Version II ②

Fall 2015: 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) **(8 points)** We consider a 2-way associative 32 KB cache with 4 KB blocks. The CPU generates the address 0x204865.

   a. **(3 points)** What is the block address (in Hexa **OR** binary) of Block ***B*** containing address 0x204865? 0x 20 48 65 = 1 0000 0100, 1000 0110 0101

   Block Address — 12 bits (offset)

   BA = 1 0000 0100 = 0x 2 04

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

   Number of blocks in cache = 32KB / 4KB = 8

   Number of sets = 8 / 2 (2-way assoc.) = 2^2 ⇒ index taken 2 bits

   BA = 10 0000 0100 (index)

   then Tag = 100000 01

   c. **(2 points)** What is the index where Block ***B*** will be stored in the cache?

   Based on b, index = 0

   ⇒ this block will be in set # 0

2) **(12 points)** Consider the cache on the figure below. Answer the following questions based on the figure below

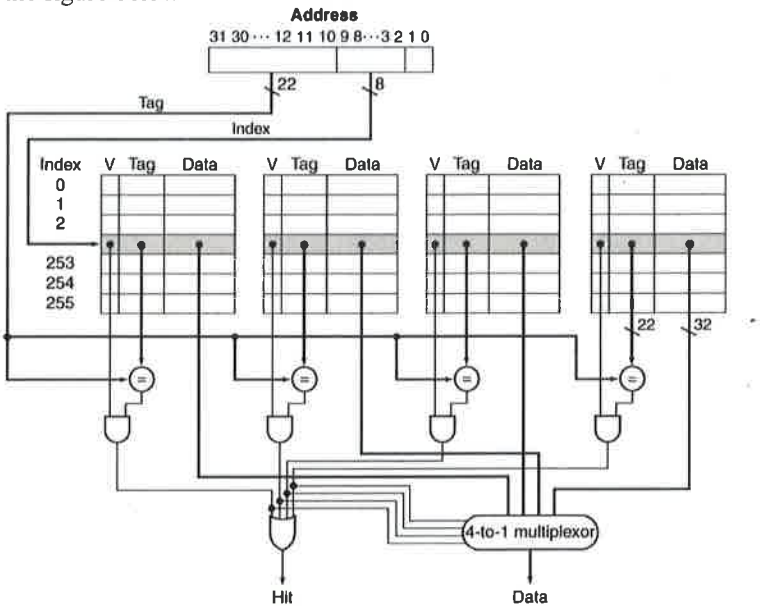


Figure 1

00

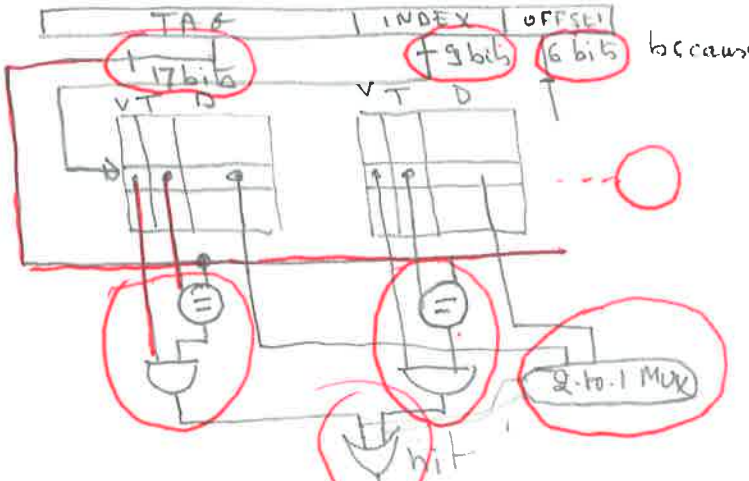
a. **(2 points)** Is the above cache 2-way associative cache? Explain why

No, there are four blocks per set (as shown on the figure).

b. **(10 points) Redraw** Figure 1 (above) such that the cache becomes 2-way associative with a block size of 64 bytes and a cache size of 64 KB of data. Just like for Figure 1, **indicate clearly the size** of the busses as well as the smallest and largest Indexes.

