Name: CS520

Midterm Exam

9 Mar 2016, 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) `"dead"` be represented using ASCII in the memory of a Big Endian machine with a byte-addressable memory? The bytes are shown below in increasing memory address order, left to right. The ASCII code for `'d'` is `0x64`. The ASCII code for `'e'` is `0x65`. The ASCII code for `'a'` is `0x61`. [5 pts]

   ◯ `0x64 0x65 0x61 0x64.`

   ◯ `0x64 0x61 0x65 0x64.`

   √ `0x64 0x65 0x61 0x64 0x00.`

   ◯ `0x64 0x61 0x65 0x64 0x00.`

   ◯ `0x00 0x64 0x61 0x65 0x64.`

   ◯ none of the above.

2. The algorithm for negating a two's complement integer value is: [5 pts]

   ◯ complement all bits.

   ◯ complement the sign bit.

   √ complement all bits and add one.

   ◯ subtract one.

   ◯ none of the above.

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

```
unsigned int f(void)
{
  int i = -1;
  return *(unsigned char *) &i;
}
```

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

   √ always return 255.

   ◯ always return -1.

   ◯ return 0 if the machine is little-endian and 255 otherwise.

   ◯ return -1 if the machine is little-endian and 255 otherwise.

   ◯ return 0 if the machine is big-endian and 255 otherwise.

   ◯ return -1 if the machine is big-endian and 255 otherwise.

   ◯ none of the above.

4. For a 32-bit integer on agate.cs.unh.edu, what is the hexadecimal for the most negative value (leftmost on the number line) that can be represented? [5 pts]

   ◯ `0xFFFFFFFF`.

   √ `0x80000000`.

   ◯ `0x80000001`.

   ◯ `0xEFFFFFFF`.

   ◯ none of the above.

5. What would -19 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]

   ◯ `0xEC 0xFF 0xFF 0xFF`.

   ◯ `0x80 0x00 0x00 0x13`.

   ◯ `0xFF 0xFF 0xFF 0xEC`.

   ◯ `0x13 0x00 0x00 0x80`.

   √ none of the above.

6. Which of these have two representations for zero? [5 pts]

   ◯ two's complement integer.

   √ one's complement integer.

   √ IEEE double-precision floating-point.

   √ IEEE single-precision floating-point.

7. Interpret `0x41CC0000` as IEEE single-precision floating-point. What is its the decimal value? [5 pts]

   ◯ `12.75`.

   ◯ NaN.

   √ `25.5`.

   ◯ `51.0`.

   ◯ 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 positive value.

   √ represents a negative value.

   ◯ represents a zero value.

   ◯ represents an infinity value.

   ◯ represents a NaN value.

   √ represents a denormalized value.

9. Interpret `0x97979797` and `0x17979797` as IEEE single-precision floating-point. Add the two values together. What is the result? [5 pts]

   √ `0x00000000.`

   ◯ `0x7FC00000.`

   ◯ `0x7F800000.`

   ◯ `0x17800000.`

   ◯ `0x97800000.`

   ◯ none of the above.

10. Interpret `0x03FFFFFF` as a 32-bit two's complement value. Convert it to IEEE single-precision floating point. What is the result? [5 pts]

   ◯ `0x4C7FFFFF.`

   √ `0x4C800000.`

   ◯ `0x4C000000.`

   ◯ `0x7F800000.`

   ◯ none of the above.

11. Represent this UTF-16 value, `0x0111`, in UTF-8. What is the result? [5 pts]

   ◯ `0xF0 0xC0 0xC4 0x91.`

   ◯ `0xE0 0xC4 0x91.`

   ◯ `0xD1 0x84`

   √ `0xC4 0x91.`

   ◯ `0x11.`

   ◯ none of the above.

12. Represent this UTF-32 value, `0x00000111`, in UTF-16. What is the result? [5 pts]

   √ `0x0111.`

   ◯ `0x8111.`

   ◯ `0xD801 0xDC11.`

   ◯ `0xD811 0xDC10.`

   ◯ none of the above.

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

   √ The UTF-8 encoding of an ASCII character is always one byte long.

   ◯ The UTF-8 encoding of a Unicode character is always 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.

   ◯ The UTF-16 encoding of a Unicode character is always shorter than the UTF-32 encoding of the character.

14. The Byte-Order Mark (BOM) is encoded as 0xFEFF. Which of the following statements are true? [5 pts]

   ◯ A file containing UTF-16 characters that starts with `0xFE 0xFF` is in little-endian format.

   √ A file containing UTF-16 characters that starts with `0xFF 0xFE` is in little-endian format.

   ◯ The endian-ness of a file of UTF-16 characters depends on the machine that is reading the file.

   ◯ The endian-ness of a file of UTF-16 characters depends on the machine that wrote the file.

15. What is the encoding of this RISC-V instruction: `add x0,x1,x2`. [5 pts]

   ◯ 000100B3.

   √ 00208033.

   ◯ 00100133.

   ◯ 00209033.

   ◯ none of the above.

16. Decode this RISC-V instruction: 0x00408093. What is the result? [5 pts]

   ◯ `add x1,x1,x4`.

   ◯ `slti x1,x0,4`.

   ◯ `addi x1,x0,8`.

   √ `addi x1,x1,4`.

   ◯ none of the above.

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

   √ Insymbols in an object file are symbols defined in the file and made available to the linker.

   √ Outsymbols in an object file are symbols referenced in the file but not defined in the file.

   √ The insymbol section of an object file contains each insymbol and its address.

   √ The outsymbol section of an object file contains each outsymbol and the address of every reference to the outsymbol.

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

   √ Absolute addresses in an object file never need to be relocated during linking.

   ◯ Relative addresses in an object file never need to be relocated during linking.

   √ PC-relative addresses in an object file never need to be relocated during linking.

19. A linker tries to match up an outsymbol in one file with an insymbol in another file. Which of the following statements are true? [5 pts]

   √ If a match is found, then the address of the insymbol is used to fill in the "holes" in the instructions that reference the outsymbol.

   ◯ If a match is found, then an error is reported.

   √ If a match is not found, then the outsymbol is placed in the outsymbol section of the file produced by the linker.

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

   ◯ An assembler has two passes because the use of a label may come after its definition.

   √ The first pass of an assembler determines the address of each label defined in the program.

   √ The second pass of an assembler encodes the instructions.

   √ The assembler uses a symbol table to store labels and their addresses.