Prelim 2

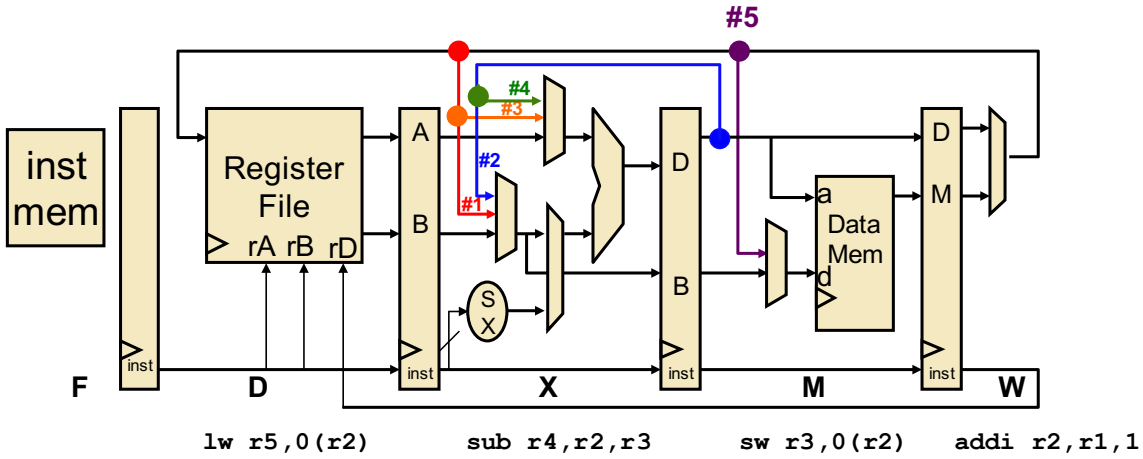
Computer Science 3410, Cornell University

Spring 2016, Prof. Bracy 5 May 2016

Solutions

1. Data and Control Hazards [16 pts]

   (a) [8 pts] **Forwarding.** In the pipeline below, the register file is read in the 1st half of the cycle and written in the 2nd half of the cycle. Bypasses #1-#5 are shaded. In the D/X latch, the 1st operand to an ALU instruction is stored in the A slot of the latch. The 2nd operand is stored in the B slot of the latch.



   For each input specified below, state where the instruction gets its data. 1-5 refer to the bypasses shown in the above figure. RF stands for register file. ERROR means that the instruction cannot possibly get its data in order to execute correctly given the processor above and the timing specified in the diagram.

	Circle One						
r2 of the sw	#1	#2	#3	#4	#5	RF	ERROR
r2 of the sub	#1	#2	#3	#4	#5	RF	ERROR
r3 of the sub	#1	#2	#3	#4	#5	RF	ERROR
r2 of the lw	#1	#2	#3	#4	#5	RF	ERROR

Answer: ***#4, #3, RF, ERROR***

   (b) [8 pts] **Control Dependences.** Suppose we had asked you to implement a 2-instruction control delay slot for Project 2. In other words, the next *two* instructions immediately following a control instruction will always be executed. Which of the following statements about this concept are true? Check all that apply.

_____ 2 delay slots would be harder to implement in Logisim than 1.
_____ It will be harder for a compiler to fill the 2nd delay slot than to fill the 1st.
_____ Processors with 1 delay slot benefit *more* from branch prediction than one with 2 delay slots.
_____ A processor with 2 delay slots might support a faster clock than one with only 1.

Answer: Items 2-4 should be checked. (Item 1 is not not true.)

2. ISAs [8 pts]

For each multiple choice question there is only *one* correct answer.

(a) [2 pts] What is one advantage of a CISC ISA?

A It naturally supports a faster clock.
B Instructions are easier to decode.
C The static footprint of the code will be smaller.
D The code is easier for a compiler to optimize.
E You have a lot of registers to use.

Correct Answer: **C**

(b) [2 pts] What processor feature does a compiler **not** need to know about?

A There is a control delay slot.
B The number of inputs each instruction can have.
C The processor performs statically-scheduled multiple issue execution.
D The processor performs dynamically-scheduled multiple issue execution.
E Whether the processor supports predication.

Correct Answer: **D**

(c) [2 pts] What is **not** a good reason to limit the number of instructions offered by a particular ISA?

A Offering more instructions leads to executables with more static instructions.
B More complicated instructions are difficult to pipeline.
C More instructions may lead to needing more bits in the opcode field which could make static instructions larger.
D It is very difficult to ever stop supporting a particular instruction once it has been added to the ISA.
E More instructions will complicate the decode logic of the processor.

Correct Answer: **A**

(d) [2 pts] Which of the following statements about ISAs is **true**?

