Fall 2016: COMP 7300 Advanced Computer Architecture

Test 2 (100 pts)

Grading policy:

¼ Credit for correct answer only

¾ Credit for well written and solid justification/facts/arguments. **Show your work.**

For Test 2, we assume a CPU with a 32-bit address bus.

A) Cache

1) **(10 points)** We consider a 4-way associative 32 KB cache with 2 KB blocks. The CPU generates the address 0x3159A6.

a. **(3 points)** What is the block address (in Hexa **OR** binary) of Block ***B*** containing address 0x3159A6? = 11 0001 0101 | 1001 1010 0110

2 KB = 2^{11} — Block address | 11 bits offset

b. **(4 points)** What is the index where Block ***B*** will be stored in the cache?

Block Address = 11 0001 0101 | 1

Tag | 2 bit index

Number of sets = $\frac{32\,KB}{2\,KB \times 4\,ways} = 4\ set = 2^2$

Index = $(11)_2 = 3$

c. **(3 points)** What is Block B's tag (in Hexa **OR** binary)?

Tag is the remaining = 11 0001 010

(25 points) Virtual Memory: We consider a 32 bit address bus and a 1 MB physical memory. Page size is 32 KB. Assuming a validity bit, a dirty bit, and **one** reference bit.

a) (3 points) How many pages does the logical space have?

$\frac{\text{Logical Space}}{\text{Page Size}} = \frac{2^{32}}{2^{15}} = 2^{17}$

b) (3 points) How many entries does the page table have?

Number of entries = number of pages = 2^{17}

c) (3 points) How many frames does the physical space have?

$= \frac{\text{Physical space}}{\text{Frame size}} = \frac{PS}{\text{Page Size}} = \frac{1\,MB}{32\,KB} = \frac{2^{20}}{2^{15}} = 2^5$

d) (4 points) What is (in Kbytes) the size of the page table?

Each entry contains V + D + R + Frame number

Then Size = number of entries × 8 = $(1+1+1+5)\,2^{17}$ bytes = 128 Kbyte

e) Consider the partial page table below starting at entry 0. **Pay attention!!!**: the values in the page table **INCLUDE** (as most significant bits) the validity bit, the dirty bit, and the reference bit, respectively.

0	0x54
1	0xA5
2	0x61
3	0xB2
4	0x34
5	0x89
6	0x77
7	0xE3
.....	.....

a. **(12 points a+b)** The CPU generates the address 0x28854. Is this address is in the main memory? If yes, what is its physical address?

0x28854 = 10 1000 1000 0101 0100 (15 bits offset, Page size = 2^{15})

Page # is 5

entry is 0x89 = 1000 1001

V=1 then page is in main memory Physical A = 01001 offset 0x48854

b. The CPU generates the address 0x17874. Is this address is in the main memory? If yes, what is its physical address?

0x17874 = 1 0111 1000 0111 0100

Page # = 2 offset

entry(2) = 0x61 = 0110 0001

V=0 ⇒ the page is not in main memory

Exercise 1: (30 points) We assume that the data cache size is 4 KB (data) and the block size is 64 bytes. We assume that M[i][j] is adjacent to M[i][j+1] in the memory and that the cache is fully associative using LRU replacing. Consider the following code manipulation a matrix of **short** integers (2 bytes per short):

```
short A[64][64];
int i,j;
for (j = 0; j < 32; j++)
      for (i = 0; i < 64; i++){
            A[i][j] = A[i][j] ^ A[i][n-1-j];
            A[i][n-1-j] = A[i][j] ^ A[i][n-1-j];
            A[i][j] = A[i][j] ^ A[i][n-1-j];
      }
```

This code performs a SPECIAL column-wise transpose. As indicated on **Figure 2**, the transposition is performed as follows: the first column(j=0) and the last column (j=63) are swapped, the second column (j=1) and the column (j = 62) are swapped, and so on…

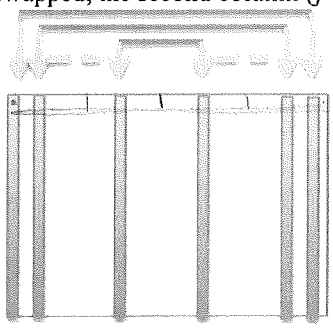


Figure 2

We assume that the code is in a separate instruction cache and the variables i and j are registers.

a) **(2 points)** How many blocks are needed to store one line of the matrix?

One line contains 64 shorts = 64 × 2 bytes = 128 bytes
One block is 64 bytes
⇒ one line take 2 blocks

b) **(2 points)** How many blocks are needed to store the full matrix?

One matrix has 64 lines = 64 × 2 blocks = 128 blocks

c) **(2 points)** How many blocks does the cache contain?

$= \frac{\text{Cache Size}}{\text{Block Size}} = \frac{4 \text{ KB}}{64} = \frac{2^{12}}{2^6} = 2^6$ blocks

d) **(6 points)** How many compulsory misses occur?

Since each block is referenced and no block was in the cache ⇒ number of compulsory misses is equal to the number of blocks in the matrix i.e 128 compulsory misses (see b)

e) **(4 points)** How many capacity misses occur?

Since ① the cache cannot contain all the matrix and ② each block is brought in back in the cache for each element.