Number of blocks in cache = $\frac{64KB}{64} = 2^{10}$

Number of sets = $\frac{2^{10}}{2\ (2\text{ way as})} = 2^{9}$



-1.5 per item missing/added/wrong

Exercise 1: (30 points) We assume that the data cache size is 2 KB (data) and the block size is 32 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:

```
char 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…

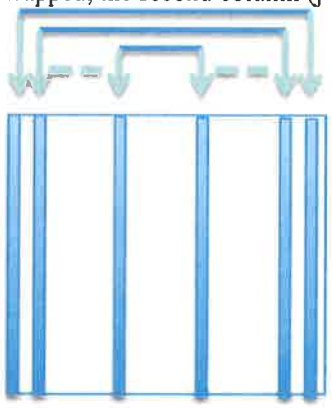


Figure 2

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

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

One line takes 64 characters = 64 bytes.

Number of blocks = $\frac{\text{Line Size}}{\text{block Size}} = \frac{64}{32} = 2$ blocks

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

The full matrix takes 64x64 bytes

Number of blocks = $\frac{\text{Matrix Size}}{\text{block Size}} = \frac{64 \times 64}{32} = 64 \times 2 = 2^7 = \boxed{128}$

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

Number of blocks in cache = $\frac{\text{Cache Size}}{\text{block Size}} = \frac{2\,KB}{32} = \frac{2^{11}}{2^5} = 2^6$

$= \boxed{64 \text{ blocks}}$

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

All blocks in the matrix will be accessed during the execution of the loop when j = 0 (access to the first column).

So, for each line, the CPU will have 2 compulsory misses (2 blocks per line).

⇒ total compulsory misses = 2 x number of lines

$= 2 \times 64 = 128$

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

After executing the first column, each time the CPU accesses an item, the item will not be in the cache because the cache can contain only half of the matrix ⇒ each swap will provoke 2 capacity miss

There are 64 swaps per column and 31 columns are swapped (after the first column) ⇒ Number of cap. miss = 64x31x2

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

The cache is associative ⇒ # of conflict misses = 0

g) **(6 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++){
        body unchanged
    }
```

whenever a line is brought in the cache, it is fully used and is never accessed again.
Therefore, there are no more capacity misses

h) **(2 points)** What would be the number of compulsory misses with a loop interchange?

Compulsory misses shall occurs because the blocks must be brought the first time ⇒ # of compulsory misses remains the same = 128

i) **(4 points)** What should be the minimal size of the cache if we wanted to use blocking?

The cache must at least contain one full line to avoid capacity misses
⇒ the cache should at least contain 2 blocks
= 2 x 32 = 64 bytes

⇒

B) Virtual Memory (15 points)

We consider a 32-bit address bus and a 2 MB physical memory. Page size is 32 KB. Assuming a validity bit and a dirty bit, the objective is to compute the size of the table.

a) **(2 points)** How many entries does the page table have?

$$\#\text{ of entries} = \#\text{ of pages} = \frac{2^{32}}{32KB} = \frac{2^{32}}{2^{15}} = 2^{17}$$

$$= \frac{\text{Logical Space}}{\text{Page Size}}$$

b) **(2 points)** How many frames are in the main memory?

$$\#\text{ of frames} = \frac{\text{Physical Spaces}}{\text{Frame Size}} = \frac{2\,MB}{32\,KB} = \frac{2^{21}}{2^{15}} = 2^{6}$$

c) **(4 points)** What is the size of each entry (page table)?

Each entry takes one validity bit, one dirty bit and a frame #. The frame # takes 6 bits because there are 2^6 frames in the physical space

⇒ an entry will take $1+1+6 = 8$ bits

d) **(3 points)** What is then the size of the page table?

