Prelim 2, Version 1

Computer Science 3410, Cornell University
Spring 2017, Professor Bracy 4 May 2017

Solutions

Problem	Topic	Points	Score
1	Multiple Choice	18	
2	Translation	5	
3	Think of the Possibilities!	8	
4	The Need for Speed	12	
5	Call Me Maybe	15	
6	Batman and Robin	12	
7	Cache on Hand?	10	
8	Miss Congeniality	10	
Total		90	

Your Name ______________________________

Your NetID __________

1. Multiple Choice [18 pts] There is only 1 correct answer per question. If you write down multiple answers, we will only grade the first one.

   (a) [2 pts] **Multi Level Page Tables.** What is *not* a reason for implementing a multi-level page table?

      A. Multi-Level Page Tables support a faster translation than Single Level Page Tables.

      B. Unlike Single Level Page Tables, Multi-Level Page tables do not need to be contiguous in memory.

      C. Unlike Single Level Page Tables, Multi-Level Page tables do not need to have a PTE allocated for every virtual page number.

      D. Unlike Single Level Page Tables, Multi-Level Page tables allow two processes to map distinct virtual page numbers to the same physical page.

      E. In the common case, a Multi-Level Page Tables takes up fewer pages in memory than Single Level Page Tables.

      Correct Answer: **A**

   (b) [2 pts] **Conflict Misses.** You can have a conflict miss in a fully associative cache.

      A. True

      B. False

      C. Cannot be answered with the information given

      Correct Answer: **B**

   (c) [2 pts] **Coherence.** Two cores sharing a single cache still need a coherence protocol for their shared cache in order to maintain coherence.

      A. True

      B. False

      C. Cannot be answered with the information given

      Correct Answer: **B**

(d) [2 pts] **C Basics.** What value is printed in the last line of `main`?

```
void set1(int val1, int *val2) {
   *val2 = val1 + 6;
}

int set2(int val1, int *val2) {
   return (*val2 + 3);
}

int main() {
   int x = 5;
   int *y = &x;
   set1(set2(x, y), y);
   printf(%d\n, x);
}
```

A. 8
B. 11
C. 14
D. It depends on the address of x.
E. None of the above.

Correct Answer: **C**

(e) [2 pts] **VI Protocol.** The VI protocol's biggest drawback is the upgrade misses.

A. True
B. False
C. Cannot be answered with the information given

Correct Answer: **B**

(f) [2 pts] **Asynchronicity.** Which of these exceptional events are asynchronous?

A. Software Exception
B. Hardware Interrupt
C. System Call
D. A and C
E. B and C

Correct Answer: **B**

(g) [2 pts] **Synchronization.** Which of the following statements about Synchronization is *false*?

A. LL/SC is helpful when cooperating threads of a parallel program need to synchronize to get proper behavior for reading and writing shared data.

B. LL/SC is helpful When cooperating processes on a uniprocessor need to synchronize for reading and writing shared data.

C. LL/SC are an example of hardware support for synchronization.

D. A programmer can benefit from LL/SC instructions without understanding them well enough to use them correctly at the assembly level.

E. LL/SC are the instructions a programmer should use to start and end a critical section.

Correct Answer: **E**

(h) [2 pts] **Exceptions.** A five-stage pipeline (IF, ID, EX, MEM, WB) executes the following instruction sequence:

```
A    lw $1, 0($2)     # page fault
B    XXX $1, $2, $1   # undefined instruction
C    add $1, $2, $1   # arithmetic overflow
D    sub $1, $2, $1   # hardware error
```

Which exception (A, B, C, or D) should be recognized first in the above sequence?

Correct Answer: **A**

(i) [2 pts] **Saving State.** In order to preserve the processor state of the user process, the OS handler saves both caller- and callee- saved registers after which of the following?

A. Software Exception

B. Hardware Interrupt

C. System Call

D. A and B

E. A, B, and C

Correct Answer: **B or D**

2. Translation [5 pts]

   Consider a machine with a 20-bit virtual address, a 16-bit physical address and 2KB pages in memory. Each page table entry is 4 bytes.

   (a) [1 pt] How large is the page offset?

