CS250 Spring 2007 Midterm Exam Solutions, March 7, 2007
Prof. Chris Clifton

Time will be tight. If you spend more than the recommended time on any question, **go on to the next one**. If you can't answer it in the recommended time, you are either going in to too much detail or the question is material you don't know well. You can skip one or two parts and still demonstrate what I believe to be an A-level understanding of the material.

Keep in mind that I am concerned about your knowledge of concepts. If you show that you understand the material, but don't quite get the right answer, I can give partial credit. So describe (briefly) your reasoning if you aren't sure of your answer.

Note: It is okay to use abbreviations in your answers, as long as the abbreviations are unambiguous and reasonably obvious.

Solutions (not necessarily the only correct ones) given in italics, along with a rough anticipated grading strategy, subject to change.

1 Integer Representations (16 minutes, 10 points)

1.1 Converting binary to decimal (3 minutes, 3 points)

1. Given the following 8 bit unsigned binary integer, give the corresponding hexadecimal and decimal values:

   `0 0 1 1 0 0 1 1`

   Hex: *0x33* (1 point) Decimal: *51* (1 point)

2. What would the decimal value be if the above number instead represented an 8 bit signed (2's complement) integer?

   The same, 51 (1 point)

1.2 Converting decimal to binary (6 minutes, 3 points)

1. Give the 8 bit 2's complement representation of the integer -23.
   1 1 1 0 1 0 0 1 (1 point for sign, 1 for value)

2. Give the 32 bit 2's complement representation of the integer -23.
   11111111111111111111111 1 1 1 0 1 0 0 1 (1 point for correct sign extension)

1.3 Arithmetic (7 minutes, 4 points)

1. Compute the sum of the following two signed (2's complement) binary numbers. Give your answer as a binary value. Assume you are using a 16 bit machine (i.e., not only inputs, but also the result, is limited to 16 bits.)

   ```
   0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 1
   1 1 1 0 0 1 1 1 1 0 0 0 1 1 1 0
   ```

   Answer:

   ```
   0 0 1 1 0 0 1 1 0 1 0 0 0 0 1 1
   ```

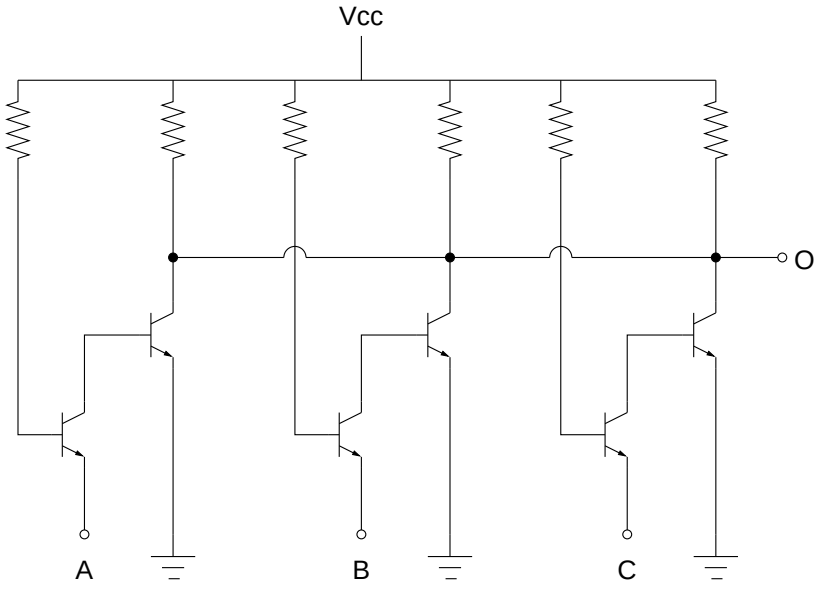
   (1 point for showing addition, 1 for getting it right / 16 bits.)

2. Would your answer be different if they were unsigned numbers? Explain your answer, in terms of how this might be an issue on a real computer and how it might be handled.

   The addition wouldn't be different, (1 point) *but the answer would no longer be correct (it is larger than could be represented in 16 bits.) An "overflow" flag should be set to indicate this.* (1 point for noting either.)

2 Circuits (6 minutes, 5 points)

You are given the following circuit:



1. Complete the truth table for the above circuit:

A	B	C	O
0	*0*	0	*1*
0	*0*	*1*	*0*
0	*1*	*0*	*0*
0	*1*	*1*	*0*
1	*0*	*0*	*0*
1	*0*	*1*	*0*
1	*1*	*0*	*0*
1	*1*	*1*	*0*

   (1 point for getting the inputs of the truth table filled in correctly, 1 for getting symmetry (and, or, nand, or nor), 1 for getting it all right.)

