CS2630: Computer Organization
Midterm practice

Since there are no past midterms to look at, this document provides some practice questions. The questions are the type you should expect.

The best way to use this practice is to take it like you are taking a real exam. One caveat is that the length of the practice exam is *not* representative of the length of the midterm, so you will probably take more than 75 minutes on the practice. Grade yourself with the solutions *after* you've taken this practice.

The exam will be open book/open note. No electronics. Show your work when appropriate.

Binary / number representation

1. How many things can you name uniquely with N bits? 2^N

2. What is the result of the following expression? Assume the integers are 32-bit. Give your answer in decimal.

   (6 << 2) | 1
   25

3. What is the least number of bits needed to store the number 0x4023 when using two's complement?

0100 0000 0010 0011
valid answer might be 15 or 16

4. X is a variable holding an integer. Give an expression using only X and bitwise operations that evaluates to 0 if X has a 0 in the least significant bit and 1 if X has a 1 in the least significant bit.

X & 1

5. Give the standard name for 2^{35} bytes. For example, 2^{10} is 1 kibi-byte.

   32 gibi-bytes

6. Give the decimal number -3 in binary using two's complement (you may show your answer using any number of bits that is sufficient to store the number):
   101 with 3 bits, 1101 with 4 bits, 11101 with 5 bits …

Memory organization

1. Suppose I have an architecture where addresses are 64 bits and integers are 32 bits.

Consider the following Java code.

```
int[] x = new int[3];
x[0] = 4;
x[1] = 5;
x[2] = 6;
```

Suppose the array starts at address 0x80.

a) Draw a diagram of memory after this code runs. You should label each byte with an address, and show the value of each byte (in hexadecimal).

Address	Contents
…	
0x80	0x04
0x81	0x00
0x82	0x00
0x83	0x00
0x84	0x05
0x85	0x00
0x86	0x00
0x87	0x00
0x88	0x06
0x89	0x00
0x8A	0x00
0x8B	0x00
…	

b) What is the address of the last word of the array?

0x88

2. Match the region of a running program's memory with what to store there. Some regions may have zero, one, or multiple answers.

   what to store:

   a) an array of characters representing a message that might be printed
   b) a constant
   c) return address of a procedure call

d) the instructions for the program
e) a binary tree that may have elements inserted while the program is running
f) values for registers that need to be preserved across a function call

regions:

stack_____c, f_____

heap_____e_____

.data_____a,b_____

.text_____d_____

MIPS programming

1. Write one instruction (not a pseudo instruction) to set register `$t5` to zero.

2. Suppose we want to divide (integer division) the positive integer stored in register `$s0` by 16 and save the result in register `$s1`. Circle the one instruction that does that.

```
(a) sll $s1, $s0, 4
(b) sra $s1, $s0, 2
(c) sra $s1, $s0, 4 ←
(d) sll $s0, $s1, 4
(e) None of the above
```

3. The following MIPS code tries to reverse the contents of array A of words. The base address starts in register `$a0` and the length of the array starts in register `$a1`. Fill in the blanks (there are five) to make the program work.

`reverse:`	`add $t0, $zero, $a0`	`# t0=a0 points to start of A`
	`addi $t1, $a1, -1`	`# t1 = a1-1`
	`sll$t1,$t1,__2__`	`# t1=t1x4`
	`add $t1, $a0, $t1`	`# t1 points to last element of A`

loop:	lw $t2,0($t0)	# t2 gets 1st element of array A
	lw $t3,0($t1)	# t3 gets last element of array A
	sw __$t2_, 0($t1)	# store ___ into end of A
	sw _$t3__, 0($t0)	# store ___ into start of A
	addi $t1, $t1, _-4__	# Update $t1
	addi $t0, $t0, _4__	# Update $t0
	bgt $t1, $t0, loop	# Continue until $t0>=$t1

4. In a MIPS program, assume that the program counter PC currently contains address 0x000000F0 and program needs to jump to the label LL, which is labeling address 0x3FFFFFF0. If you use an instruction like j LL, then you can easily do it, and this is fine when you are writing a program. However, in this case, it cannot be a TAL instruction in the J-type instruction format J. Here is the question:

   (a) Why can't j LL be a basic machine instruction in the J-type format? [Hint: See how far the program counter has to jump]

jump address = (PC+4)[31:28], value_for_address_field, 00

So if the j LL instruction is at 0x000000F0, then it can only jump to addresses that look like 0x0XXXXXXX.

   (b) What sequence of TAL instructions should the assembler translate j LL to?

lui $at, 0x3FFF
ori $at, 0xFFF0
jr $at

5. In class, we discussed how each card in a 52-card deck (4 suits and 13 values A,2,3,…,K) can be represented by a 6-bit binary number.

   Below is how a card is stored in a 32-bit integer

   26 unused bits| _ _ _ _ | _ _
   fvalue suit

   a)
   Suppose we want to write a function `check_fvalue`. The signature of the function as it would appear in Java is:

   ```
   int check_fvalue(int cardA, int cardB)
   ```

   CardA and CardB are integers such that the 6-bit representation of the card is in the least significant bits and the rest of the bits are unknown. The function returns 0 if the cards had different **fvalues** and 1 if the cards had the same **fvalue**. Write `check_fvalue` as a function in MIPS. Use the conventions for functions written in MIPS.

```
check_fvalue:
srl $t0, $a0, 2
srl $t1, $a1, 2
beq $t0,$t1,same
addiu $v0, $zero, 0
jr $ra
same:
addiu $v0, $zero, 1
jr $ra
```

   b) Write MIPS code that calls the check_value on a card with the fvalue of 7 and a card with the fvalue of 2. The suit of the cards can be anything.

```
addiu $a0, $zero, 7
sll $a0, $a0, 2
addiu $a1, $zero, 2
sll $a1, $a1, 2
jal check_fvalue
```

Compile/Assemble/Link/Load

1. For each step, list the tasks that step is responsible for. Some tasks might belong to multiple steps.

tasks:

a) translate a source file written in a higher level language to assembly code
b) turn branch labels into addresses
c) copy the .text and .data segments from disk into memory
d) for each unresolved reference to a symbol, find the definition of the symbol
e) jump to the main() function to start the program
f) translate pseudo instructions to real instructions
g) optimize the code

steps:

compile:____a,g_____

assemble:____b,f_____

link:_____d_____

load:_____c,e_____

because of dynamic linking, you can also put “d” in load