Universidad Carlos III de Madrid

Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

Rules:

- Read the exam carefully before starting
- Mobile phones, electronic devices, books and notes are not allowed.
- Mobile phones have to be completely disconnected
- Write the exam using a pen (not a pencil)
- **Exam duration is 180 minutes**
- **Use a different page for each exercise. If you don't answer one question, use a white page with the exercise number and your name.**

NAME:
SURNAME:
NIA / ID NUMBER:

Exercise 1: (15 POINTS)

Given a single core computer used for running a financial application. This application uses 90% of the execution time for computational intensive operations. The remaining 10% is spent waiting for Input/Output hard disk accesses. The computational intensive part is divided in 75% for floating point operations and 25% for other instructions. The average CPI of a floating point operation is 12. The rest of the instructions have an average CPI of 4.

We have two different alternative platforms for running this application:

- **Alternative A**: A single core processor with a clock frequency 50% higher than the original computer. For this new system the floating point instructions take 10% more clock cycles per instruction and the rest of the instructions take 25% more cycles than the original ones. The Input/Output time is the same as the original platform.
- **Alternative B**: A four core processor with a frequency 50% smaller than the original computer. The floating point instructions require 20% less clock cycles than the original ones. The rest of the instructions take the same amount of clock cycles as the original ones. The Input/Output time is the same as the original platform.

Answer the following questions:

a) What would be the overall speedup of Alternative A?
b) What would be the overall speedup of Alternative B? Assume that the computational intensive part is completely parallelizable and that the Input/Output part is sequential.

Solution:

The execution time for instruction for the original architecture will be:

T(inst,orig) = 0.75 * 12 * IC * P + 0.25 * 4 * IC * P = (9+1) * IC * P

Architecture A:

The execution time for instruction for architecture A will be:

T(inst,A) =(0.75 * (1,1 * 12) + 0.25 * (1.25 * 4)) * IC * (P/1.5) = (9.9+1.25) * IC * P / 1.5 = 11,15 / 1.5* IC * P = 7,433 * IC *P

The Speedup for instructions will be:

S(inst,A) = T(inst,orig) / T(inst,A) = 10 / 7,433 = 1,345

Using Amdahl law the overall speedup will be:

S(A) = 1 / (0.1 + 0.9 / 1.345) = 1,3

Architecture B:

The execution time for instruction for architecture A will be:

T(inst,B) = (0.75 * 1.2 * 12 + 0.25 *4) * (IC / 4) * (P /0.5) = (10.8+1) * 2 / 4 * IC * P = 5,9 * IC * P

The instruction speedup will be:

S(inst,B) = T(inst,orig) / T(inst,B) = 10/5.9 = 1.695

Using Amdahl law the overall speedup will be:

S(B) = 1 / (0.1 + 0.9 / 1,695) = 1.585

Exercise 2: (15 POINTS)

Answer the following questions:

a) Define when a shared memory system is coherent (maximum 3-4 text lines).
b) Write the conditions that are necessary to have a memory-coherent architecture.
c) Write the restrictions that guarantee sequential consistency.
d) In the acquire/release consistency model, describe the *acquire* operation.
e) In the acquire/release consistency model, describe the *release* operation.

Exercise 3: (10 POINTS)

Explain briefly the differences between OpenMP static, dynamic, and guided schedulers in OpenMP.

Exercise 4: (20 POINTS)

A given processor is intended to run the following code fragment:

```
i0: lw $r4, 0($r1)
i1: lw $r5, 0($r2)
i2: add $r4, $r4, $r5
i3: sw $r4, 0($r3)
i4: addi $r1, $r1, 4
i5: addi $r2, $r2, 4
i6: addi $r3, $r3, 4
i7: bne $r3, $r0, i0
```

Assume that the processor has a segmented architecture of 5 steps (fetch, decode, execute, memory and post-writing) without forwarding. All operations are executed in one cycle per stage, except:

- Load and store instructions, that require two cycles for the memory stage (an additional cycle).
- Branch instructions require an additional cycle in the execution stage. Assume that these instructions do not have any branch prediction.

Answer the following questions:

a) Obtain the RAW risks.
b) Show the timing diagram with the phases of execution of each instruction for one iteration.
c) Determine how many cycles it takes to execute a single iteration if there is no branch prediction.
d) Propose a loop unrolling solution, assuming that the loop runs 1000 iterations. Unroll with a factor of four iterations and write the resulting code.
e) Determine the speedup obtained with loop unrolling.

