Computer Science and Engineering Department
Computer Architecture

Ordinary Exam
January 23th, 2015

ATENCIÓN:

- Please read carefully all the questions before starting.
- Books, notes and calculators are not allowed.
- Mobile phones must be disconnected during the test (disconnected, not muted).
- Only answers in ink will be corrected. Please, do not use pencil.
- **Time limit is 150 minutes.**
- **Give every exercise in separate sheets. If you do not answer an exercise, please give a blank sheet with the exercise number.**

NAME:
SURNAME:
NIA:

Exercise 1 [2 points]:

The following code fragment is stored starting from memory address 0x1000100C in a machine where all instructions occupy 4 bytes:

```
loop:   lw $r2, 0($r0)
        addi $r3, $r2, 20
        sw $r3, 0($r1)
        addi $r0, $r0, 4
        addi $r1, $r1, 4
        bnez $r2, loop
```

This code runs in a machine with a L1 data cache which is 2 ways set-associative and with a size of 32 KB and a L1 instruction cache with the same characteristics. It also has a L2 unified cache which is 8 ways set-associative with a size of 1MB. In both cases the line size is 32 bytes. It is assumed that a cache hit in L1 requires 4 cycles, a cache hit in L2 requires 14 additional cycles, and penalty for bringing a memory block from main memory to L2 is 80 cycles. All caches have a write-back policy.

Initially, value of registers is:

- $r0: 0x00010000
- $r1: 0x00080000

Starting from location 0x00010000 all the values in memory are different from zero until location 0x000100FC. In memory location 0x000100FC there is a zero value.

1.1: Determine which should be the average access time assuming that a program (different from the one above) performs on average 2 data accesses per instruction and has the following miss rate:

- L1 instructions: 10%
- L1 data: 5%
- L2: 2%

1.2: Determine the number of misses produced during the execution of the provided code fragment for data L1 cache, instruction L1 cache, and L2 cache.

1.3: Prepare a time diagram for a MIPS architecture with a 5-stage pipeline, for the first loop iteration assuming that initially there are no data and no instructions in caches and with the following considerations:

- There is no forwarding hardware.
- Architecture allows that an instruction writes a register and another instruction reads that same register without problems.
- Branches are handled flushing the pipeline.
- Effective branch addresses are computed in the execution stage.

NOTE: Keep in mind when preparing the diagram the stalls due to misses in cache hierarchy for instructions (stage IF) as well as for data reads and writes (stage M).

1.4: Repeat the time diagram for the second iteration.

ANSWER

1.1: For level 1 cache, 2 data accesses are performed for every instruction access. Thus miss ratio is obtained as a weighted average:

m(L1) = (m(L1I)+2*m(L1D))/3

m(L1) = (0.1 + 2 * 0.05) / 3 = 0.2/3 = 0.0667

And average access time would be:

T = 4 + m(L1) * (14 + m(L2) * 80) = 4 + 0.0667 * (14 + 0.02 * 80) = 5.04 cycles

1.2: The loop is run a total of $2^8/4 = 2^6$ = 64 iterations.

Let's analyze separately instructions and data accesses.

First instruction is stored at address 0x1000100C. Last address is stored at address 0x1000100C + (6-1) * 4 = 0x10001020.

On first iteration, the first instruction gives a cache miss and brings block in addresses 0x10001000 – 0x1000101F, that contains the address of interest. The following instructions (I2, I3, I4 and I5) give hits. Finally, instruction I6 again gives a miss. Thus, first iteration gives 2 misses and 4 hits. The rest of iterations give hits for every access.

As the loop is run 64 times, access to instructions gives the following:

- Misses L1I: 2
- Hits L1I: 4 + 63 * 6
- Misses L2: 1

- Hits L2: 0

On each loop iteration, an address is read from the range 0x00010000 – 0x000100FC. This corresponds to $2^8/2^5 = 8$ cache lines. Values are written on range 0x00080000 – 0x000800FC, which are written in 8 cache lines. As caches are set associative no conflict may happen between read and written data. As no data is removed from L1 cache there are no writes in L2 cache.