   2KB $= 2^{11}$ bytes, so 11 bits for the offset

   (b) [2 pts] How many pages in memory would be required to store the entire single-level page table?

   virtual address is 20 bits, 11 bits are offset, so 9 bits are VPN
   need space for 2^9 entries, each entry is 2^2 bytes, so need 2^{11} bytes
   each page is exactly 2^{11} bytes, so we need exactly 1 page.

   (c) [2 pts] What would the benefit be of having a 2 level page table instead? Be specific.

   In this case there is actually no benefit to having a 2 level page table since the entire page table already fits in 1 page. Access time would just get slower and you would not save any space in memory. If you forced the page table into two levels, you'd actually end up using more memory.

3. Think of the Possibilities! [8 pts]

For each combination, fill in the table indicating whether the combination is:

A. Possible

B. Impossible

C. Possible but never detected

D. Cannot say with the given information

If you believe it is possible, state under what circumstance this would happen. If it is not possible, not detectable, or not knowable, explain why.

TLB	**Page Table**	**Cache**	Possible? (A-D)	*Explanation*
Miss	Hit	Hit	A	TLB misses, but entry found in page table; after retry, data is found in cache.
Hit	Hit	Miss	C	Possible, but Page Table is never really checked if TLB hits.
Miss	Miss	Miss	A	TLB misses followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	B	cannot have a translation in TLB if page is not present in memory.

4. The Need for Speed [12 pts] (parts a–d)

Consider a 1 GHz (1 cycle = 1 ns) pipelined MIPS processor with an instruction mix of 20% loads and 80% integer operations. The processor has an instruction cache with a 90% hit rate and a data cache with an 80% hit rate. Both caches are accessed in 1 cycle. It takes 10ns to access DRAM. Integer operations take 1 cycle to execute. In optimal circumstances, an instruction finishes every cycle, yielding a CPI of 1.

(a) [3 pts] What is the CPI of this machine?

baseline CPI = 1, but 10% of all instructions will incur a 10 cycle penalty for an i-cache miss, 20% of loads will incur a 10 cycle penalty for a d-cache miss: 1 + (.1 * 10) + (.2 * .2 * 10) =
1 + 1 + .4 = 2.4

(b) [3 pts] **Modification 1:** You double the size of the data cache, which increases its hit rate to 90%. In order to maintain a 1 cycle access time, the clock rate is slowed to 500MHz. DRAM speed is unchanged. What is the CPI of this modified machine?

baseline CPI = 1, but 10% of all instructions will incur a 5 cycle penalty for an i-cache miss, 10% of loads will incur a 5 cycle penalty for a d-cache miss: 1 + (.1 * 5) + (.1 * .2 * 5) =
1 + .5 + .1 = 1.6

(c) [4 pts] **Modification 2:** Instead of modifying the data cache (as in the previous section), you add a *unified* level 2 cache with a 4 cycle access time and a 90% hit rate. This L2 cache will be accessed *in parallel* with the first level caches. (L1 hits do not wait for L2 results.) Clock frequency remains unchanged from the original processor. What is the CPI of this modified machine?

in first part we had 14% of instructions incurring a 10 cycle penalty to go to memory
now these same 14% incur a different penalty:
3 additional cycles to go to the L2 (not 4 b/c it's done in parallel) and then 10% of these must still go to memory:
1 + (0.14 * (3 + .1 * 10)) =
1 + (0.14 * 4) = 1 + 0.56 = 1.56

(d) [2 pts] Which processor is faster: the original, Modified 1, or Modified 2? Explain.

Modified 2 is the fastest. Easy to see because it has the fastest CPI and fastest clock.

5. Call Me Maybe [15 pts] Consider the following typedef and function in C. The `main` calls function `get_point` which calls function `scale_y`, using the 3410 Calling Conventions. Note that the dot ('.') operator takes higher precedence than the ampersand (&) operator.

