CSCI-UA.0201-003

Computer Systems Organization

Final Exam Spring 2013

Last name: First name:

Notes:

- **If you perceive any ambiguity in any of the questions, state your assumptions clearly**
- **Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question.**
- **This exam is open book/notes but no electronic devices.**

---

1. (5 points) Circle the correct answer among the choices given. If you circle more than one answer, you will lose the grade of the corresponding question.

(A) Which of the following has a larger size (in terms of bytes)?

1. pointer to a character
2. pointer to an integer
3. pointer to a float
4. They are all of the same size.

(B) Which of the following data types is not affected by the byte ordering of the machine (i.e. big endian vs little endian)?

1. char 2. int 3. float 4.double

(C) Presenting +5 in signed integer or in IEEE 754 yields the same bit pattern.

1. The above statement is true
2. The above statement is false
3. It depends on whether the machine is 32-bit or 64-bit
4. It depends on whether the machine is big-endian or little endian.

(D) To uniquely identify a connection we need:

1. A socket and a port number
2. Two sockets
3. An IP address and a port number
4. Two IP addresses

(E) Semaphores are needed the most when we have:

1. Several threads not sharing any variables
2. Several threads sharing a lot of variables
3. Several threads sharing read only variables
4. Only single thread and no concurrency

2. (5 points) Consider the following assembly representation of a function `foo` containing a `for` loop:

```
foo:
      pushl %ebp
      movl %esp,%ebp
      pushl %ebx
      movl 8(%ebp),%ebx
      leal 2(%ebx),%edx
      xorl %ecx,%ecx
      cmpl %ebx,%ecx
      jge .L4
.L6:
      leal 5(%ecx,%edx),%edx
      leal 3(%ecx),%eax
      imull %eax,%edx
      incl %ecx
      cmpl %ebx,%ecx
      jl .L6
.L4:
      movl %edx,%eax
      popl %ebx
      movl %ebp,%esp
      popl %ebp
      ret
```

Fill in the blanks to provide the functionality of the loop:

```
int foo(int a){
    int i;
    int result = _______________;
    for( _________; _________; i++ )
    { _____________________;
      _____________________;
     }
    return result;
}
```

3. (8 points) For the following two functions (assume n >= 0):

```
int fact(int n)
{
  int i;
  int result = 1;

  for (i = n; i > 0; i--)
    result = result * i;

  return result;
}
```

```
int fact_u2(int n)
{
  int i;
  int result = 1;

  for (i = n; i > 0; i-=2) {
    result = (result * i) * (i-1);
  }

  return result;
}
```

(a) What does the function **`fact`** do?

(b) For which values of n do **`fact`** and **`fact_u2`** produce similar results?

(c) What is the minimum modification that we can do to **`fact_u2`** such that it always produces same result as **`fact`**?

4. (4 points) For the following C code, indicate how many times "Hello" is printed.

```
void doit() {
  fork();
  fork();
  printf("hello\n");
  return;
}

int main() {
  doit();
  printf("hello\n");
  exit(0);
}
```

Answer: _____ output lines.

5. (8 points) The left part of the following figure shows a heap with the addresses of each word and the content.

- The addresses grow from bottom to top
- The memory allocator uses an implicit list
- Each memory block has a head of 32 bits and a footer of 32 bits
- Bits 0 (right most), 1, and 2 of the header (and footer) are as follows:
    - Bit 0: 1 means block is allocated, 0 means free
    - Bit 1: 1 means previous block is allocated, 0 means free
    - Bit 2: always 0
- The remaining 29 bits indicate the total size of the block, which is multiple of 8. This means you have to add three zeros to right to get the size.

What you need to do is to complete the right size of the Figure which shows the heap just after executing **free(0x400b010).** Write the content in hexadecimal.

Address	
0x400b028	0x00000012
0x400b024	0x400b611c
0x400b020	0x400b512c
0x400b01c	0x00000012
0x400b018	0x00000013
0x400b014	0x400b511c
0x400b010	0x400b601c
0x400b00c	0x00000013
0x400b008	0x00000013
0x400b004	0x400b601c
0x400b000	0x400b511c
0x400affc	0x00000013

Address	
0x400b028	
0x400b024	0x400b611c
0x400b020	0x400b512c
0x400b01c	
0x400b018	
0x400b014	0x400b511c
0x400b010	0x400b601c
0x400b00c	
0x400b008	
0x400b004	0x400b601c
0x400b000	0x400b511c
0x400affc	