Name: CS520

Midterm Exam

17 October 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) `"012"` 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 `'0'` is `0x30`. The ASCII code for `'1'` is `0x31`. The ASCII code for `'2'` is `0x32`. [5 pts]

   ◯ `0x00 0x32 0x31 0x30`.

   ◯ `0x30 0x31 0x32`.

   ◯ `0x32 0x31 0x30`.

   √ `0x30 0x31 0x32 0x00`.

   ◯ none of the above.

2. Interpret `0xC1` and `0xF1` 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]

   ◯ `0xB3`.

   ◯ `0x1B2`.

   √ `0xB2`.

   ◯ `0x1B3`.

   ◯ none of the above.

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

```
unsigned int f(void)
{
  int i = 257;
  return *(unsigned char *) &i;
}
```

   On a machine with a byte-addressable memory, and with 32-bit integers, 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]

   √ `0xFEAA`.

   √ `0x00FF`.

   ◯ `0x007A`.

   √ `0x70FA`.

5. What would -13 look like as a 32-bit two's complement integer in the memory of big-endian machine? The bytes are shown below in increasing memory address order, left to right. [5 pts]

   ◯ `0xF3 0xFF 0xFF 0xFF`.

   ◯ `0x80 0x00 0x00 0x0D`.

   √ `0xFF 0xFF 0xFF 0xF3`.

   ◯ `0x0D 0x00 0x00 0x80`.

   ◯ none of the above.

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

   ◯ It stores the exponent as a one's complement value.

   √ There are two infinity values, positive infinity and negative infinity.

   √ Denormalized values have a stored exponent with all 0 bits and a stored significand that is not all 0 bits.

   ◯ NaN values have a stored exponent with all 1 bits and a stored significand that is all 0 bits.

7. Interpret `0xC15C0000` as IEEE single-precision floating-point. What is its decimal value? [5 pts]

   √ `-13.75`.

   ◯ NaN.

   ◯ `-27.5`.

   ◯ `-54.0`.

   ◯ positive infinity.

   ◯ none of the above.

8. Interpret `0x70811111` and `0x71822222` as IEEE single-precision floating-point values. Which of the following statements are true? [5 pts]

   ◯ `0x70811111` is greater than `0x71822222`

   √ `0x70811111` is less than `0x71822222`

   √ They are both positive values.

   ◯ They are both denormalized values.

   ◯ They are both NaN values.

9. Interpret `0x41A00001` and `0x40A00002` as IEEE single-precision floating-point. Add the two values together. What is the result? [5 pts]

   ◯ `0x41C80001`.

   √ `0x41C80002`.

   ◯ `0x41E40002`.

   ◯ `0x41E40001`.

   ◯ none of the above.

10. Interpret `0x4000000007FFFFFD` as an IEEE double-precision floating point value. Convert it to IEEE single-precision floating point. What is the result? [5 pts]

- ◯ `0x407FFFFF`.
- √ `0x40000000`.
- ◯ `0x4C800000`.
- ◯ `0x407FFFFD`.
- ◯ none of the above.

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

- ◯ `0xAB 0xBA`.
- √ `0xEA 0xAE 0xBA`.
- ◯ `0xEA 0xBA 0xBB`.
- ◯ none of the above.

12. Represent this UTF-32 value, `0x00101010`, in UTF-16. What is the result? [5 pts]

- √ `0xDBC4 0xDC10`
- ◯ `0x0010 0x1010`.
- ◯ `0xD810 0xDC11`.
- ◯ `0xDA42 0xDE10`.
- ◯ none of the above.

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

- √ The UTF-16 encoding of an ASCII character is always two bytes long.
- ◯ The UTF-8 encoding of an ASCII character is always two bytes long.
- √ The UTF-8 encoding of a Unicode character is sometimes shorter than the UTF-16 encoding of the character.
- ◯ The UTF-8 encoding of a Unicode character is always shorter than the UTF-32 encoding of the character.

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

- √ `0xF0 0x80 0x80 0x81`.
- √ `0xE0 0x80 0x81`.
- √ `0xC0 0x81`.
- ◯ `0x01`.

15. Which of the following statements about an ELF file are true? [5 pts]

- √ The .text section contains encoded instructions.
- ◯ An ELF file contains only ASCII characters.
- √ An ELF file starts with a "magic number", which identifies the file as an ELF file.
- √ An ELF file contains a section header table, which identifies the name of each section and where it begins in the file.

16. Consider the following RISC-V program: [5 pts]

```
      beq  x5,x5,bottom
      sbreak
      sbreak
      sbreak
      sbreak
   bottom:
      sbreak
```

What is the encoding of the `beq` instruction?

- √ `0x00528A63`.
- ◯ `0x00550563`.
- ◯ `0x00250520`.
- ◯ `0x63055500`.
- ◯ none of the above.

17. Which of the following statements about linking are true? [5 pts]

- ◯ If a symbol is an outsymbol in both input files to a linker, then it is an insymbol in the output file of the linker.
- √ A linker tries to match an outsymbol in one file with an insymbol in the other file.
- ◯ A linker combines two intput assembly language files into a single output assembly language file.
- ◯ If a symbol is an insymbol in both input files to a linker, then it is an outsymbol in the output file of the linker.

18. Decode the RISC-V instruction `0xFF068393`. How would this instruction be displayed by the Program 3 disassembler? [5 pts]

- ◯ `sw x9,x3,-8`.
- √ `addi x7,x13,-16`.
- ◯ `slti x3,x7,-8`.
- ◯ `addi x3,x7,-8`.
- ◯ none of the above.

19. Which of the following statements about 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 a reference to a label may come after its definition.
- ◯ The first pass of an assembler determines whether to encode address fields in instructions using PC-relative addresses.
- √ An assembler reads an assembly language source file as input and writes an object file as output.
- √ The second pass of an assembler determines the address of a referenced label by looking it up in a symbol table created by the first pass of the assembler.