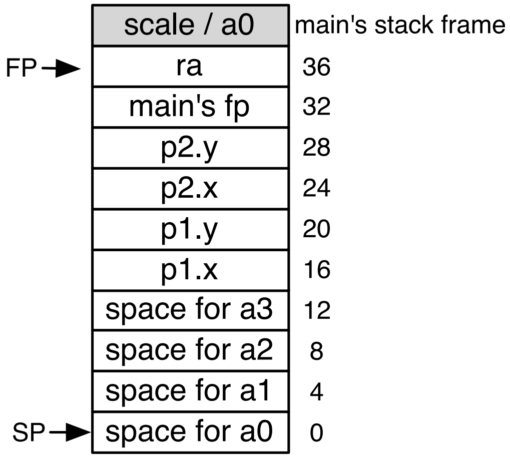
```
typedef struct point_t {
    int x;
    int y;
} rect_t;
```

```
1  void get_point(int scale, int val) {
2      point_t p1, p2;
3      p1.y = 1 + val;
4      p2.y = 3;
5      scale_y(scale, &p1.y, val);
6      p1.x = scale * p1.x * p2.y;
7      return &p1;
8  }
```

(a) [12 pts] In as few lines of code as possible, write the assembly for `get_point`, beginning with the prologue and **ending with line 5**. Do **not** use any `s` registers. To be eligible for partial credit, a line must have a comment, explaining what the line accomplishes.

```
Prologue:
ADDIU $sp, $sp, -40   # make stack frame
SW $ra,36($sp)        # store $ra
SW $fp, 32($sp)       # store $fp
ADDIU $fp, $sp, 36    # update $fp
SW $a0, 4($fp)        # save $a0/scale

Body:
ADDIU $t0, $a1, 1     # 1 + val
SW $t0 20($sp)        # store it into p1.y
ADDIU $t1, $0, 3      # 0 + 3
SW $t1 28($sp)        # store it into p2.y
MOV $a2, $a1          # val is 3rd arg
ADDIU $a1, $sp, 20    # &p1.y is 2nd arg
JAL scale_y           # call scale_y
```



	scale / a0	main's stack frame
FP →	ra	36
	main's fp	32
	p2.y	28
	p2.x	24
	p1.y	20
	p1.x	16
	space for a3	12
	space for a2	8
	space for a1	4
SP →	space for a0	0

Note: p1 and p2 must be on the stack because they need to be addressed. scale needs to be saved on to the stack b/c we don't know whether scale_y will clobber the value in a0. val does NOT need to be saved into another register besides a2 b/c it is not used after the call the scale_y. Storing a0 in get_point's stack frame is fine (although main did make room for it on the main stack frame).

(b) [1 pt] Which line in `get_point` is bad C code? line 6, 7, or 1 Line 1-and-7

(c) [2 pts] Explain in 1-2 sentences why.
(*We will only read your first 2 sentences.*)

Line 6 is bad practice because it uses an uninitialized value. Line 7 is bad practice to return an address to something that is allocated on a no-longer allocated stack frame. This memory location will likely be overwritten by the next function call, and writing to the location could create problems for future stack frames that rely on that location to hold specific data. Line 1 declares the function returning void yet Line 7 returns an int *. This would prevent compilation!

6. Batman and Robin [12 pts] You maybe have known the word ***dynamic*** from expressions like 'the dynamic duo' but in CS 3410 you have learned some very specific, technical meanings of the word dynamic. Choose **THREE** of the following 5 instances of the word **dynamic**. Explain what is meant by `dynamic` (as opposed to static) and answer the follow-up question. If you answer more than 3, only the first 3 will be graded.

- **Dynamic Random Access Memory (DRAM).**
  - What does dynamic mean? DRAM values require regular refresh, in particular they must be refreshed whenever they are read.
  - What is one way in which DRAM is better than SRAM? DRAM is denser and cheaper than SRAM.
- **Dynamic Instruction Count.**
  - What does dynamic mean? Dynamic refers to the number of instructions executed at runtime, as opposed to the size of the static binary executable
  - Why might one care more about the *static* instruction count? If one has very little storage space and were concerned about fitting the instructions themselves in memory.