A Because clock frequencies are no longer increasing, a return to CISC ISAs makes sense.

B Which cache coherence protocol is implemented needs to needs to be specified in the ISA.

C Atomic operations need to be specified in the ISA.

D ISA design has stagnated; no new ISAs of any commercial significance have appeared in the past 10 years.

E A system call is OS-specific and not an ISA-level feature.

Correct Answer: **C**

3. Calling Conventions [16 pts]

The following assembly is the body of a function compiled for a MIPS processor with a control delay slot:

```
Body:   addiu $s3, $a0, 0
        addiu $s5, $a1, 0
        addiu $s6, $a2, 0
Loop:   sll $t1,$s3,2
        add $t1,$t1,$s6
        lw  $t0,0($t1)
        bne $t0,$s5, Exit
        nop
        addi $s3,$s3,1
        j Loop
        nop
Exit:   addiu $a0, $s3, 0
        jal record
        nop
        addiu $v0, $s3, 0
```

Write the prologue and epilogue for this body. Use the class calling conventions.

```
PROLOGUE:
ADDIU $sp, $sp, -36
SW $ra, 32($sp)
SW $fp, 28($sp)
SW $s3, 24($sp)
SW $s5, 20($sp)
SW $s6, 16($sp)
ADDIU $fp, $sp, 32
```

```
EPILOGUE:
LW $s6, 16($sp)
LW $s5, 20($sp)
LW $s3, 24($sp)
LW $fp, 28($sp)
LW $ra, 32($sp)
ADDIU $sp, $sp, 36
JR $ra
NOP
```

4. Linkers and Loaders. [10 pts]

Below is a modified version of `heaptest.c` from Project 4:

```
-----------------------------------------------------------------
#include <stdio.h>
#include heaplib.h

#define HEAP_SIZE  16
static int ARR_SIZE = 4;

int main() {

    char heap[HEAP_SIZE];
    hl_init(heap, HEAP_SIZE * sizeof(char));
    char* ptr = (char *) hl_alloc(heap, ARR_SIZE * sizeof(char));

    ptr[0] = 'h';
    ptr[1] = 'i';
    ptr[2] = '\0';
    printf(%s\n, ptr);
    return 0;
}
-----------------------------------------------------------------
```

Where does the assembler place each of the following symbols from the above program in the object file that it creates?

Choose one:
(a) Text Segment
(b) Data Segment
(c) Exported reference in the symbol table
(d) Imported reference in the symbol table
(e) None of the above

Circle One:

(1) `HEAP_SIZE` E

(2) `ARR_SIZE` B

(3) `heap[]` E

(4) `hl_init` D

(5) `'h'` B

5. Caches [24 pts] (parts a–d)

(a) [9 pts] For each workload, choose the best block size for your cache among the choices given. Assume that integers and pointers are all 4 bytes each. Use the intuitions you have developed in this class to decide what `best` means for a cache.

Choose one: (a) 1 byte (b) 4 bytes (c) 8 bytes (d) 16 bytes (e) 32 bytes

BEST BLOCK SIZE (Circle One)

```
int scores[NUM_STUDENTS] = 0;
```

A B C D E

```
int sum = 0;
 for (i = 0; i < NUM_STUDENTS; i++) {
    sum += scores[i];
}
```

```
typedef struct _item_t {
   int value;
   item_t *next;
   char *name;
} item_t;

int sum = 0;
item_t *curr = list_head;
while (curr != NULL) {
    sum += curr->value;
    curr = curr->next;
}
```

A B C D E

```
// H = 16, W = 16
int A[H][W];

for(x=0; x < W; x++)
    for(y=0; y < H; y++)
        sum += A[y][x];
```

Assume a 256 byte cache.

A B C D E

Answers: ***E, C, B***

(b) [9 pts] **Memory Access Time.** Processor A is a 1 GHz processor (1 cycle = 1 ns). There is an L1 cache with a 1 cycle access time and a 50% hit rate. There is an L2 cache with a 10 cycle access time and a 90% hit rate. It takes 100 ns to access memory.

(3 points) What is the average memory access time of Processor A in ns?

1 + 50% (10 + 10% x 100) =
1 + 5 + 5 = 11 ns

Processor B attempts to improve upon the hit rate of Processor A by making the L1 cache larger. The new hit rate is 90%. In order to maintain a 1 cycle access time, Processor B can only run at 500 MHz (1 cycle = 2 ns).

(3 points) What is the average memory access time of Processor B in ns?

2 + 10% (20 + 10% x 100) =
2 + 2 + 1 = 5 ns

(3 points) Which processor would you buy, and why?