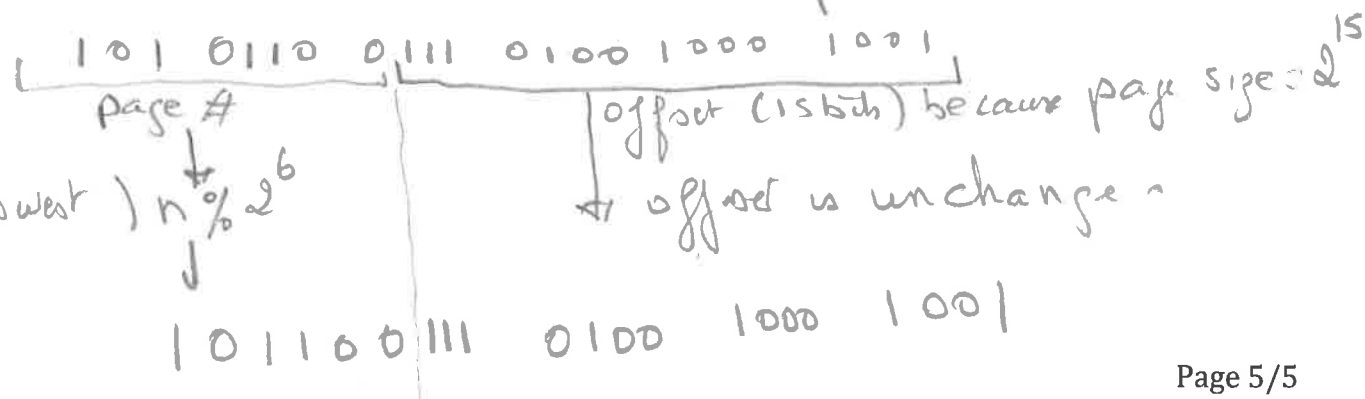
Page table Size = # of entries × size of an entry