- **Dynamically Linking.**
  - What does dynamic mean? The linking happens at runtime with shared libraries rather than when the program is compiled
  - What is one advantage of a dynamic linking? (1) size of static executable is smaller, (2) can benefit from updates to a library, (3) multiple processes can use same DLL at the same time
- **Dynamic Memory Allocation.**
  - What does dynamic mean? The memory is given during the execution of a program, on the heap rather than the stack
  - What is one advantage of allocating memory dynamically? (1) Can decide how much memory needed at runtime (2) memory stored on the heap can be passed between functions as it is not abandoned when a function returns (3) size of memory can change later (more flexibility)
- **Dynamic Scheduling.**
  - What does dynamic mean? The processor chooses what order to schedule instructions at runtime
  - What is one advantage of a static scheduling? Simpler hardware

7. Cache on Hand? [10 pts]

The following problem concerns basic cache lookups.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Physical addresses are 12 bits wide.
- The cache is 4-way set associative, with a 2-byte block size and 32 total lines.

(a) [3 pts] The box below shows the format of a physical address. Indicate how each bit would be used by filling in the boxes with ***O** for block offset*, ***I** for cache index*, and ***T** for cache tag*).

11	10	9	8	7	6	5	4	3	2	1	0
T	T	T	T	T	T	T	T	I	I	I	O

In this table, **all numbers are hexadecimal**. The contents of the cache are as follows:

4-way Set Associative Cache																
Index	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1
0	29	0	34	29	87	0	39	AE	7D	1	68	F2	8B	1	64	38
1	F3	1	0D	8F	3D	1	0C	3A	4A	1	A4	DB	D9	1	A5	3C
2	A7	1	E2	04	AB	1	D2	04	E3	0	3C	A4	01	0	EE	05
3	3B	0	AC	1F	E0	0	B5	70	3B	1	66	95	37	1	49	F3
4	80	1	60	35	2B	0	19	57	49	1	8D	0E	00	0	70	AB
5	EA	1	B4	17	CC	1	67	DB	8A	0	DE	AA	18	1	2C	D3
6	1C	0	3F	A4	01	0	3A	C1	F0	0	20	13	7F	1	DF	05
7	0F	0	00	FF	AF	1	B1	5F	99	0	AC	96	3A	1	22	79

For the given physical address, indicate the cache entry accessed and the cache byte value returned **in hex**. Indicate whether a cache miss occurs.

If there is a cache miss, enter "-" for "Cache Byte returned".

Physical address: `3B6`

(b) [2 pts] Physical address format (one bit per box)

11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	1	0	1	1	0

(c) [5 pts] Physical memory reference

Parameter	Value
Cache Offset (CO)	0x0
Cache Index (CI)	0x3
Cache Tag (CT)	0x3B
Cache Hit? (Y/N)	Y
Cache Byte returned	0x66

8. Miss Congeniality [10 pts]

Consider a dual core processor with 4-bit addresses. Each core has a private **8B** cache with 2 byte cache lines. Core 0's cache is fully associative. Core 1's cache is direct mapped.

The caches use an MSI coherence protocol. For simplicity, the diagram below shows only the full 4-bit address of the block-aligned memory address. Both caches happen to have been filled from left to right (Core 0's cache had accesses in the order: 1011, 1111, 0010, 0000. Core 1's cache had accesses in the order: 0000, 1010, 0101, 1111. The diagram below shows memory references at time 100, 101, and 102. Each reference has format `X:ABCD:Y` where `X` indicates the Core that the reference originated from and `Y` indicates whether it is a read or a write.

CORE 0 Cache state prior to access				t	Memory Reference	CORE 1 Cache state prior to access			
Way 0	Way 1	Way 2	Way 3			Set 0	Set 1	Set 2	Set 3
1010:S	1110:S	0010:S	0000:S	100	0:0011:R	0000:S	1010:S	0100:M	1110:S
				101	1:1101:R				
				102	1:1111:W			1100:S	
	1110:I			**103**	**???**				1110:M

Your job is to suggest 6 different Memory References that at time `103` that would be categorized in each of the following ways. The first one has been done for you. If no reference receives the desired categorization, please write 'not possible'. Note that your references do *not* happen one after the other. They represent 6 possible references for time `103` only.

Goal	**t=103 Memory Reference**
Hit	1:1110:R
Cold Miss	0:0100:R
Conflict Miss	1:0100:R
Capacity Miss	not possible
Coherence Miss	0:1110:R
Upgrade Miss	1:0000:W