Universidad Carlos III de Madrid

Computer Science ane Engineering
Departamento of Computer Science ane Engineering

Computer Architecture
January 20th, 2021

ARCOS

NOTES:

- Please, read carefully all the exam before starting to answer.
- You are not allowed to use books or notes.
- Mobile phones must be switched off during the whole exam (switched off, not just muted).
- Please, do not use pencil. Only answers using ink pens will be graded.
- **You have a maximum time of 120 minutes for this part.**
- **If you consider that a question is not clear, make assumptions as you consider. No clarifications will be performed during the exam.**

Exercise 1 [0.5 points]:
During a normal execution, a program exhibits the following distribution of instructions:

- Arithmetic instructions: 60% of the total.
- Branch instructions: 15% of the total.
- Load/store instructions: 20% of the total.
- Other instructions: 5% of the total.

The number of cycles per instruction is (global CPI) 2.25. We are evaluating an improvement in the arithmetic unit that will make arithmetic instructions to require 80% of the original cycles.

Determine the new global CPI.

SOLUTION:

By applying Amdalh's Law to get the global speedup, we should consider the following:

- The improvable fraction is 0.6 (arithmetic instructions).
- Speedup of improvement is 100/80 = 1.25.

S = 1/(0.4 + 0.6/1.25) =1/(0.4+0.48) = 1/0.88 = 1.136

The new CPI may be obtained by dividing the original CPI by the global speedup:

CPI = 2.25 /1,136 = 1.981

Exercise 2 [3 points]:
Given the following code fragment

```
loop:   lw $t1, 0($t0)          #I1
        lw $t2, 1000($t0)       #I2
        mul $t2, $t1, $t2       #I3
        lw $t1, 2000($t0)       #I4
        add $t2, $t1, $t2       #I5
        sw $t2, 3000($t0)       #I6
        addi $t0, $t0, 4        #I7
        addi $t3, $t3, -4       #I8
        bnz $t3, loop           #I9
```

You are asked:

Universidad Carlos III de Madrid

Computer Science ane Engineering
Departamento of Computer Science ane Engineering

ARCOS

Computer Architecture
January 20th, 2021

1. [0.5 points] Identify all the RAW data dependencies in the code within an iteration (do not identify dependencies among different iterations). Express all dependencies in the format $t12: I11 -> I12 (dependency of instruction I11 to instruction I12 due to register $t12).

2. [1 point] Provide a time diagram for the first full iteration (including fetch for the first instruction fo the next iteration) for a RISC pipeline with 5 stages where there is no forwarding hardware. Reads from instruction cache require two memory cycles. Reads from data cache require two memory cycles and writes require three memory cycles. For branch instructions both the branch address and branch outcome are known at the end of the execution stage and are always predicted to not-taken.

3 [1 point] Provide a time diagram for the first iteration, but assuming that there is forwarding hardware.

4 [0.5 points] Determine the speedup thanks to forwarding for the execution of the first iteration.

SOLUTION:

1.- The following dependencies are identified:

- $t1: I1 -> I3
- $t2: I2 -> I3
- $t2: I3 -> I5
- $t1: I4 -> I5
- $t2: I5 -> I6
- $t3: I8 -> I9

2.-

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I1	IF	IF	ID	EX	M	M	WB										
I2			IF	IF	ID	EX	M	M	WB								
I3					IF	IF	ID	-	-	**EX**	M	WB					
I4							**IF**	IF	-	ID	EX	M	M	WB			
I5										IF	IF	ID	-	-	**EX**	M	WB
I6												IF	IF	-	ID	-	-
I7															IF	IF	-
I8																	

Instruction	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
I6	**EX**	M	M	M	WB												
I7	ID	EX	-	-	M	WB											
I8	IF	IF	ID	-	EX	M	WB										
I9			IF	IF	ID	-	-	EX	M	WB							
I10					IF	IF	-	ID	--								
I1									**IF**	IF	ID	EX	M	M	WB		

3.-

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I1	IF	IF	ID	EX	M	M	WB										
I2			IF	IF	ID	EX	M	M	WB								
I3					IF	IF	ID	-	**EX**	M	WB						
I4							IF	IF	ID	EX	M	M	WB				
I5									IF	IF	**ID**	-	EX	M	WB		
I6											IF	IF	ID	EX	M	M	M
I7													IF	IF	ID	EX	-
I8															IF	IF	ID
I9																	IF

Instruction	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
I5																	
I6	WB																
I7	M	WB															
I8	EX	M	WB														
I9	IF	ID	EX	M	WB												
I10		IF	-														
I1				IF	IF	ID	EX	M	M	WB							

4. In the first case 25 cycles are required, while 20 are required in the second case:

S = 25/20 = 1.25

Exercise 3 [0.5 points]: An application represents image in the following way:

```
struct pixel {
  uint8_t r, g, b;
};

struct image_aos {
  int32_t width, height;
  std::vector<pixel> buf;
};
```