- Misses L1D: 8 reads + 8 writes
- Hits L1D: 56 reads + 56 writes
- Misses L2: 8 reads
- Hits L2: 0

1.3: We number instructions as follows:

```
loop:  lw $r2, 0($r0)        #1
       addi $r3, $r2, 20     #2
       sw $r3, 0($r1)        #3
       addi $r0, $r0, 4      #4
       addi $r1, $r1, 4      #5
       bnez $r2, bucle       #6
```

The following RAW dependencies are found:

- $r2: I2 -> I1
- $r3: I3 -> I2
- $r0: I4 -> I1
- $r1: I3 -> I5

As there is no forwarding, when there is a RAW dependency, the WB stage must be waited:

Instr.	1-98	99	100	101	102	103-198	199	200	201	202	203	204	205
I1: lw $r2, 0($r0)	IF	ID	EX	M	M	M	WB						
I2: addi $r3, $r2, 20		IF1	IF2	IF3	IF4	stall	ID	EX	M	WB			
I3: sw $r3, 0($r1)							IF1	IF2	IF3	IF4	ID	EX	M
I4: addi $r0, $r0, 4											IF1	IF2	IF3
I5: addi $r1, $r1, 4													

Instr.	206	207	208	209	210	211	212	213	214	215-302	303	304	305
I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20													
I3: sw $r3, 0($r1)	M	M	M	M	M	M	M	M	M	M	WB		
I4: addi $r0, $r0, 4	IF4	ID	EX	Stall	Stall	Stall	Stall	Stall	Stall	Stall	M	WB	
I5: addi $r1, $r1, 4		IF1	IF2	IF3	IF4	ID	stall	stall	stall	stall	EX	M	WB
I6: bnez $r2, bucle						IF	IF	IF	IF	IF	IF	IF	IF
I7:¿?													
I1: lw $r2, 0($r0)													

Instr.	306	307	308	309	310	311	312						
I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20													
I3: sw $r3, 0($r1)													
I4: addi $r0, $r0, 4													
I5: addi $r1, $r1, 4													
I6: bnez $r2, bucle	IF	IF	IF	ID	EX	M							
I7: ¿?				IF	flush								
I1: lw $r2, 0($r0)						IF							

- First instruction stalls 98 cycles (80+14+4) in fetch as there is a miss in all the memory hierarchy.
- Data read in first instruction is a read miss and requires 98 cycles too.
- Second instruction is a hit and requires 4 cycles to perform fetch from L1I.
- Instruction I3 is a hit and requires 4 cycles to perform fetch from L1
- Data write in instruction I3 is a write miss and requires 98 cycles.
- Instruction I4 is a hit and requires 4 cycles to perform fetch from L1I.
- Instruction I4 cannot start its memory stage until memory access inf I3 has not finished.
- Instruction I5 is a hit and requires 4 cycles to perform fetch from L1I
- Instruction I5 cannot start its execution cycle until execution from I4 has not finished.
- Instruction I6 is a miss and requires 98 cycles to perform fetch from L1I.
- Instruction I7 (following bnez) cannot start fetch until fetch unit has not been released.
- Although branch target address is known at the end of I6 decode stage, branch outcome (taken or not taken) is not known until the end of execution stage.

In total 310 cycles are required.

1.4: In this iteration all accesses to instructions are hits. The same applies to data accesses.

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: lw $r2, 0($r0)	IF	IF	IF	IF	ID	EX	M	M	M	M	WB		
I2: addi $r3, $r2, 20					IF	IF	IF	IF	Stall	Stall	ID	EX	M
I3: sw $r3, 0($r1)											IF	IF	IF
I4: addi $r0, $r0, 4													
I5: addi $r1, $r1, 4													

Instr.	14	15	16	17	18	19	20	21	22	23	24	25	26
I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20	WB												
I3: sw $r3, 0($r1)	IF	ID	EX	M	M	M	M	WB					
I4: addi $r0, $r0, 4		IF	IF	IF	IF	ID	EX	M	WB				
I5: addi $r1, $r1, 4						IF	IF	IF	IF	ID	EX	M	WB
I6: bnez $r2, bucle										IF	IF	IF	IF

Instr.	27	28	29	30									
I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20													
I3: sw $r3, 0($r1)													
I4: addi $r0, $r0, 1													
I5: addi $r1, $r1, 1													
I6: bnez $r3, bucle	ID	EX	M	WB									
I7: ¿?	IF	flush											
I0: lw $r2, 0($r2)			IF										

In total 28 cycles are required.