⇒ number of capacity misses = Number of elements in matrix − the number of compulsory misses = 64×64 − 128 = $2^{12} - 128$

f) **(2 points)** How many conflict misses occur?

No conflict misses because cache is fully associative

g) **(5 points)** Would a loop interchange decrease the number of capacity misses? If yes, rewrite the loops with the loop interchange (**no need** to repeat the swapping instructions) and provide the new number of capacity misses.

Yes

```
for (i=0; i<64; i++){
    for (j=0; j<32; j++){
    }
}
```

Now, each time a block is brought in the cache, all elements are used and the block is never brought in back, then we will have only compulsory misses
⇒ no capacity misses

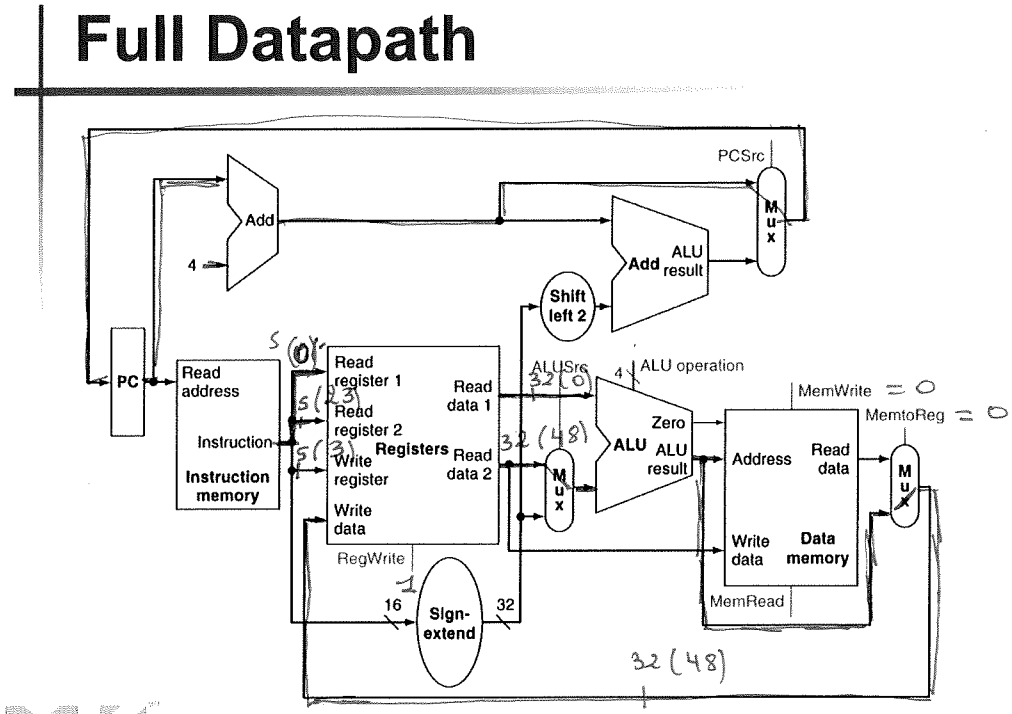
h) **(1 point)** What would be the number of compulsory misses with a loop interchange?

128 see e

i) **(6 points)** Using loop interchange, what should be the minimal size of the cache to completely eliminate the capacity misses?

This algorithm can work with no capacity misses if the cache can contain a full line
⇒ a minimal size is 2 x 64 bytes = 128 bytes

Exercise 2) (20 points)
Consider the datapath on this figure:



1) **(3 points)** We propose to implement the ***new*** instruction:

mov `$r1, $r2` // $r1 <- $r2 (moves content of register $r2 into register $r1)
Note that Register $0 cannot be modified because it must always contain the value 0.

Example: if $r1 = 8 and $r2 = 120, the CPU will store the value 120 in Register $r1 while Register $r2 remains unchanged after the execution of the instruction.

Propose an instruction format for ***mov*** (consistent with the instruction set defined so far. This means that the first 6 most significant bits are reserved for the opcode).

Opcode	$0	$r2	$r1	shamt	funct

2) **(1 point)** Does the full datapath support the *new* instruction? **Answer only Yes or No. (No justification is needed here)**. If the answer is Yes, jump to Question 4.

3) **(2 points)** If the answer to Question 2) is **NO**, add **on the figure** all **needed** lines, units, resources, inputs, outputs, multiplexers.... to support the *new* instruction.

4) **(6 points)** Based on the instruction code you proposed, **draw/highlight on the figure** the datapath used by the *new* instruction.

5) **(8 points)** After you draw the datapath, label the relevant buses with size and content for the instruction ***mov*** *$3, $23*. Provide the values of the control inputs. Assume that $23 = 48.

Programming: (15 points)

Write a program that inputs a 32 bit address (integer) and display its tag assuming. We consider a 4-way associative 32 KB cache with 2 KB blocks. Your program must:

1) prompt the user to input an integer (address)
2) compute the tag
3) display the tag

If you do not remember the system call number of some function, just use any number and comment it.

Grading guidelines: first mistake is forgiven, second **major** mistake costs 2 pts, 3rd (4 pts), 4th (6 pts)....

```
# Test 1 Code
.data
   prompt:      .asciiz "Enter an address : "
   theResult:   .asciiz "\n Tag is "



.text
```