The program uses a function *to_gray()* for converting the image to gray scale.

```
void to_gray(image_aos & img) {
  for (int i=0;i<img.width;++i) {
    for (int j=0; j<img.height; ++j) {
      const int k = i*img.height+j;
      uint8_t val = 0.2989 * img.buf[k].r + 0.587 * img.buf[k].g + 0.114 img.buf[k].b;
      img.buf[k].r = val;
      img.buf[k].g = val;
      img.buf[k].b = val;
    }
  }
}
```

Type uint8_t is used to represent 1 byte unsigned integers and type int32_t is used to represent 4 bytes signed integers.

Propose an alternative implementation using an structure of arrays.

```
struct image_soa {
  // Define structure fields
};

void to_gray(image_soa & img) {
  // Implement the function
}
```

SOLUTION:

```
struct image_soa {
  int32_t width, height;
  std::vector<uint8_t> r, g, b;
};

void to_gray(image_soa & img) {
  for (int i=0; i<img.width; ++i) {
    for (int j=0; j<img.height; ++j) {
      const int k = i*img.width +j;
      imgg.r[k] = 0.2989 * img.r[k] + 0.587 * img.g[k] + 0.114 img.b[k];
      img.g[k] = img.r[k];
      img.b[k] = img.r[k];
    }
  }
}
```

Exercise 4 [2 points]:

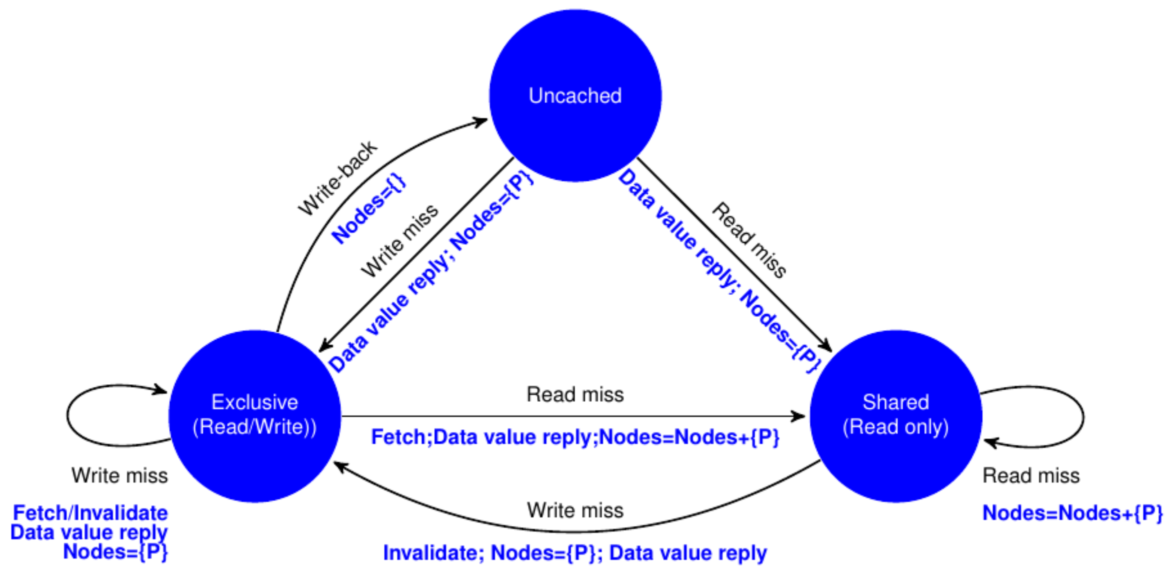
A shared memory machine has four NUMA nodes with the following characteristics:

- Each node has a single processor.
- The processor has a single cache level with 64 KB 8 way set-associative cache and line size of 64 bytes.
- RAM memory in each node is 64 GB.
- Directory is distributed among nodes.