Computer Science and Engineering Department
Computer Architecture

Ordinary Exam
January 23th, 2015

Exercise 2 [3 points]:

Given the following code fragmens executed by 3 threads using the coherence protocol MSI:

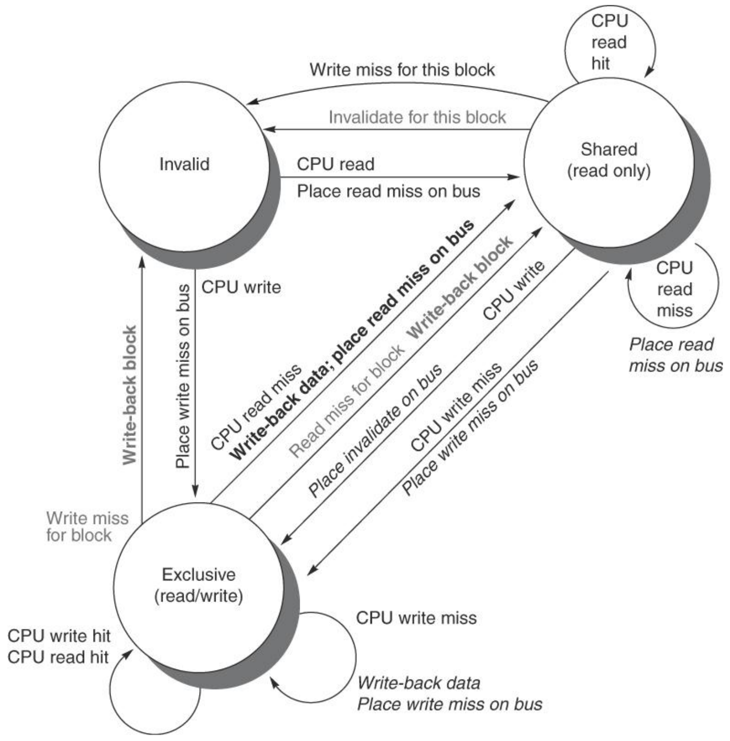
```
// Thread 0

for(i=0;i<16;i++){
    a[i]= 16;
 }
```

```
// Thread 1
tmp=0;
for(i=0;i<16;i++){
    a[i]=tmp;
    tmp+=a[i];
 }
```

```
// Thread 2
cnt=0;
for(i=0;i<4;i++){
    cnt+=b[i];
 }
```

The computer is an architecture CC-NUMA with:

- Two 32 bit processors with a single cache level that is private to each processor and has direct mapping. The size of a cache line is 16 bits.
- All caches are initially empty.
- Threads 0 and 2 are executed in processor 0 while thread 1 is executed in processor 1.
- Variables i, tmp, and cnt are stores in registers (they are not stored in memory).
- The block containing a[0] is associated to the same cache line than the block containing b[0].



Fill the following tables and justify answers for the following questions:

- Starting from the initial situation, provide in the following table the state transitions for block storing a[0] when thread 0 executes first completely and then thread 1. Provide the bus traffic associated to each thread. Note: in the case of several transitions associated to one thread, provide all of them.

Code	Transition P0	Transition P1	Bus signals
Thread 0			
Thread 1			

- Starting from the initial situation, provide in the following table the state transitions for block a[0] when thread 1 executes first completely and the thread 0. Provide the bus traffic associated to each thread. Note: in the case of several transitions associated to one thread, provide all of them.

Code	Transition P0	Transition P1	Bus signals
Thread 1			
Thread 0			

- Starting from the initial situation, provide in the following table the state transitions for block storing a[0] when thread 0 executes first completely and then thread 2 executes completely, and then thread 1. Provide the bus traffic associated to each thread. Note: in the case of several transitions associated to one thread, provide all of them.

Code	Transition P0	Transition P1	Bus signals
Thread 0			
Thread 2			
Thread 1			

- For each of the former scenarios, provide the number of cache misses for each process.

Scenario	Cache misses P0	Cache misses P1
Thread 0 -> Thread 1		
Thread 1 -> Thread 0		
Thread 0 -> Thread 2 -> Thread 1		

ANSWERS

A

Code	**Transition P0**	**Transition P1**	**Bus signals**
Thread 0	**I -> E**	**I**	**Write miss**
Thread 1	**E -> I**	**I -> E ; E -> E**	**Write miss; Write-back block**

B

Code	**Transition P0**	**Transition P1**	**Bus signals**
Thread 1	**I**	**I -> E ; E -> E**	**Write miss**
Thread 0	**I -> E**	**E -> I**	**Write miss; Write-back block**

C

Code	**Transition P0**	**Transition P1**	**Bus signals**
Thread 0	**I -> E (stores a[0])**	**I**	**Write miss**
Thread 2	**E -> S (stores b[0])**	**I**	**Read miss; write back block**
Thread 1	**S -> S (b[0])***	**I -> E ; E -> E**	**Write miss**

