University of Algiers 1

Faculty of Sciences
Department of Computer Science

Final Exam L2-ADO

29-jan-2025

Duration : 90 minutes

Surname: First name: Group: Student ID No:	/ 20

Exercise 1: (6 points)

A system has a main memory of 16 megabytes and a 32 kilobytes direct-mapped cache with 8 bytes per block.

1/ How many lines are there in the cache memory?

(32*1024 bytes / cache)/(8 bytes/block) = 4*1024 blocks **(1pt. missing details = no mark)**

2/ Show how the main memory address is decomposed (i.e. give the widths of the T/I/O fields).

(1pt. missing details = no mark)
16 MB of bytes (2^{24} = 16 MB) implies a 24-bits address is needed – From question 1/ we have 12 bits for the index field ($2^2 2^{10}$=2^{12}), a block of 8 bytes require 3 bits for the offset and we are left with 24-12-3 =9 bits for the tag

Assume a 2-way set associative cache (i.e. there are two blocks per cache line). Assume also a 14-bit tag, 8 bits for the set line, and 2 bits for the offset.

3/ How many blocks are contained in this cache?

a) 8 b) 256 c) 512 d) 1024 e) 16K

(1pt. wrong/missing details = -1pt)
There are eight bits identifying the set, thus there are 2^8 = 256 sets. Since there are two blocks per set, then there are 2×256 = 512 blocks.

4/ How many blocks are there in RAM according to the cache system defined above?

a) 2^{24} b) 2^{22} c) 2^{14} d) 2^8 e) Cannot be determined

(1pt. wrong/missing details = -1pt)
2^{22} blocks. Memory address width is 24 bits seen as block bits and offset bits. Taking out the 2 bits for the offset id, we are left with 22 bits for addressing the blocks.

The timing for a cache system is as follows: checking the cache requires 1 cycle. On a cache hit, the data is delivered to the CPU by the end of the first cycle. On a cache miss, an additional 9 cycles are needed to fetch the word from the main memory, storing it in the cache, and returning a copy to the CPU.

5/ To achieve an average memory access time (AMAT) of 1.4 cycles, what is the minimum cache hit ratio required?

1.4 = 1+(1-hit ratio)*(9+1) **(1pt. missing details = no mark)**
0.4 = (1-hit ratio)*10 >> minimum possible value of hit ratio is 0.96.

6/Consider a set-associative cache of 2KB (1KB=2^{10} bytes) with a cache block size of 64 bytes. Assume a 32-bit address is used for accessing the cache. If the width of the tag field is 22 bits, the associativity of the cache is __2__.

a) 1 b) 2 c) 4 d) 8 e) 16

offset bits: log2(64)=6 bits. index bits: 32-22-6 = 4 (16 total sets). **(1pt. wrong/missing details = -1pt)**
Total cache size 2KB divided into 16 sets we get 128 bytes per set. Each cache line has 64 bytes. So, it is a 2-set associative

Exercise 2: (8.5 points)

Consider the following MIPS code:

```
1:    add   $a0, $at, $0
2:    sub   $t1, $v1, $a0
3:    add   $a0, $a1, $a2
4:    lw    $v0, 100($v1)
5:    lw    $v0, 0($v0)
6:    sw    $v0, 100($a0)
7:    and   $v0, $v0, $at
8:    beq   $t1, $at, loop
9:    and   $t1, $t1, $at
```

1/ Fill in the table below assuming a "five-stage" pipelined data path with a “data forwarding unit”, “double-pumping” and a “prediction unit” that assumes branches will not be taken. Indicate, if applicable, “data forwarding” occurrences with arrows, “pipeline stalls” with “nop” bubbles, and double-pumping usage with circles **(3.5 pts)**