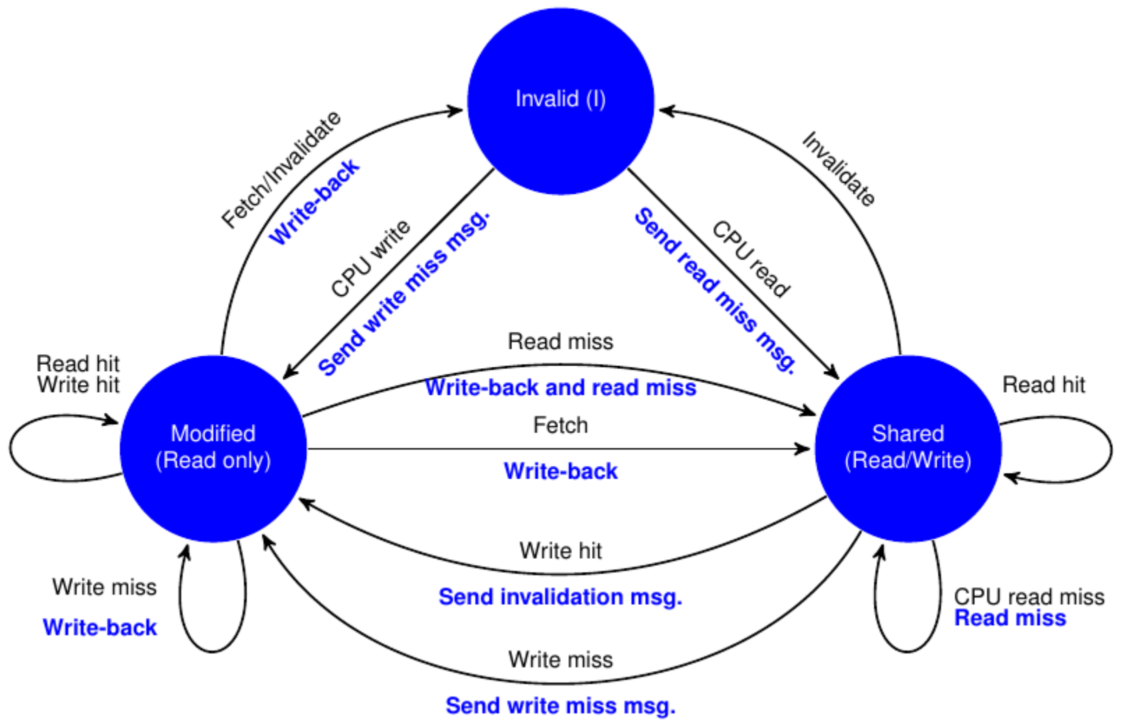
Computer Science ane Engineering
Departamento of Computer Science ane Engineering

Computer Architecture
January 20th, 2021

The directory state transition diagram is the following:



State transitions for individual caches are as follows:



4.1 [0.25 points] ¿How many different MSI state machines are there in this machine?

4.2 [0.25 points] ¿What is the memory space size that can be addressed from node number 2?

4.3 [1.5 points] Complete the following table, for the cache entry where variable x is stored, where you should specify the states in computing node caches (M → Modified, S → Shared, I → Invalid), as well as network traffic for each node. Column instruction states the node in which instruction is executed (p1: lw $t1, x means execut lw $t1, x in node P1).

Instruction	State P1	Traffic P1	State P2	Traffic P2	State P3	Traffic P3	State P4	Traffic P4
Initial	I	--	I	--	I	--	I	--
P1: lw $t1, x								
P1: lw $t2, x								
P2: lw $t2, x								
P3: lw $t2, x								
P3: sw $t2,x								
P1: lw $t1, x								
P4: sw $t1, x								
P4: lw $t1, x								

Additionally, complete the following table where you should specify the directory state (U → Uncached, S → Shared, E → Exclusive), as well as the nodes list in the directory and actions from directory.

Instruction	Directory State	Directory List	Directory Action
Initial	U	{ }	--
P1: lw $t1, x			
P1: lw $t2, x			
P2: lw $t2, x			
P3: lw $t2, x			
P3: sw $t2,x			
P1: lw $t1, x			
P4: sw $t1, x			
P4: lw $t1, x			

SOLUTION:

4.1: Each cache line contains an independent MSI state machine with 3 possible states. The local cache for each processor is 64 KB size and the block size is 64 B giving 1024 lines per cache. There are four processors and consequently the total number of state machines is 4 * 1024 = 4096.

4.2: Each node has 64 GB RAM, but can access to the memory of other nodes. Thus, processor number 2 can access to 4 * 64 GB = 256 GB.

4.3:

Instruction	State P1	Traffic P1	State P2	Traffic P2	State P3	Traffic P3	State P4	Traffic P4
Initial	I	--	I	--	I	--	I	--
P1: lw $t1, x	S	Read miss	I	--	I	--	I	--
P1: lw $t1, x	S	--	I	--	I	--	I	--
P2: lw $t2, x	S	--	S	Read Miss	I	--	I	--
P3: lw $t2, x	S	--	S	--	S	Read Miss	I	--
P3: sw $t2,x	I	--	I	--	M	Invalidation	I	--
P1: lw $t1, x	S	Read Miss	I	--	S	Write-Back	I	--
P4: sw $t1, x	I	--	I	--	I	--	M	Write miss
P4: lw $t1, x	I	--	I	--	I	--	M	--

