Version I ①

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 0x568402.

   a. **(3 points)** What is the block address (in Hexa **OR** binary) of Block ***B*** containing address 0x568402?

   0101 0110 1000 0100 0000 0010
   Block address | offset

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

   Number of blocks = 32 KB / 4 KB = 8 blocks
   Number of sets = 8/2 (2 ways associative) = 4
   Block Address = 01 01 01 10 10 00
   Tag | index
   Tag = 01 01 01 10 10 = 0x15A

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

   Based on b, index = 0
   ⇒ the block will be stored in set # 0

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

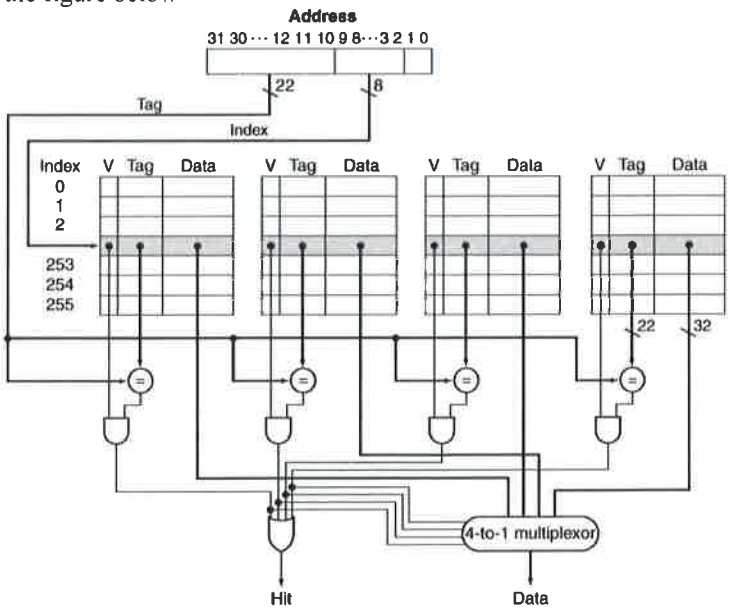


Figure 1

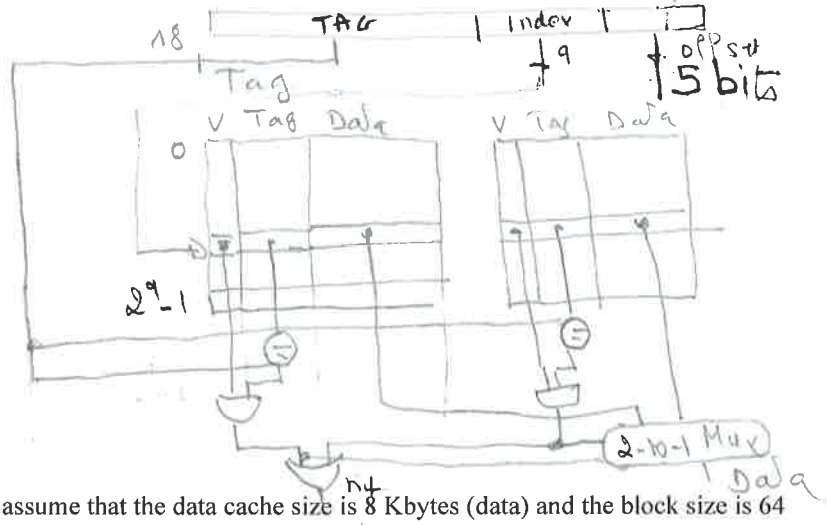
a. **(2 points)** Is the above cache 2-way associative cache?

no, ; There are four blocks for each set (as shown on figure)

b. **(10 points) Redraw** Figure 1 (above) such that the cache becomes 2-way associative with a block size of 32 bytes and a cache size of 32 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 = $\frac{32\text{ kb}}{32} = 2^{10}$