Although memory access times are over 2x faster on Processor B, the frequency of Processor B is 2x slower. Since most workloads are not more than 50% memory instructions, Processor A is still likely to be the faster processor for most workloads. (If I did have workloads that were over 50% memory instructions, Processor B might be the better choice.)

(c) [4 pts] Consider a 32-bit processor without virtual memory. It has a 24KB, 3-way set associative cache with a 32-byte line size.

How many index bits are there? Your Answer: _____

24 KB cache / 3 ways = 8KB per way = 2^{13} bytes per way
2^{13} Bytes / 2^5 bytes per line = 2^8 cache lines per way
= 8 bits for the index

How many tag bits are there? Your Answer: _____

32 address bits - 8 index bits - 5 offset bits = 19 tag bits

(d) [2 pts] **Cache Design.** Consider a 32-bit processor without virtual memory; its cache has 1 byte cache lines. What would happen if the most significant bits of the address were used as the cache index and the least significant bits of the address were used as the cache tag? What is the best assessment of the result of this change?

A The correctness of the processor will be compromised.
B The performance of the processor will be compromised.
C The correctness and the performance of the processor will be compromised.
D Neither the correctness nor the performance will be compromised.

Correct Answer: **B**

6. TLB Performance Optimizations [15 pts]

On machines with virtual memory, before an instruction can be fetched, the PC must be translated from a virtual address to a physical address. This means the TLB and

the I-Cache must be accessed in serial (TLB first, I-Cache second). One performance optimization employed by many processors is to access the TLB and the I-Cache **in parallel**. The cache index is extracted from the *virtual address* and is used to perform the cache lookup. At the same time, the Virtual Page Number (VPN) is extracted from the virtual address and is used to perform the TLB lookup. After the Physical Page Number (PPN) is read from the TLB, it is used to construct the tag which is then used to determine whether there is a tag match in the cache (*i.e.*, a cache hit).

Consider a machine with 16-bit virtual and physical addresses. For cache lookups, the bottom 3 bits of the physical address are used as the byte offset. The next 4 bits of the physical address are the index, and the remaining 9 bits of the physical address are the tag. Suppose pages are 256 bytes each.

(a) [2 pts] How many bits are needed to represent the page offset?

each page is 256 bytes = 2^8, so 8 bits

(b) [2 pts] The cache is physically addressed. Why is it okay to use the index bits from the virtual address to perform the cache lookup? Be brief.

The index lives inside the 8 bits that comprise the page offset. Since the page offset is not changed by address translation, the index will not change.

(cont'd from previous page) Consider the virtual address **1000 1000 1000 1000**.

(c) [2 pts] What is the value of the VPN in binary?

`1000 1000`

(d) [3 pts] Suppose that this VPN's PPN (which is stored in the TLB) is all 0s. (For example, if the PPN were 4 bits long, it would be 0000.) What tag value (*in binary*) should be used for the tag match in the cache?

`0000 0000 1`

(e) [6 pts] You want to make the cache 4x larger. What strategy would prevent the parallel access of the TLB and I-Cache? *Answer all that apply.*

A Increasing the associativity of the cache by 4x.

B Increasing the line size of the cache by 4x.

C Increasing the number of entries in the cache by 4x.

Correct Answer: **B and C**

7. Exceptional Control Flow [8 pts] (parts a–d)

(a) [2 pts] What does *asynchronous* mean in the context of exceptional control flow?

Asynchronous means that the exception is not caused by or associated with the execution of an instruction of the running process.

Below are 4 simplified descriptions of how different exceptional situations are handled:

Sequence A:

- current instruction (at PC) triggers handler
- control passes to handler
- execution never returns to the user process

Sequence B:

- current instruction (at PC) triggers handler
- control passes to handler
- execution returns to user process
- current instruction (at PC) executes once more

Sequence C:

- current instruction (at PC) completes
- control passes to handler
- execution returns to user process
- next instruction (at PC+4) executes

Sequence D:

- current instruction (at PC) triggers handler
- control passes to handler
- execution returns to user process
- next instruction (at PC+4) executes

Circle One:

(b) [2 pts] Which sequence best describes a **system call**? A B C D

(c) [2 pts] Which sequence best describes a **fault**? A B C D

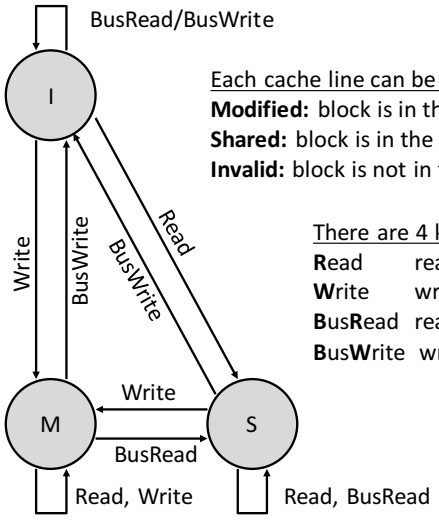
(d) [2 pts] Which sequence best describes an **interrupt**? A B C D