$= 2^{17} \times 8\,bit = 2^{17}$ bytes

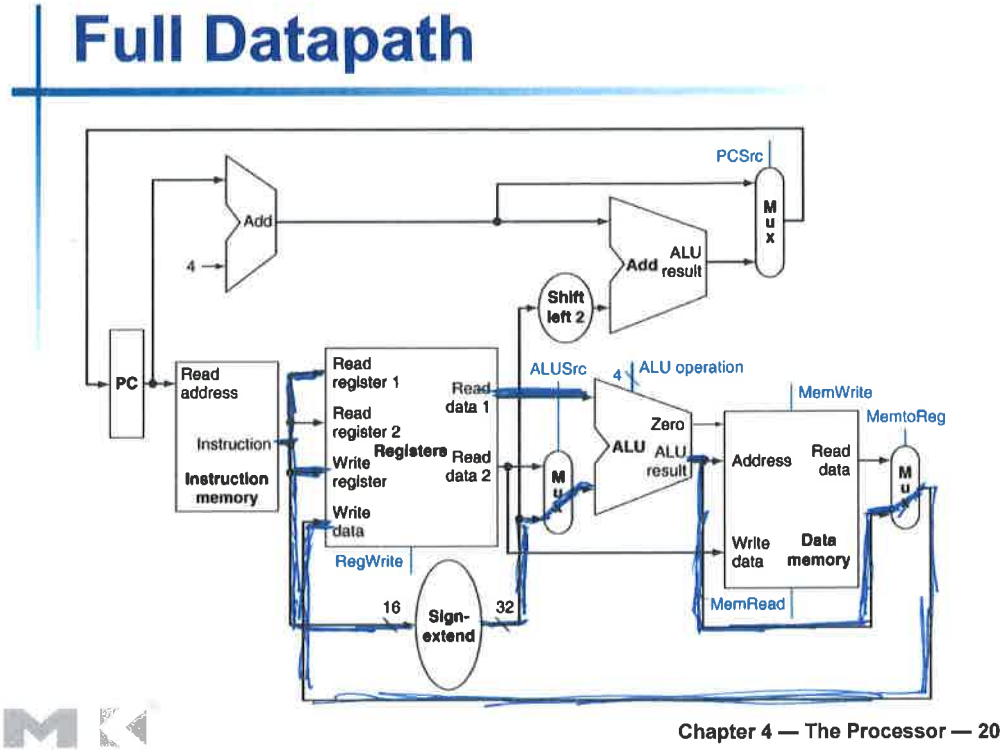
$= 2^{7}$ KB

$= 128$ KB

e) **(4 points)** Suppose that physical memory is managed like direct-mapped cache: each page with number i is stored in the frame with number n % 2^6. What is in hexadecimal the physical address corresponding to the virtual (logical address) 0x567489?



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



1) **(2 points)** Consider the following instruction:

subw $r1, Constant($r2) $r1 <- Constant - $r2 (Constant is 16 bits wide)

Example: if $r2 = 8 and Constant = 120, the CPU will compute the difference (120 - 8) and will store it in Register $r1, i.e $r1 = 112 after the execution of the instruction.

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

6 bit	5 bit	5 bit	16 bits
Opcode	$r1	$r2	CONSTANT

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

YES

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 instruction *subw*.

No need of additional element

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

5) **(8 points)** After you draw the datapath, label the relevant buses with size and the content for the instruction ***subw $3, 250($23)***. Assume that $3 = 32, $23 = 48.

6) **(1 point)** Provide the final values of the registers $3 and $25.

Exercise 3: (15 points) Deriving the pipeline speed up
Consider a monocycle CPU with 4 operations (stages) that take, 200ps, 200ps, 200ps, and 400ps.

1) **(1 point)** What is the latency of one instruction on the monocycle CPU?

Latency = sum of all stages = 200 ps + 200 ps + 200 ps + 400 ps
= 1000 ps = 1 ns

2) **(1 point)** Consider a program P with n instructions. What is the execution time for Program P (expression as a function of n) on the monocycle CPU?

Execution time = number of instruction X Execution time of one inst
= n X 1000 ps = n X 1 ns

3) **(11 points)** We want to pipeline the CPU above with one stage per operation. We neglect the buffer time between stages. All the questions below apply to the pipeline

.

a) **(3 points)** What should be the clock frequency for the pipelined CPU?

The clock cycle must accodommate the longest stage, i.e 400ps
⇒ Clock Rate = $\frac{1}{\text{cc duration}} = \frac{1}{400ps} = 2.5\ 10^9/s = 2.5\ GHz$

b) **(3 points)** What is the latency for one instruction for the pipelined CPU?

Latency = number of stages x duration of one stage
= 4 X 400 ps = 1,600 ps = 1.6 ns

c) **(2 point)** What is the throughput (bandwidth) in MIPS of the pipelined CPU?

Throughput = $\frac{\text{Number of instruction for}}{\text{Time}} = \frac{1}{400ps} = \frac{1000}{400}\ 10^9\ Ins/s$
= 2,500 Inst/s.
= 2500 MIPS

d) **(3 points)** Consider a program P with n instruction. What is the execution time for Program P? (Make sure to take into account the phase to fill up the pipeline and wind it up)

Instruction 1 ends after 4 cycles
2 ——— 5
i ——— i+3

n instructions will take (n+3) clock cycles

4) **(2 points)** What is the expression of the speed up (monocycle versus pipelined)? If n tends to infinity, what is the speed up?

Speed up = $\frac{\text{Old time}}{\text{new time}} = \frac{n \times 1\ ns}{(n+3) \times 400\ ps} = \frac{n \times 1000\ ps}{n \times 400ps + 1200ps} = \frac{1on}{4n+12}$

$\lim_{n \to \infty}$ speed up = $\frac{10}{4} = 2.5$