Inst. \ Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`add $a0, $at, $0`	IF	ID	EX	~~MEM~~	WB													
`sub $t1, $v1, $a0`		IF	ID	EX	~~MEM~~	WB												
`add $a0, $a1, $a2`			IF	ID	EX	~~MEM~~	WB											
`lw  $v0, 100($v1)`				IF	ID	EX	MEM	WB										
`lw  $v0, 0($v0)`					IF	ID	nop	EX	MEM	WB								
`sw  $v0, 100($a0)`						IF	nop	ID	nop	EX	MEM	~~WB~~						
`and $v0, $v0, $at`								IF	nop	ID	EX	~~MEM~~	WB					
`beq $t1, $at, loop`										IF	ID	EX	~~MEM~~	~~WB~~				
`and $t1, $t1, $at`											IF	ID	EX	~~MEM~~	WB			

2/ List all possible data hazards in the code (i.e. affected instructions and registers)

(2 pt. missing details = no mark)

between inst. 1 and 2 (`1.rd = 2.rt = $a0`)
between inst. 4 and 5 (`4.rt = 5.rs = $v0`)
between inst. 5 and 6 (`5.rt = 6.rt = $v0`)
between inst. 5 and 7 (`5.rt = 7.rs = $v0`) – Resolved by the double-pumping feature.

3/ Apply code rescheduling to reduce as much as possible the number of stalls

(1 pt. missing details = no mark)

The stall on `$v0` between instructions 4 and 5 can be eliminated if inst. 4 is scheduled before inst. 3 in the pipeline. This gives: 1 , 2 , **4** , 3 , **5** , **6** , 7 , 8 , 9 (will eliminate a single stall between inst. 4 and 5). In fact, we need to sandwich two statements between instructions 4, 5 and 6 in order to avoid stalling the pipeline. The new order of execution would be: 1 , **4** , 2 , **5** , 3 , **6** , 7 , 8 , 9 (will remove all the stalls).

4/ Indicate which register values are forwarded when handling data hazards for the rescheduled code

(2 pts. missing details = no mark)

```
$a0: inst. 1 -> inst. 3 (old inst. 2),
$v0: inst. 2 (old inst. 4) -> inst. 4 (old inst. 5),
$v0: inst. 4 (old inst. 5) -> (inst. 6).rt,
$a0: inst. 5 (old inst. 3) -> (inst. 6).rs
```

Exercise 3: (5.5 points)

a. Write in hexadecimals the machine codes associated with the following MIPS instructions

```
1/ sllv $t3, $t1, $t0
```

(1 pt. missing details = no mark)

R type. Op = 0, rt = 9 ($t1), rs = 8 ($t0), rd = 11 ($t3), sh = 0, func = 0x4 => 0x01095804
0.25pt … 0.5 pt … 0.25 pt (only if details are correct)

```
2/ jr $t2
```

(1 pt. missing details = no mark)

R type. Op = 0, rs = 10 ($t2), rt = 0, rd = 0, sh = 0, func = 0x8 => 0x01400008
0.25pt … 0.5 pt … 0.25 pt (only if details are correct)

```
3/ andi $t2, $t1, 0x0671
```

(1 pt. missing details = no mark)

I type. Op = 0xC, rt =10 ($t2), rs = 9 ($t1), Imm = 0x0671 => 0x312A0671
0.25pt … 0.5 pt … 0.25 pt (only if details are correct)

b. If the first statement below is located at memory address 0x4000, give the correct instruction offset for the following branch instructions

```
A: beq $t0, $t1, C
B: beq $t1, $t2, C
C: beq $t2, $t3, A
D: beq $t3, $t4, C
E: beq $t4, $t5, E
```

(2.5 pt. missing details = no mark)

```
A=0x4000 , B=0x4004 , C=0x4008 , D=0x400C , E=0x4010

beq $t0, $t1, C   => (0x4008 - (0x4000 + 4)) / 4 =  1
beq $t1, $t2, C   => (0x4008 - (0x4004 + 4)) / 4 =  0
beq $t2, $t3, A   => (0x4000 - (0x4008 + 4)) / 4 = -3
beq $t3, $t4, C   => (0x4008 - (0x400C + 4)) / 4 = -2
beq $t4, $t5, E   => (0x4010 - (0x4010 + 4)) / 4 = -1
                          … 0.25pt …          0.25pt (if details are correct)
```