PRACTICE MIDTERM EXAM FOR COMPSCI/ECE 250

Revised 2021, Tyler Bletsch

These questions are pasted in from various sources; ignore the question numbers and point values. Answers to SOME questions are provided in blue.

Practice question 1

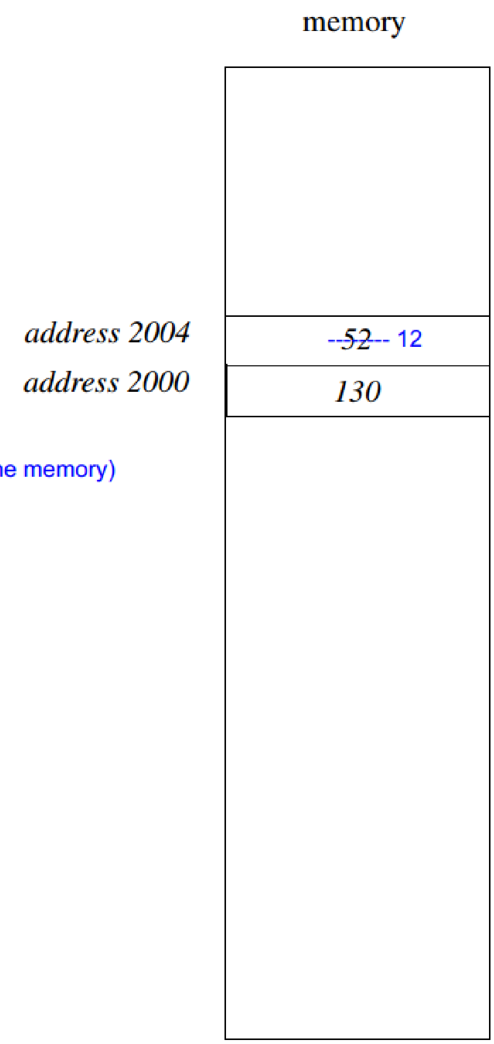
1) [10 points]

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

```
To get -3 in binary, start with 3 and negate (flip all bits and add one):

    000011
    111100  < bits flipped
    111101  < added one
    ^ this is -3

Same to get -4 in binary:

    000100
    111011  < bits flipped
    111100  < added one
    ^ this is -4 in binary

Now we add:

   1111     < carries
    111101
  + 111100
  --------
    111001  < sum


Check our work -- let's convert the sum to decimal. first we negate it to
make it positive:

    111001
    000110  < bits flipped
    000111  < added one
    ^ this is the negation of our sum

    111 in binary is 7 in decimal

    this makes sense, as -3 + -4 = -7
```

Practice question 2

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?

12

(the store doesnt change the register, it changes the memory)

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

130

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

sw $5, 8($2)

or as a joke answer: sw $5, 1878($3)

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

lw $8, -4($2)

Practice question 3

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.

Sign bit: 1 (negative)
Fractional representation: -11 3/4
Binary representation: -1011.11
Binary representation, normalized: -1.01111 * 2^3
Mantissa with the first one removed: 01111
Exponent with bias added: 3+127 = 130
Biased exponent in binary: 10000010

1 10000010 01111000000000000000000

Practice question 4

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

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

Same as array[7]
Type: int
Value: 49

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

The variable array, like all pointers on a system with 32-bit words, is 32-bits long, which is 4 bytes long.

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

It's the size of an integer, which on MIPS, is 32-bits, or 4 bytes.

(c) What is the datatype of fun?

int**

(A pointer to a pointer to an int. Size is still 4 bytes, since it's a pointer)

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

Practice question 5

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.

```
int foo (int num){
  int temp = 0; //temp MUST be held in $t0
  if (num <0) {
       temp = num + 2;
  }else{
       temp = num - 2;
  }
  int sumA = bar(temp); // sumA MUST be held in $s0
  int sumB = sumA + temp + num;// sumB MUST be held in $s1
return (sumB + 2);
}


int bar (int arg){
```

line(s) of C	instruction(s)	what code MUST do (if not obvious from C code)

1	# need 20 bytes for s0,s1,t0,t1,ra
	# why t1 even though its not needed in the problem?
	# because i need to backup a0 before the call
	addiu $sp,$sp,-20
	sw $s0,0($sp)
	sw $s1,4($sp)
	sw $ra, 8($sp)
		create stack frame large enough for callee-saved and caller-saved registers; save callee-saved registers (ONLY necessary ones)

2	li $t0, $t0, 0 # alternately, i could do "move $t0,$0"	

3-7	bgez $a0, else # invert the compare to get to the else
	#then
	addi $t0, $a0, 2
	j end_if # bypass the else
	else:
	addi $t0, $a0, -2
	end_if:	

8	move $t1,$a0 # backup num
	move $a0, $t0
	sw $t0, 12($sp)
	sw $t1, 16($sp)
	jal bar
		save caller-saved registers (ONLY necessary ones); call bar() with appropriate arguments

after line 8	lw $t0, 12($sp)
	lw $t1, 16($sp)
	mov $s0, $v0
		restore caller-saved registers; get value returned from bar() and put it in appropriate place