Solution:

A:

$r4: i0 -> i2
$r5: i1 -> i2
$r4: i2 -> i3
$r3: i6 -> i7

Universidad Carlos III de Madrid

Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

B:

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
I0	F	D	X	M1	M2	W																
I1		F	D	X	-	M1	M2	W														
I2			F	-	-	-	-	D	X	M	W											
I3								F	-	-	D	X	M1	M2	W							
I4											F	D	X	-	M	W						
I5												F	D	-	X	M	W					
I6													F	-	D	X	M	W				
I7															F	-	-	D	X	X	M	W
I0																			F	D	X	M

- I0: requires of two memory cycles.
- I1: requires of two memory cycles and needs to wait one cycle for accessing the memory
- I2: needs WB of I1 for deconding
- I3: Needs to wait until I2 frees the fetch stage.
- I4: Needs to wait until I3 frees the memory stage.
- I5: Needs to wait until I4 frees the execution stage.
- I6: Needs to wait until I5 frees the decoding stage.
- I7: needs WB of I6 for deconding.
- I0: Next iteration: needs to complete the deconding stage of I7

C:

It requires of 17 cycles because the next instruction needs to complete the deconding stage of the branch instruction.

D:

In the following there is one possible solution (there are other valid alternatives).

```
i0: lw $r4, 0($r1)
i1: lw $r5, 0($r2)
i2: lw $r6, 4($r1)
i3: lw $r7, 4($r2)
i4: lw $r8, 8($r1)
i5: lw $r9, 8($r2)
i6: lw $r10, 12($r1)
i7: lw $r11, 12($r2)
i8: add $r4, $r4, $r5
i9: add $r6, $r6, $r7
i10: add $r8, $r8, $r9
i11: add $r10, $r10, $r11
i12: sw $r4, 0($r3)
i13: sw $r6, 4($r3)
```

```
i14: sw $r8, 8($r3)
i15: sw $r10, 12($r3)
i16: addi $r3, $r3, 16
i17: addi $r2, $r2, 16
i18: addi $r1, $r1, 16
i19: bne $r3, $r0, i0
```

E:

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I0	F	D	X	M1	M2	W														
I1		F	D	X	-	M1	M2	W												
I2			F	D	-	X	-	M1	M2	W										
I3				F	-	D	-	X	-	M1	M2	W								
I4						F	-	D	-	X	-	M1	M2	W						
I5								F	-	D	-	X	-	M1	M2	W				
I6										F	-	D	-	X	-	M1	M2	W		
I7												F	-	D	-	X	-	M1	M2	W
I8														F	-	D	-	X	-	M
I9																F	-	D	-	X
I10																		F	-	D
I11																				F

Instr.	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
I8	W																			
I9	M	W																		
I10	X	M	W																	
I11	D	X	M	W																
I12	F	D	X	M1	M2	W														
I13		F	D	X	-	M1	M2	W												
I14			F	D	-	X	-	M1	M2	W										
I15				F	-	D	-	X	-	M1	M2	W								
I16						F	-	D	-	X	-	M	W							
I17								F	-	D	-	X	M	W						
I18										F	-	D	X	M	W					
I19												F	D	X	X	M	W			
I0														F	D	X	M1	M2		

One unrolled iteration requires of 33 cycles, thus the number of cycles per iteration will be 33/4=8.25.

An the overall speedup will be

S = 20/ 8.25 = 2.42

Universidad
Carlos III de Madrid

Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

Universidad
Carlos III de Madrid

Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

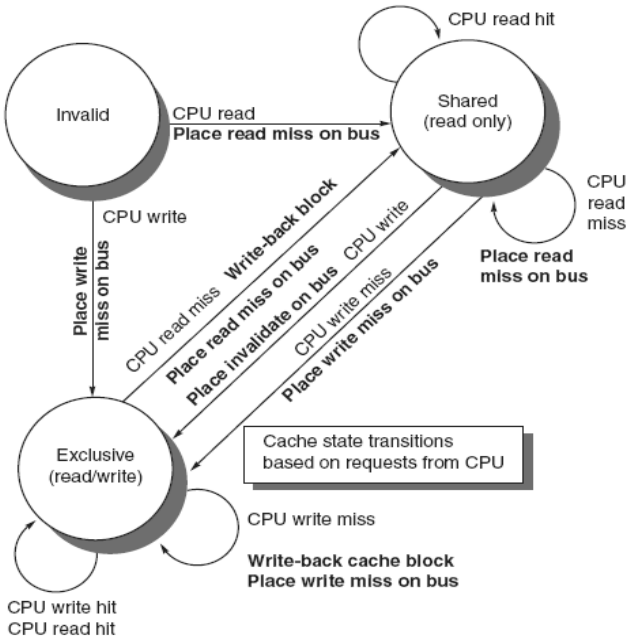
Universidad Carlos III de Madrid

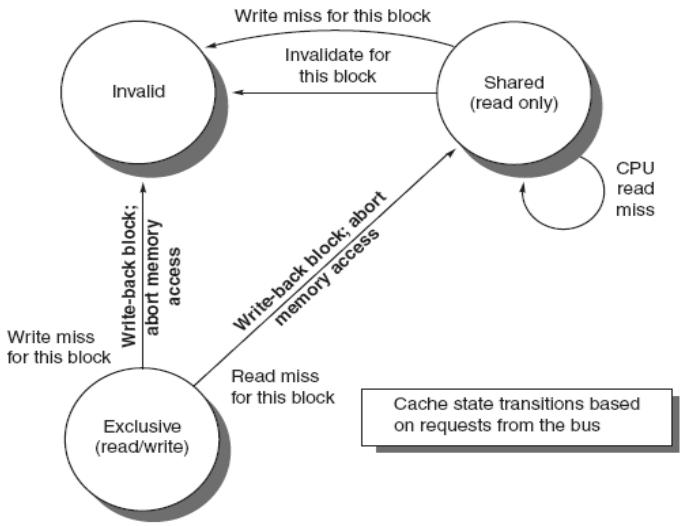
Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

Exercise 5: (20 POINTS)

Given a symmetric shared memory system based on the snooping bus protocol and consisting of three processors. Each processor has a private cache based on the MSI protocol. The cache memories are direct-mapped and have only four cache lines with blocks of two words. Each cache uses the complete memory address of the block for the **tag field**.