Instruction	Directory State	Directory List	Directory Action
Initial	Uncached	{ }	--
P1: lw $t1, x	S	{ P1 }	Data Reply
P1: lw $t1, x	S	{ P1 }	--
P2: lw $t2, x	S	{ P1, P2 }	Data Reply
P3: lw $t2, x	S	{ P1, P2, P3 }	Data Reply
P3: sw $t2,x	E	{ P3 }	Invalidate Data Reply
P1: lw $t1, x	S	{ P1, P3 }	Fetch Data Reply
P4: sw $t1, x	E	{ P4 }	Invalidate Data Reply
P4: lw $t1, x	E	{ P4 }	--

Exercise 5 [2 points]: Consider the following code for a mutex and condition variable based buffer.

```
template <typename T>
class locked_buffer {
public:
  explicit locked_buffer(int n) :
    size_{n},
    buf_{new std::optional<T>[size_]}
  {
  }

  locked_buffer(const locked_buffer &) = delete;
  ~locked_buffer() = default;

  int size() const noexcept { return size_; }

  bool empty() const noexcept {
    std::lock_guard<std::mutex> l{mut_};
    return is_empty();
  }

  bool full() const noexcept {
    std::lock_guard<std::mutex> l{mut_};
    return is_full();
  }


  void put(const std::optional<T> & x);
  std::optional<T> get();

private:

  int next_position(int p) const noexcept { return p + ((p+1>=size_)?(1-size_):1); }
  bool is_empty() const noexcept { return (next_read_ == next_write_); }
  bool is_full() const noexcept {
    const int next = next_position(next_write_);
    return next == next_read_;
  }

private:
  const int size_;
  const std::unique_ptr<std::optional<T>[]> buf_;
  int next_read_ = 0;
  int next_write_ = 0;
  mutable std::mutex mut_;
  std::condition_variable not_full_;
  std::condition_variable not_empty_;
};
```

Consider now the following code for buffer based on atomic types:

```
template <typename T>
class atomic_buffer {
public:
  explicit atomic_buffer(int n) :
    size_{n},
    buf_{new std::optional<T>[size_]}
  {
  }

  ~atomic_buffer() = default;

  int size() const noexcept { return size_; }
  bool empty() const noexcept { return next_read_ == next_write_; }
  bool full() const noexcept {
    const int next = next_position(next_write_.load());
    return next == next_read_.load();
  }

  void put(const std::optional<T> & x);
  std::optional<T> get();

private:
  int next_position(int p) const noexcept { return p + ((p+1>=size_)?(1-size_):1); }

private:
  const int size_;
  const std::unique_ptr<std::optional<T>[]> buf_;
  alignas(64) std::atomic<int> next_read_ {0};
  alignas(64) std::atomic<int> next_write_ {0};
};
```

```
template <typename T>
void atomic_buffer<T>::put(const std::optional<T> & x)
{
  const int next = next_position(next_write_.load());
  while (next == next_read_.load()) {}
  buf_[next_write_.load()] = x;
  next_write_.store(next);
}
```

Answer the following questions about *locked_buffer*:

5.1 [0.25 points]: What is the difference between *empty()* and *is_empty()* from the concurrency viewpoint?
5.2 [0.25 points]: Why is it necessary to mark variable mut with the reserved keyword mutable?
5.3 [0.5 points]: Write an implementation for *put()* member function in *locked_buffer*.

Answer the following questions about *atomic_buffer*:

5.4 [0.25 points]: What would be the effect of removing the *alignas(64)* specification for fields *next_read_* and *next_write_*?
5.5 [0.25 points]: Why is the value 64 used for *alignas* specifier?
5.6 [0.5 points]: Implement *get()* member function in *atomic_buffer*.

SOLUTION

5.1: Public function empty() is protected by a mutex while private function is_empty() is not. Consequently, only the first one can be safely used in concurrent codes (thread safety).

5.2: Member variable mut_ is qualified as mutable sot that it can be modified from const member functions.

5.3: A possible implementation for put() could be:
5.1: La función pública empty() está protegida por un mutex mientras que la función privada is_empty() no lo está.

```
template <typename T>
void locked_buffer<T>::put(const std::optional<T> & x)
{
  using namespace std;
  unique_lock<mutex> l{mut_};
  not_full_.wait(l, [this] { return !is_full(); });
  buf_[next_write_] = x;
  next_write_ = next_position(next_write_);
  not_empty_.notify_one();
}
```

5.4: If alignas specifiers are removed both member variables could be allocated to the same cache line and lead to a problem of flase sharing.

5.5: 64 is the cache line size in most current processors. This value guarantees that both data members will be allocated in different cache lines.

5.6: A possible implementation for *get()* could be:

```
template <typename T>
std::optional<T> atomic_buffer<T>::get()
{
  while (empty()) {}
  auto res = buf_[next_read_.load()];
  next_read_.store(next_position(next_read_.load()));
  return res;
}
```