PRACTICE MIDTERM EXAM FOR COMPSCI/ECE 250

ANSWER-FREE version

Summer 2016, Prof. Tyler Bletsch

These questions are pasted in from various sources; ignore the question numbers and point values. Answers to ~~SOME~~ *none of* questions are provided in blue. For those without answers, you're encouraged to post your proposed answers, and the teaching staff will confirm/correct them for the benefit of the whole class.

1) [10 points]

(a) Add the following base-10 numbers using **6-bit** 2s complement math: -3, -4. **Show your work!**

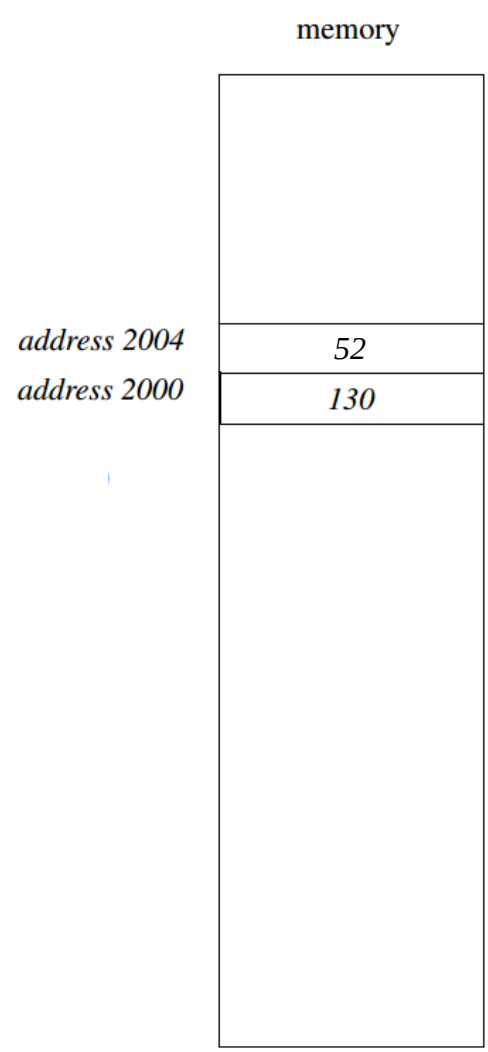
2) Assume that $2 = 2000 and $3=12. Assume that memory holds the values at the addresses shown on the left. "lw" = load word, and "sw" = store word.

(a) If the computer executes sw $3, 4($2), then what is the value of $3 after this instruction?

(b) If, after the instruction in part (a), the computer executes lw $3, 0($2), what is the value of $3 after this instruction?

(c) What single instruction could you use to write the value in $5 into address 2008?

(d) What single instruction could you use to read the word of memory at address 1996 and put the result in $8?



3) [10] The IEEE 754 floating point standard specifies that 32-bit floating point numbers have one sign bit, an 8-bit exponent (with a bias of 127), and a 23-bit significand (with an implicit "1"). Represent the number -11.75 in this format.

4) [10] The following questions are based on the following code snippet.

(a) What is *(array+7) ? Please give its datatype and its value.

(b) On a MIPS machine, how big (how many bytes) is the variable array?

(c) On a MIPS machine, how big (how many bytes) is array[2]?

(c) What is the datatype of fun?

```
int* array = (int*) malloc(42*sizeof(int));
int** fun = &array;
for (int i=0; i<42; i++){
    array[i] = i*i;
}
free (array);
```

5) [25] Convert the following C code for the function foo() into MIPS code. **Use appropriate MIPS conventions for procedure calls**, including the passing of arguments and return values, as well as the saving/restoring of registers. Assume that there are 2 argument registers ($a0-$a1), 2 return value registers ($v0-$v1), 3 general-purpose callee-saved registers ($s0-$s2), and 3 general-purpose caller-saved registers ($t0-$t2). Assume $ra is callee-saved. The C code is obviously somewhat silly and unoptimized, but YOU MAY NOT OPTIMIZE IT -- you must simply translate it as is.

```
1: int foo (int num){
2:   int temp = 0; //temp MUST be held in $t0
3:   if (num <0) {
4:       temp = num + 2;
5:   }else{
6:       temp = num - 2;
7:   }
8:   int sumA = bar(temp); // sumA MUST be held in $s0
9:   int sumB = sumA + temp + num;// sumB MUST be held in $s1
10: return (sumB + 2);
11:}

12: int bar (int arg){
```

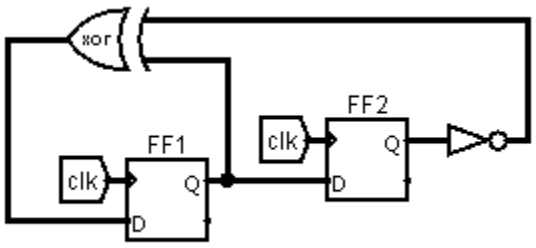
line(s) of C	instruction(s)	what code MUST do (if not obvious from C code)
1		create stack frame large enough for callee-saved and caller-saved registers; save callee-saved registers (ONLY necessary ones)
2		
3-7		
8		save caller-saved registers (ONLY necessary ones); call bar() with appropriate arguments
after line 8		restore caller-saved registers; get value returned from bar() and put it in appropriate place
9		
10		pass return value back to whoever called foo(); restore callee-saved registers; destroy stack frame; return to caller

1) [10 points] Write the truth table for the output of the following boolean expression that has three inputs (a, b, c): output = $abc + \bar{a}c + b\bar{c}$

2) [10 points] Convert the following truth table into a boolean expression in product-of-sums format. Note that there are three inputs (a,b,c) and one output. Do NOT simplify or optimize in any way.

a	b	c	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Question 1 [20 points]: Consider the circuit below. Assuming the two flip flop start with a value of zero, what will the state of the flip flips be for the clock cycles shown? The initial state is done for you.



Clock cycle	FF1	FF2
0	0	0
1		
2		
3		

Same exercise, but with a different starting condition:

Clock cycle	FF1	FF2
0	1	1
1		
2		

Question 2 [17 points]: Draw a finite state machine that will output a 1 if and only if a sequence of characters of the following form is received: exactly one 'D', *zero* or more 'O's, and exactly one 'G'. (If you happen to know regular expression notation, this is the expression /DO*G/.) Examples of matching inputs include: "DG", "DOG", "DOOOOG". Your machine can be of the Mealy or Moore variety. It doesn't matter what your machine does after it outputs 1.