Name: CS520

Midterm Exam

8 March 2017, 80 minutes, 20 questions, 100 points

The exam is closed book and notes.

Please keep all electronic devices turned off and out of reach.

Note that a question may require *multiple* checked boxes for a correct answer. Checking *some* but not *all* of the required boxes will result in a *partial* answer worth only 2 of the 5 points. Checking any box that shouldn't be checked results in an *incorrect* answer, worth zero.

1. How would the following C string (i.e. null terminated) `"feed"` be represented using ASCII in the memory of a Little Endian machine with a byte-addressable memory? The bytes are shown below in increasing memory address order, left to right. The ASCII code for `'f'` is `0x66`. The ASCII code for `'e'` is `0x65`. The ASCII code for `'d'` is `0x64`. [5 pts]

   √ `0x66 0x65 0x65 0x64 0x00`.

   ◯ `0x66 0x65 0x65 0x64`.

   ◯ `0x64 0x65 0x65 0x66`.

   ◯ `0x00 0x64 0x65 0x65 0x66`.

   ◯ `0x64 0x65 0x65 0x66 0x00`.

   ◯ none of the above.

2. Interpret `0x27` and `0xF0` as 8-bit two's complement integers and add them together to produce an 8-bit two's complement integer. The result is: [5 pts]

   ◯ `0x18`.

   √ `0x17`.

   ◯ `0x117`.

   ◯ `0x23`.

   ◯ none of the above.

3. Consider the following C function: [5 pts]

```
unsigned int f(void)
{
  int i = 0;
  return *(unsigned char *) &i;
}
```

   On a machine with a byte-addressable memory, the function will:

   √ always return 0.

   ◯ always return 1.

   ◯ return 0 if the machine is little-endian and 1 otherwise.

   ◯ return 1 if the machine is little-endian and 0 otherwise.

   ◯ none of the above.

4. Which of the following 16-bit two's complement values will overflow when they are moved to an 8-bit two's complement container? [5 pts]

- ◯ `0xFFAA`.
- √ `0xAAFF`.
- √ `0xAFFA`.
- √ `0xFAFA`.

5. What would -69 look like as a 32-bit two's complement integer in the memory of little-endian machine? The bytes are shown below in increasing memory address order, left to right. [5 pts]

- √ `0xBB 0xFF 0xFF 0xFF`.
- ◯ `0xFF 0xFF 0xFF 0xBB`.
- ◯ `0x80 0x00 0x00 0x45`.
- ◯ `0xFF 0xFF 0xFF 0xBA`.
- ◯ none of the above.

6. Which of the following statements about IEEE single-precision floating-point are true? [5 pts]

- √ There are two infinity values, positive infinity and negative infinity.
- ◯ It stores the exponent as a two's complement value.
- √ Denormalized values have a stored exponent with all 0 bits and a stored significand that is not all 0 bits.
- ◯ It has one more negative value than positive value.

7. Interpret `0xC24C0000` as IEEE single-precision floating-point. What is its decimal value? [5 pts]

- ◯ `-12.75`.
- ◯ NaN.
- ◯ `-25.5`.
- √ `-51.0`.
- ◯ negative infinity.
- ◯ none of the above.

8. Interpret `0x807FFFFF` as IEEE single-precision floating-point. Which of the following are accurate descriptions of the value? [5 pts]

- √ represents a negative value.
- ◯ represents a zero value.
- ◯ represents an infinity value.
- ◯ represents a NaN value.
- √ represents a denormalized value.

9. Interpret `0x40A00001` and `0x40A00002` as IEEE single-precision floating-point. Add the two values together. What is the result? [5 pts]

- √ `0x41200002`.
- ◯ `0x41200001`.
- ◯ `0x41200003`.
- ◯ `0x41300002`.
- ◯ `0x41300001`.
- ◯ none of the above.

10. Interpret `0x07FFFFFD` as a 32-bit two's complement value. Convert it to IEEE single-precision floating point. What is the result? [5 pts]

- ◯ `0x4C7FFFFF`.
- ◯ `0x4C800000`.
- √ `0x4D000000`.
- ◯ `0x7F800000`.
- ◯ none of the above.

11. Represent this UTF-16 value, `0x1011`, in UTF-8. What is the result? [5 pts]

- ◯ `0x10 0x11`.
- ◯ `0xC1 0x91`.
- √ `0xE1 0x80 0x91`.
- ◯ `0xF1 0x80 0x80 0x91`.
- ◯ none of the above.

12. Represent this UTF-32 value, `0x00011011`, in UTF-16. What is the result? [5 pts]

- ◯ `0x0001 0x1011`.
- √ `0xD804 0xDC11`.
- ◯ `0xD810 0xDC11`.
- ◯ `0xD841 0xDC01`.
- ◯ none of the above.

13. Which of the following statements are true? [5 pts]

- ◯ The UTF-16 encoding of an ASCII character is always one byte long.
- √ The UTF-16 encoding of a Unicode character can be shorter than the UTF-8 encoding of the character.
- ◯ The UTF-32 encoding of a Unicode character can be shorter than the UTF-16 encoding of the character.
- ◯ The UTF-32 encoding of a Unicode character can be shorter than the UTF-8 encoding of the character.

14. Which of the following UTF-8 sequences contain at least one error? [5 pts]

- √ `0xF0 0x80 0x80 0x80`.
- ◯ `0x00 0x01 0x02 0x03`.
- √ `0xE1 0x81 0xF0`.
- ◯ `0xC9 0x8F`.

15. Consider the following vm520 program: [5 pts]

```
        jmp  bottom
        alloc  10
    bottom:
        halt
```

What is the encoding of the `jmp` instruction? The `jmp` opcode is `0x14`, and it uses format 2.

- ◯ `0x140A0000`.
- √ `0x0000A014`.
- ◯ `0x00000A14`.
- ◯ `0xFFFF6014`.
- ◯ none of the above.

16. Which of the following statements about a symbol in the insymbol section of a vm520 object file are true? [5 pts]

- √ It must be defined in the file.
- √ It must be exported in the file.
- √ It might be referenced in the file.
- ◯ It might be imported in the file.

17. Which of the following statements about a symbol in the outsymbol section of a vm520 object file are true? [5 pts]

- ◯ It might be defined in the file.
- ◯ It might be exported in the file.
- √ It must be referenced in the file.
- √ It must be imported in the file.

18. What is the encoding of the following vm520 instruction? The opcode for `ldimm` is 0x03, and it uses format 4. [5 pts]

```
        ldimm  r4 ,  -23
```

- ◯ `034FFFE8`.
- ◯ `FFFE8403`.
- ◯ `FFE80403`.
- √ `FFFE9403`.
- ◯ none of the above.

19. Which of the following statements about vm520 PC-relative addresses are true? [5 pts]

- ◯ When combining code from two different object files, a linker must add the length of the code in the first file to all PC-relative addresses in instructions in the second file.
- √ At execution time they are added to the PC to get the actual address.
- √ They can contain negative values.

20. Which of the following statements are true? [5 pts]

- √ An assembler has two passes because the definition of a label may come after its use.
- √ The first pass of an assembler determines the address of each label defined in the program.
- √ The second pass of an assembler is necessary to finish encoding the instructions.
- √ The assembler uses a symbol table to store labels and their addresses.