Answers: D, B, C

8. Multicore [24 pts] (parts a–c)

(a) [12 pts] This question concerns a 2-way set-associative 32-byte cache with 8-byte blocks. This is in a multiprocessor. You are simulating the cache associated with CPU 0. CPU 0 shares a bus with CPU 1. Reference addresses come from both the local processor (CPU 0) and other processors (CPU 1 via the bus). The cache uses an LRU replacement policy but would *always* replace an invalid block before a valid one.

This multiprocessor uses a simplified 3-state MSI protocol as defined here:



Each cache line can be in 1 of 3 states:
Modified: block is in the cache, has been written
Shared: block is in the cache, has only been read
Invalid: block is not in the cache

There are 4 kinds of events:

Read	read by CPU 0
Write	write by CPU 0
BusRead	read miss by CPU 1 on the bus
BusWrite	write miss by CPU 1 on the bus

There are 5 kinds of misses: cold, conflict, capacity, upgrade and coherence.

The initial contents of the cache (base block address : state) are given in the first line of the table. Assume that the initial blocks were accessed in the following order: 000, 050, 020, 070. For each Event, fill in:
(1) whether the event results in a Hit or Miss if applicable. (If a miss, what kind?)
(2) what bus event (if any) must be generated for CPU 1 to see
If the Event triggers a change in the cache, show that changed state on the next line. To simplify the notation, we use an *octal* block address. For example: $046_{octal} = 000\ \ 100\ \ 110_{binary}$. The first access has been done for you.

Cache state prior to access						Event addr:action	Hit or Miss? (if applicable)	Bus Event addr:action (if applicable)
Set 0			Set 1					
Way 0	LRU	Way 1	Way 0	LRU	Way 1			
000:S	0	020:S	050:M	0	070:S	051:R	hit	—
				1		034:W	cold miss	030:BW
				0	030:M	024:W	upgrade miss	020:BW
		020:M				020:BW	—	—
		(020):I				072:R	conflict miss	070:BR
			070:S	1		022:W	coherence miss	020:BW
		020:M						

Students might change this LRU bit to 1 so as to indicate that Way 1 should be evicted next. This is not required, but it will not be considered incorrect either.

(b) [4 pts] Most modern cache coherence protocols also include an E (Exclusive) state. A cache line in the Exclusive state means (1) this local cache has the *only* cached copy of the data and (2) this copy is clean. Under the MESI protocol, cache lines begin in the Invalid state and a Read can transition from I to E or from I to S.

(2 points) Under what conditions would a Read transition the cache line from

the I state to the S state? (Limit your answer to 1 sentence.)

If the data associated with that address is already cached in another core's cache.

(2 points) What is the E state designed to prevent? (Why is it useful?) Do not write more than 2 sentences.

It removes upgrade misses that occur when a cache line is in the shared state in one cache only and that core tries to write to the cache line. The core can simply transition from the E state to the M state without experiencing a miss.

(c) [8 pts] Many CPU architectures provide a Compare-And-Swap (CAS) instruction. It is a single, atomic assembly instruction that can be called with three register inputs (one of which is also a register output) with the semantics:
`CAS addr, compare_value, new_value`

- `addr`: an address in memory
- `compare_value`: as an input, it is the value you expect to be at `addr`; as an output, it is 1 for success (meaning the compare was a match and the new-value was stored at `addr`) and 0 for failure (meaning the compare was not a match and no new value was written to `addr`)
- `new_value`: the new value you will write to `addr` *if the compare_value matched*

Or if you prefer C to English:

```
ATOMIC void CAS(int *addr, int *comp_val, int new_val) {
    if (*addr != *comp_val) {
        *comp_val = 0;
    } else {
    *addr = new_val;
    *comp_val = 1;
    }
}
```

Below is the MIPS implementation of `mutex_lock`. Rewrite the body of this function so that it uses a `CAS` instruction instead of LL/SC. The function behavior should not change. The address of the lock is the argument to the function.

```
mutex_lock(int *m) {
  test:  LI $t0, 1
         LL $t1, 0($a0)
         BNEZ $t1, test
         SC $t0, 0($a0)
         BEQZ $t0, test
}
```

```
mutex_lock(int *m) {
   test:  LI $t0, 0   // if zero (free)
          LI $t1, 1   // grab lock (1)
          CAS $a0,$t0,$t1
          BEQZ $t0, test
```