Each block has 4 words. Threads 0 and 1 access 4 blocks while thread 2 accesses 1 block. In thread 1, the first line causes a miss while the second is a hit. Thus:

Scenario	Misses in cache P0	Misses in cache P1
Thread 0 -> Thread 1	4	4
Thread 1 -> Thread 0	4	4
Thread 0 -> Thread 2 -> Thread 1	4 + 1	4

Exercise 3 [1 point]

Given the following code parallelized with OpenMP, and assuming that there are 4 threads available (`export OMP_NUM_THREADS=4`) and that iter = 16:

```
#pragma omp parallel for private(j)
for (i = 0; i < iter; ++i) {
   for (j = iter - (i+1); j < iter; ++j) {
      //Function with computational load of 2s
      compute_iteration(i, j, ...);
   }
}
```

Answer the following:

a. Fill in the folllowing table a possible iterations mapping for the loop execution with index 'i' with static scheduling, `schedule(static)`. Provide in the table which thread performs each loop iteration (each distinct value from 'i') and how much time takes each iteration. Besides, compute the approximate execution time per thread and the total execution time.

#Iter	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Thread (ID)																
Time (s)																

b. Fill in the following table a possible iterations mapping for the loop execution with index 'i' with dynamic scheduling and *chunk* 2, `schedule(dynamic, 2)`. Provide in the table which thread performs each loop iteration (each distinct value from 'i') and how much time takes each iteration. Besides, compute the approximate execution time per thread and the total execution time.

#Iter	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Thread (ID)																
Time (s)																

c. Justify which of the former schedulings would be better for a generic case (variable number of iterations and threads).

ANSWER:

a. Static

	It 0	It 1	It 2	It 3	It 4	It 5	It 6	It 7	It 8	It 9	It 10	It 11	It 12	It 13	It 14	It 15
Thread	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
Tiempo	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

Time per thread:

Thread 0: 20s
Thread 1: 52s
Thread 2: 84s
Thread 3: 116s

Total time: 116s

a. Dynamic

	It 0	It 1	It 2	It 3	It 4	It 5	It 6	It 7	It 8	It 9	It 10	It 11	It 12	It 13	It 14	It 15
Thread	0	0	1	1	2	2	3	3	0	0	1	1	2	2	3	3
Time	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

Time per thread:

Thread 0: 44s
Thread 1: 60s
Thread 2: 76s
Thread 3: 92s

Total time: 92s

b. Dynamic

Regardless the number of iterations and thread, as the workload per iteration is not uniform, it will adapt better to dynamic scheduling. It could also be considered as a valid answer *guided*.

Universidad Carlos III de Madrid

Computer Science and Engineering Department
Computer Architecture

Ordinary Exam
January 23th, 2015

ARCOS

Exercise 4 [1 point]

The following code is programmed using atomics. In point A, head contains a value of 8 and there is a try to insert a 9, and consequently the output will be **“8 8 9”**. If another thread tries to insert a 10 concurrently, provide which data will be printed if part B (starting from line 16) is executed, and which data will be printed if part C (starting from line 25) is executed.

```
1.  struct node
2.  {
3.          std::shared_ptr<T> data;
4.          node* next;
5.          node(T const& data_):data(new T(data_)), next(nullptr) {}
6.  };
7.  std::atomic<node*> head;
8.  void push(T const& data)
9.  {
10.         node* const new_node=new node(data);
11.         new_node->next=head.load();
12.
13.         //A
14.         std::cout << *(head.load()->data) << " "; // 8
15.         std::cout << *(new_node->next->data) <<  " "; // 8
16.         std::cout << *(new_node->data) << std::endl; // 9
17.
18.         if(head.compare_exchange_strong(new_node->next,new_node)) {
19.            //B
20.            std::cout << *(head.load()->data) << " ";
21.            std::cout << *(new_node->next->data) <<  " ";
22.            std::cout << *(new_node->data) << std::endl;
23.         } else {
24.            //C
25.            std::cout << *(head.load()->data) << " ";
26.            std::cout << *(new_node->next->data) <<  " ";
27.            std::cout << *(new_node->data) << std::endl;
28.         }
29. }
```

SOLUCIÓN:

Case B: Message “9 8 9”

Case C: Message “10 10 9”

It is not possible any other solution as compare_exchange_strong does not allow spurious failures.