Number of sets = $\frac{2^{10}}{2} = 2^9$



Exercise 1: (30 points) We assume that the data cache size is 8 Kbytes (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:

```
char A[128][128];
int i,j;
for (j = 0; j < 64; j++){
      for (i = 0; i < 128; 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=127) are swapped, the second column (j=1) and the column (j = 126) are swapped, and so on…

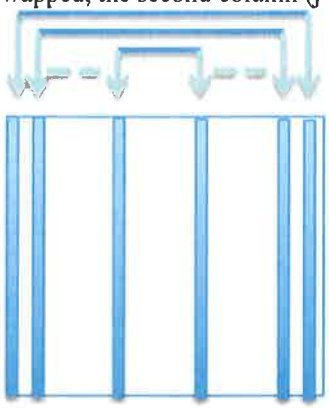


Figure 2

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 128 chars = 128 bytes

Number of blocks in a line = $\frac{\text{Line Size}}{\text{block Size}} = \frac{128}{64} = \boxed{2}$

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

The matrix has 128 lines

Number of blocks in Matrix = Number of blocks per line * # of lines

$= 2 \times 128 = \boxed{256}$

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

Number of blocks in cache = $\frac{\text{Cache Size}}{\text{Block Size}} = \frac{8KB}{64} = \frac{2^{13}}{2^6} = 2^7$

$= \boxed{128}$

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 first column)

So for each line, we will have 2 compulsory misses (2 blocks/line)

⇒ Total compulsory misses = 2 x number of lines

$= \boxed{256}$

e) **(4 points)** How many capacity misses occur? (j=0)

After executing the 1st column, each time CPU will access an item, the item will not be in the cache because the cache can contain only half the matrix ⇒

So each swap will provoke 2 capacity misses

There are 128 swaps per column and 63 columns are swapped (after the first column) ⇒

Number of capacity misses = $128 \times 63 \times 2$

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 code with the loop interchange and provide the new number of capacity misses. yes

