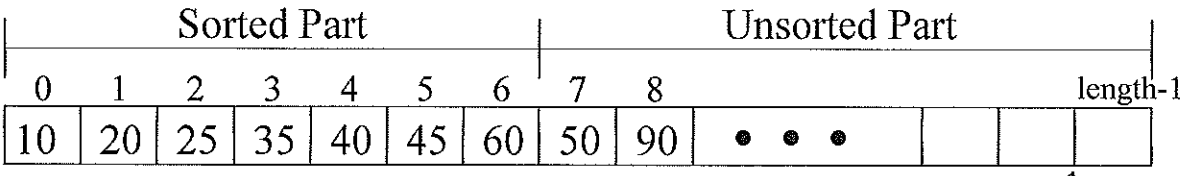
Question 8. Another simple sort is called insertion sort. Recall that in a simple sort:

- the outer loop keeps track of the dividing line between the sorted and unsorted part with the sorted part growing by one in size each iteration of the outer loop. (below the firstUnsortedIndex keeps track of the dividing line)
  - the inner loop's job is to do the work to extend the sorted part's size by one.

After several iterations of insertion sort's outer loop, an array might look like:



In insertion sort the inner-loop takes the "first unsorted item" (25 at index 6 in the above example) and "inserts" it into the sorted part of the array "at the correct spot." After 25 is inserted into the sorted part, the array would look like:



Consider the following insertion sort algorithm that sorts an array numbers:

```
InsertionSort(numbers - address to integer array, length - integer)
  integer firstUnsortedIndex, testIndex, elementToInsert;
  for firstUnsortedIndex = 1 to (length-1) do
    testIndex = firstUnsortedIndex-1;
    elementToInsert = numbers[firstUnsortedIndex];
    while (testIndex >=0) AND (numbers[testIndex] > elementToInsert ) do
      numbers[ testIndex + 1 ] = numbers[ testIndex ];
      testIndex =  testIndex - 1;
    end while
    numbers[ testIndex + 1 ] = elementToInsert;
  end for
end InsertionSort
```

a) Where in the code would unconditional branches be used and where would conditional branches be used?

b) If the compiler could predict by opcode for the conditional branches (i.e., select whether to use machine language statements like: "BRANCH_LE_PREDICT_NOT_TAKEN" or "BRANCH_LE_PREDICT_TAKEN"), then which conditional branches would be "PREDICT_NOT_TAKEN" and which would be "PREDICT_TAKEN"?

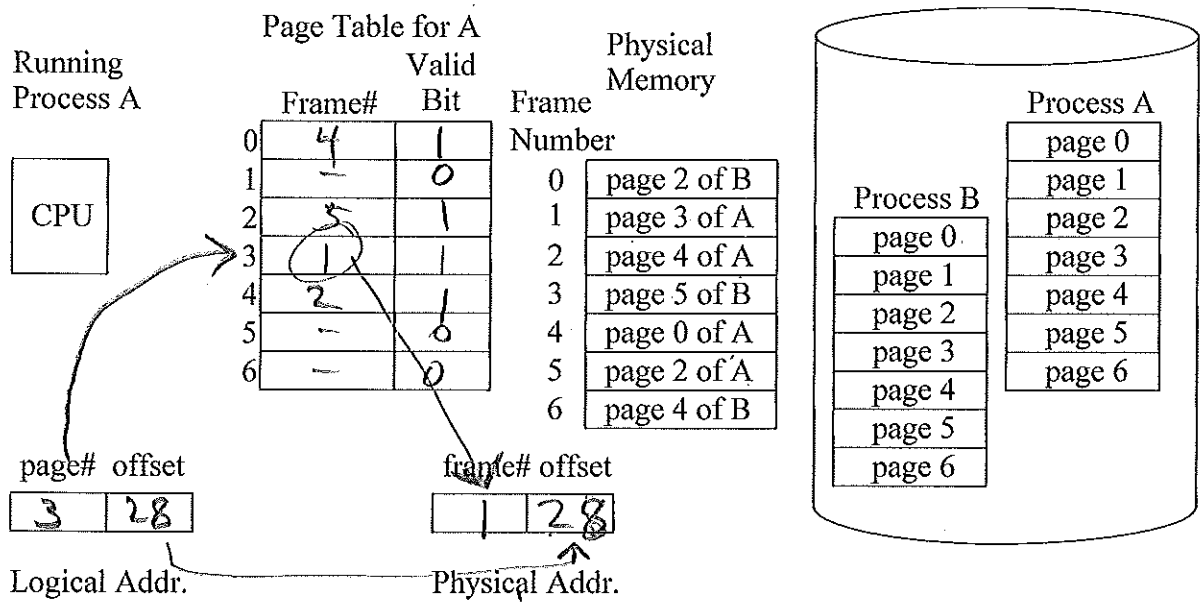
Question 4. Assume special I/O instructions (i.e., isolated I/O) are used to fill I/O-controller registers. Why can't a user program use these instructions to communicate with the I/O device directly and "by-pass" the operating system's protection checking?

Make I/O instructions privileged, so a user pgm cannot execute them.

Question 5. When a DMA controllers needs to use the bus to access memory, it performs "cycle stealing" by making the CPU wait if it is also trying to access memory. Even though the CPU is continually executing, why might the DMA device often find the bus to memory free when it needs to access memory?

The CPU is fetching and accessing data from its cache(s).

Question 6. Consider the demand paging system **with 1024-byte pages**.



a) Complete the above page table for **Process A**.

b) If process A is currently running and the CPU generates a logical/virtual address of 3100_{10}, then what would be the corresponding physical address?

page# offset

3 R 28

1024 | 3100
- 3072
28

1024
28
1052

4096 2048 1024 512 256 128 64 32 16 8 4 2 1
1 1 0 0 0 0 0 1 1 1 0 0

c) Assumptions:

- length = 100 and the numbers are initially in **descending** order before the insertion sort algorithm is called
- the five-stage pipeline discussed in class
- the outcome of conditional branches is known at the end of the E stage
- target addresses of all branches is known at the end of the D stage
- ignore any data hazards

Under the above assumptions, answer the following questions:

i) If fixed predict-never-taken is used by the hardware, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Here assume NO branch-prediction buffer)

for	while ① (testIndex>=0)	while ②	end while	endFor
2	2*99	0	1*(1+2+3+...+98+99)	1*99

ii) If a branch target buffer with one history bit per entry is used, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Assume predict-not taken is used if there is no match in the branch-prediction buffer) Explain your answer.

for	while ①	while ②	end while	endfor
2	2+4*98	0	1	1

2 first and 2 last execution of loop

only wrong once when not in BPB

1st execution not in BPB initial so 0 penalty, but wrong prediction when we dropout.

iii) If a branch target buffer with two history bit per entry is used, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Assume predict-not taken is used if there is no match in the branch-prediction buffer) Explain your answer.

for	while ①	while ②	end while	end for
2	2*99	0	1	1

only wrong when drop out since we only are wrong once when we drop out each time and never wrong twice to change prediction from NOT-TAKEN