The following tables show the initial state of each cache memory, with the least-significant word at the left.

Processor P0

Block	State	Tag	Data	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	S	0x00100708	0x00000000	0x00001234
B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11

Processor P1

Block	State	Tag	Data	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11

Universidad Carlos III de Madrid

Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

ARCOS

Processor P2

Block	State	Tag	Data	
B0	S	0x00100720	0x00000000	0x1111AAAA
B1	S	0x00100708	0x00000000	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	I	0x00100718	0x00001234	0x1111AABB

The following paragraphs are 3 different actions that have to be analyzed independently, all of them starting from the initial program state shown above.

a) P2: write 0x00100708, 0xFFFFFFFF
b) P2: read 0x00100708
c) P2: read 0x00100718

For each one the actions use a table like the one shown below. In each table show the changes that are produced in the cache memories. For the reads, write which will be the value read by the operation.

Processor	Block	Previous state	New State	Tag	Data	

Solution:

A:

P2 writes in B1 block. The new state of this block is M and an invalidation is placed in the bus. This invalidation is ignored by P1 but not by P0. In P0 block B1 takes the Invalid state.

Main memory doesn't change any memory location

Procesador P0

Block	State	Label	Data	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	I	0x00100708	0x00000000	0x00001234
B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11

Universidad Carlos III de Madrid

Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

ARCOS

P1

Block	State	Label	Data	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11

P2

Block	State	Label	Data	
B0	S	0x00100720	0x00000000	0x1111AAAA
B1	M	0x00100708	0xFFFFFFFF	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	I	0x00100718	0x00001234	0x1111AABB

New states

Processor	Block	Previous state	New state	Label	Data	
P2	B1	I	M	0X00100708	0XFFFFFFFF	0X00001234
P0	B1	S	I	0X00100708	0X00000000	0X00001234

B:

P2 reads B1 in its local cache. B2 state is S. There is a cache hit and there is no any change.

The value read is: 0x00000000

C:

There is a cache miss when P2 reads B3. A miss signal is placed in the bus and a valid block with a S state is fetched from it.

P0 ignores these actions and keeps an I state for this block

P1 keeps B3 block in S state.

The value read is 0x00000000

P0

Block	State	Label	Data	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	S	0x00100708	0x00000000	0x00001234

B2	M	0x00100710	0x00000000	0x0077AABB
B3	I	0x00100718	0x00000000	0x7FAABB11

P1

Block	State	Label	Data	
B0	I	0x00100700	0x00000000	0x7FAABB11
B1	M	0x00100728	0x00000000	0xFF000000
B2	I	0x00100710	0x00000000	0xEEEE7777
B3	S	0x00100718	0x00000000	0x7FAABB11

P2

Block	State	Label	Data	
B0	S	0x00100720	0x00000000	0x1111AAAA
B1	S	0x00100708	0x00000000	0X00001234
B2	I	0x00100710	0x00000000	0x7FAABB11
B3	**S**	0x00100718	**0x00000000**	**0x7FAABB11**

Processor	Block	Previous state	New state	Label	Data	
P2	B3	I	S	0X00100718	0X00000000	0X7FAABB11

Universidad
Carlos III de Madrid

Department of informatics
Computer Science degree
Computer Architecture

Final exam
January 24th 2014

Exercise 6: (20 POINTS)

Given the following fragment of code:

```
std::mutex m; // global mutex
int counter; // global counter

void f() {
  m.lock();
  ++counter;
  m.unlock();
}
```

We want to replace the global variable `m` and we want to avoid calls to the operating system, while ensuring mutual exclusion on the counter variable.

Answer the following questions:

a) Suggest and code a solution that provides sequential consistency and does not involve system calls.
b) Suggest and code a solution that provides acquire-release consistency.
c) Suggest and code a solution that provides acquire-release consistency and that is valid when the counter is a **double** variable.

SOLUTION:

A:

```
std::atomic<int> counter; // global counter
void f() {
        ++counter;
}
```

B:

```
std::atomic<int> counter; // global counter l

void f() {
        counter.fetch_add(1, std::memory_order_acq_rel);
}
```

C:

```
std::atomic_flag  spin = ATOMIC_FLAG_INIT;
double counter ; // global counter

void f() {
        while (spin.test_and_set(std::memory_order_acquire)) {}
        counter += 1.0;
        spin.clear(std::memory_order_release);
}
```