```
for (i=0; i<128; i++){
    for (j=0; j<64; j++){
        body unchanged
    }
}
```

Whenever a line is brought in the cache, S 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 still occur because the blocks must still be brought the first time

⇒ Compulsory misses remain unchanged = 256

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 64 = 128 bytes

B) Virtual Memory (15 points)

We consider a 32-bit address bus and a 1 MB physical memory. Page size is 16 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?

$$\#\text{ of entries} = \#\text{ of pages} = \frac{2^{32}}{2^{14}} = 2^{18}$$

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

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

$$\#\text{ of frames} = \frac{\text{Physical space Size}}{\text{Frame Size}} = \frac{1\,MB}{16\,KB} = \frac{2^{20}}{2^{14}} = 2^6$$

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

Each entry contains 1 validity bit, a dirty bit and a frame number. A frame # takes 6 bits because # of frames = 2^6

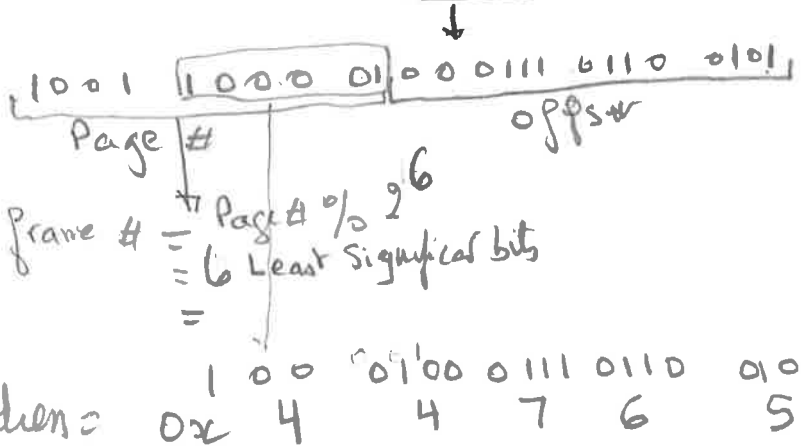
Size of an entry = 1 + 1 + 6 = 8 bits

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

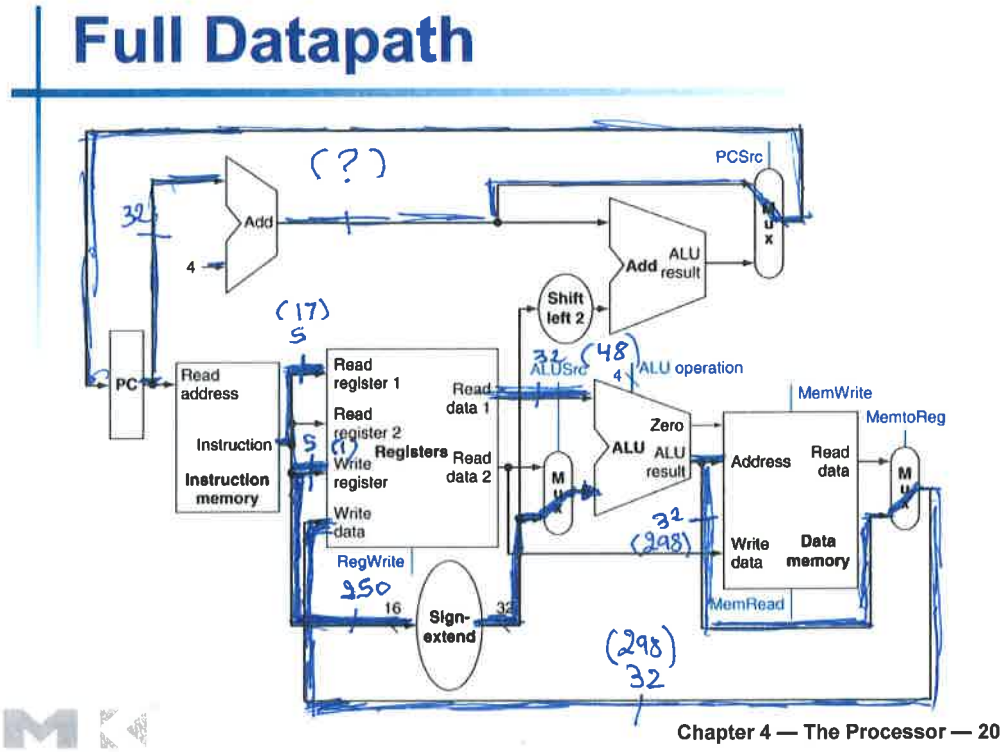
Page Table Size = # of entries x entry size

$= 2^{18} \times 8$ bits $= 2^{18}$ bytes.

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 the physical address corresponding to the virtual (logical address) 0x984765?.



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



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

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

Example: if $r2 = 8 and Constant = 120, the CPU will compute the sum (120 + 8) and will store it in Register $r1, i.e $r1 = 128 after the execution of the instruction.

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

6 bits	5 bits	5 bits	16 bits
Opcode	r1	r2	Constant

2) **(1 point)** Does the full datapath support the *addw* 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 *addw*.

No need of any additional element

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

See figure

5) **(8 points)** After you draw the datapath, label the relevant buses with size and the content for the instruction ***addw*** *$1, 250($17)*. Assume that $1 = 32, $17 = 48.

See Figure

6) **(1 point)** Provide the final values of the registers $1 and $17.

$1 = 298 $17 = 48

Exercise 3: (15 points) Deriving the pipeline speed up

Consider a monocycle CPU with 4 operations (stages) that take, 100ps, 100ps, 100ps, and 200ps.

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

Latency = 100ps + 100ps + 100ps + 200 ps = 500 ps

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 instructions * execution time of one instruction

= n * 500 ps = 500 n ps

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 frequency should set such that a clock cycle duration is equal of the execution time of the longest stage

Clock frequency = $\frac{1}{T_{max}} = \frac{1}{200\text{ ps}} = \frac{1000}{400}$ GHz = 5 GHz

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

Each instruction needs the four stages to execute, and each stage takes on the pipelined 400ps

⇒ Latency = 4 x 200ps = 800ps =

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

Each clock cycle, an instruction will be executed

⇒ Throughput = 5 10^9 instruct/s = 5 000 10^6 ins/s = 5,000 MIPS

because of the clock frequency: 5 GHz

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 takes 4 cycles
2 5
i i+3
n

⇒ n instructions will take (n+3) clock cycles

= (n+3) x 200 ps

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 = $\frac{\text{Old time}}{\text{New time}} = \frac{500\,n\text{ ps}}{(n+3)\times 200\text{ ps}} = \frac{500\,n}{400\,n + 600}$

$\lim_{n\to\infty}$ Speed up = $\frac{500}{200} = \frac{5}{2} = 2.5$