2. Give a boolean logic formula that matches your truth table. (You will get this part right as long as the formula is correct for your truth table, even if it doesn't match the circuit.)

   $\overline{A + B + C}$ (1 point for getting somewhere close to your truth table (right idea, but a mistake here or there), 1 point i

This is an interesting hybrid RTL/TTL. The power draw is low as long as the output is 1, but the key advantage is that very little power is required on the inputs, so it can have high fan-out (one output drives many inputs.)

3 Memory (16 minutes, 9 points)

You are given the following C structure:

```
struct s {
  char c;
  int  i;
  char d;
  struct s *next;
} stuff = { 'A', 65537, 'B', 0 };
```

Below are three possible assembly language representations of the above definition, along with memory dumps showing how each appears in memory. For each, give a short description of reasons why it might be a good idea to use that representation, and disadvantages to that representation. While the examples are not specific to any particular machine, it may help you to think in terms of the MIPS architecture. Full credit answers will discuss the reasons behind your answer - e.g., an answer like "this won't give the right result" must be backed up with reasons why in terms of processor/memory architecture.

Hint: You should be able to answer the question by looking at either the code or the memory dump. Also, to make your life easier the memory has been initialized to all 1's.

```
        .data   0x0000FF10
        .align  0
stuff:  .byte   65
        .word   65537
        .byte   66
        .word   0
```

Address	*Memory*			
0000FF00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0000FF10	41000100	01420000	0000FFFF	FFFFFFFF
0000FF20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

Discussion: *This is a compact representation with no wasted space.* (1 point) *While optimal for memory usage, it splits the integer i and pointer next across word boundaries,* (1 point) *which makes it difficult to load (for example, loading i from memory would require multiple instructions, as loading a single word from address 0000FF11 will fail.)* (1-2 points) (1 point for demonstrating you recognize the values in the memory dump.)

```
        .data   0x0000FF10
        .align  2
stuff:  .byte   65
        .word   65537
        .byte   66
        .word   0
```

Address	*Memory*			
0000FF00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0000FF10	41FFFFFF	00010001	42FFFFFF	00000000
0000FF20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

Discussion: *This is representation allows i and next to be loaded in a single instruction,* (1-2 points) *as they are aligned on a word boundary.* (1 point) *However, it does use more memory.* (1 point)

```
        .data   0x0000FF10
        .align  3
stuff:  .byte   65
        .align  3
        .word   65537
        .align  3
        .byte   66
        .align  3
        .word   0
```

Address	*Memory*			
0000FF00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0000FF10	41FFFFFF	FFFFFFFF	00010001	FFFFFFFF
0000FF20	42FFFFFF	FFFFFFFF	00000000	FFFFFFFF

Discussion: *This double-word alignment takes still more space.* (1 point) *For reading, it is unlikely to have an advantage.* (1 point) *But with some architectures, it may improve writes.* (1 point) *For example, if our bus is 8 bytes wide and we wanted to write a new value for i, the single word alignment would have to read c and i, change i, then write both values (as the bus always process 8 bytes at a time.) With this*

double-word alignment, we could just write the new value of i. (1 point) *(There may be some issues to be worked out, such as telling the memory controller what to place in the "unused" space.)*

4 Floating Point (8 minutes, 5 points)

This question will make use of the following two 32 bit IEEE floating point numbers:

```
      |               |
  X:  0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0
  Y:  0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0
```

Answer to 4.2 below - only the "X+Y" shown on exam.

```
X+Y:  0 0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0
```

4.1 Estimation (2 minute, 3 points)

Which range is X in? For full credit, circle the correct answer. If you aren't sure, you may circle up to three, and you will get one point if you include the correct answer.

1. $100.0 \leq X$
2. $1.0 \leq X < 100.0$
3. $0.0 \leq X < 1.0$ *Correct.*
4. $-1.0 \leq X < 0.0$
5. $-100.0 \leq X < -1.0$
6. $X < -100.0$

1 point for realizing the sign or that the magnitude is small. Full credit for realizing both.

4.2 Arithmetic (4 minutes, 2 points)

Compute the correct floating point result for the addition $X + Y$. Your answer should be given in binary, as a 32 bit IEEE floating point value. (You may find it easier to write your answer above.) WARNING: If this takes you more than two minutes, you are doing it the hard way - go back and make sure you have completed the rest of the test before coming back to this.

Answer above. (1 point for realizing you can't just add exponent - either getting the exponent right, or shifting mantissa when adding will get one point. Two points for getting it right.) *The key is realizing that if the exponents agree, you can just do binary addition of the mantissa. The exponents don't agree, but if you multiply by 2/2, you can add 1 to the exponent of Y (multiply by 2) and divide the mantissa by 2 (shift right). Then the exponents agree, and it is simple. As to computing the decimal equivalent - if you tried to do that, you probably aren't done yet.*