9	add $s1, $s0, $t0 # sumA+temp add $s1, $s1, $t1 # += num	

10	addi $v0, $s1, 2
	lw $s0,0($sp)
	lw $s1,4($sp)
	lw $ra, 8($sp)
	addiu $sp, $sp, 20
	jr $ra
		pass return value back to whoever called foo(); restore callee-saved registers; destroy stack frame; return to caller

Practice question 6

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

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

Practice question 7

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

(!a & !b & !c) | (!a & b & c) | (a & !b & c) | (a & b & !c) | (a & b & c)

Practice question 8

Simplify this expression; axioms are provided ->

(!A & B & C) | (A & B & !C) | (A & B & C)

Name	Definition
Identity law	1 & A = A 0 | A = A
Null law	0 & A = 0 1 | A = 1
Idempotent law	A & A = A A | A = A
Inverse law	A & !A = 0 A | !A = 1
Commutative law	A & B = B & A A | B = B | A
Associative law	(A&B) & C = A & (B&C) (A|B) | C = A | (B|C)
Distributive law	A | (B&C) = (A|B) & (A|C) A & (B|C) = (A&B) | (A&C)
Absorption law	A & (A|B) = A A | (A&B) = A
De Morgan's law	!(A&B) = !A | !B !(A|B) = !A & !B
Double negation law	!!A = A

Factor (A&B)

(!A & B & C) | ((A & B) & (!C | C))

Inverse law

(!A & B & C) | ((A & B) & true)

Identity law

(!A & B & C) | (A & B)

Factor B

B & ((!A & C) | A)

Distribute A

B & ((!A | A) & (C | A))

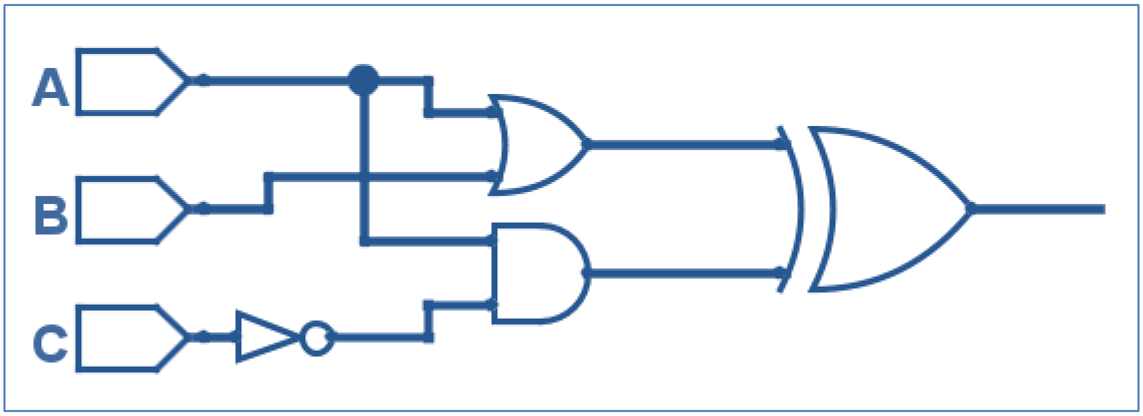
Inverse law

B & (true & (C | A))

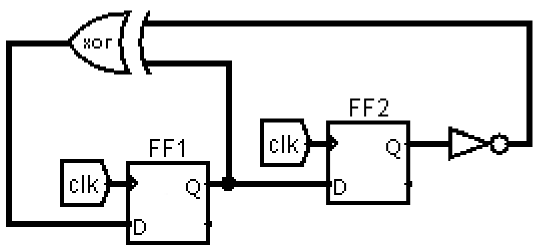
Identity law

B & (C | A)

Practice question 9: Sketch a circuit representation of the expression (A | B) ^ (A & !C)



Practice question 10: 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.

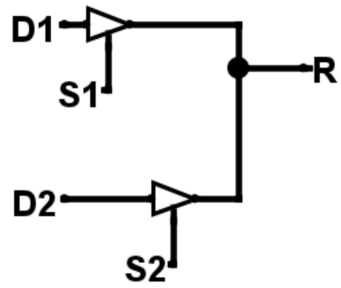


Clock cycle	FF1	FF2
0	0	0
1	1	0
2	0	1
3	0	0

Same exercise, but with a different starting condition:

Clock cycle	FF1	FF2
0	1	1
1	1	1
2	1	1

<u>Practice question 11:</u> The circuit below shows two tri-state buffers.



(a) How could you make D1's value appear on output R?
Set S1 to 1 and S2 to 0.

(b) How could you make D2's value appear on output R?
Set S1 to 0 and S2 to 1.

(c) How could you make the output R be in the high-impedance ("Z") state?
Set both S1 and S2 to 0.

(d) How could you cause a short circuit?
Set D1 to 1 and D2 to 0, then turn on both S1 and S2.

<u>Practice question 12</u>: 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.