CIS 2107
Computer Systems and Low-Level Programming
Spring 2010
Final

May 6, 2010

Name: _____

Page	Points	Score
1	15	
2	13	
3	6	
4	5	
5	17	
6	7	
7	9	
8	12	
9	4	
10	15	
12	10	
13	7	
14	10	
15	10	
Total:	140	

Instructions

The exam is closed book, closed notes. You may *not* use a calculator, cell phone, etc.

If the question reads, "answer briefly", it means just that. A sentence or two at most should be sufficient.

Unless otherwise specified, you *may* use functions from the Standard C Library.

There is some extra room on the back page.

Good luck.

(1 point) 1. What could I type in the shell to change from my current directory to my home directory?

1. _____

(1 point) 2. How would I run the program called `barf` and see its output one screen at a time?

_____

3. **The compilation process**. During the semester, at times, we've gone through each of the steps of the compilation process separately.

   We did:

   - `gcc -E flurg.c` to get `flurg.i`.
   - `gcc -S flurg.i` to get `flurg.s`.
   - `gcc -c flurg.s` to get `flurg.o`.
   - `gcc -o flurg flurg.o` to get the executable file `flurg`.

(2 points) (a) Describe what's in `flurg.i`. What do we call the process of translating `flurg.c` into `flurg.i`?

_____

_____

(2 points) (b) Describe what's in `flurg.s`. What do we call the process of translating `flurg.i` into `flurg.s`?

_____

_____

(2 points) (c) Describe what's in `flurg.o`. What do we call the process of translating `flurg.s` into `flurg.o`?

_____

_____

(2 points) (d) Describe what's in `flurg`. What do we call the process of translating `flurg.o` and any necessary libraries into `flurg`?

_____

_____

(3 points) 4. We've described the storage hierarchy in modern computers as a type of pyramid. What three things are generally true the higher up the pyramid we go?

_____

_____

(2 points) 5. Very briefly, when we declare a struct inside a .h file in C, why is it that we enclose it inside `#ifndef, #define, ..., #endif`?

_____

_____

_____

_____

(3 points) 6. Other than the problem of differing libraries, explain why it is that I can't take a Windows executable, and run it on an Intel Mac or a machine running Linux on Intel. Please do not write, "because they're different operating systems". Be more specific about what the major problems are.

(2 points) 7. We've said that a CPU will in an endless loop:

- fetch the next instruction from memory from the address in the program counter register (AKA instruction pointer)
- decode the instruction, *i.e.* figure out what the instruction is and what its operands are.
- execute the instruction, and update the program counter so that it points to address of the next instruction to execute.

Most of the time, the next instruction to execute is just a few bytes later than the current instruction. When is this not the case?

(2 points) 8. How are C function return values typically implemented in x86 assembly?

9. some conversions

(1 point) (a) 112 tbits = ? kbytes

(a) _____

(1 point) (b) 104 gbytes = ? mbytes

(b) _____

(1 point) (c) 72 bits = ? gbytes

(c) _____

10. Using the approximation trick that we talked about in class, about how much are each of the following?

(1 point) (a) 2^{26}

(a) _____

(1 point) (b) 2^{37}

(b) _____

(1 point) (c) 2^{49}

(c) _____

(2 points) 11. Convert 247_{10} to base 2 and base 16.

(2 points) 12. Calculate in base 2.

$$\begin{array}{cccccccc} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1_2 \\ & + & 1 & 0 & 1 & 1 & 1 & 1_2 \\ \hline \end{array}$$

(2 points) 13. Calculate in base 16.

$$\begin{array}{ccccc} 9 & C & 3 & A & 7_{16} \\ + & 9 & B & C & D_{16} \\ \hline \end{array}$$

14. **Some bit operations.** If we have `char i = 0x8A, j = 0xF4;`, what is the result of the following operations? Your answer must be in the form of exactly two hex digits[1].

(1 point) (a) !i

(a) _____

(1 point) (b) ~(i|j)

(b) _____

(1 point) (c) i^(j & j)

(c) _____

(1 point) (d) i||j

(d) _____

(1 point) (e) i<<3

(e) _____

[1]Yes. In real life, some of these operations could involve promoting the operands to 32-bit ints. Ignore that for now. Just pretend that we're living in the land of 8-bit arithmetic.

(12 points) 15. Redacted question about two's-complement arithmetic

(5 points) 16. How would the number -59.4375_{10} be stored in a C `float` variable?

(4 points) 17. For each item in the following piece of code, indicate in which memory segment the item will be stored:

```
1  int A[1000];
2
3  char *func(void)
4  {
5     char *p;
6     int x=100;
7     p=malloc(x);
8
9     return p;
10 }
```

18. **Some tricky declarations**. Write a very brief description in English of what is declared. For example, if the question is `int func(int A[])`, you'd write, "func is a function which is passed an array of int and returns an int".

(1 point) (a) float *fp(float);

(1 point) (b) int *(*x[10])(void);

(1 point) (c) void (*fp)(int);

19. What is the value of each of the following after `func( )` is called?

```
1  #include <string.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #define BUFLEN 25
6
7  typedef struct {
8    int x;
9    char *s;
10   int A[3];
11 } Ektorp;
12
13 void func(int, int*, Ektorp, char[]);
14
15 int main(int argc, char **argv)
16 {
17   int x=10, y=20;
18   Ektorp e;
```

points: _____
out of a possible 7

question 19 continues...

```
19    int *ip=&y;
20    char s[BUFLEN];
21    e.x=30;
22    e.A[0]=40;
23    e.s = malloc(BUFLEN);
24
25    strcpy(s, "leksvik");
26    strcpy(e.s, "fargglad");
27
28    func(x, ip, e, s);
29    return 0;
30  }
```

```
31
32  void func(int x, int *ip, Ektorp e, char s[])
33  {
34    x=1111;
35    *ip=2222;
36    ip=&(e.A[0]);
37    e.x=3333;
38    e.A[0]=4444;
39    strcpy(e.s, "nerdflerg");
40    strcpy(s, "stenstorp");
41  }
```

(1 point) (a) x

(a) _____

(1 point) (b) *ip

(b) _____

(1 point) (c) e.x

(c) _____

(1 point) (d) e.s

(d) _____

(1 point) (e) e.A[0]

(e) _____

(1 point) (f) s

(f) _____

(3 points) 20. What's the problem with this code?

```
1   #define NROWS 5;
2   #define NCOLS 10;
3
4   int main(void)
5   {
6     int i;
7     int A[NROWS][NCOLS];
8     int *p = (int*)A;
9
10    for (i=0, p=(int*)A; i<NROWS*NCOLS; p++, i++)
11      *p=0;
12
13    return 0;
14  }
```

21. **pointer arithmetic**. What is the value of each of the following assuming that the array `A` begins at address 1000 and `int`s are 4 bytes?

```
1  typedef struct {
2     int x;
3     int y;
4  } point;
5
6  int A[100];
7
8  int *ip=&A[0];
9  char *cp=&A[0];
10 point *pp=&A[0];
11
12 ip = pp+4;
13 cp+=3;
14 pp+=10;
```

(3 points) (a) ip

(a) _____

(3 points) (b) cp

(b) _____

(3 points) (c) pp

(c) _____

22. You're given `char buff[1024]`. Write the line or two of code which does the following:

(1 point) (a) writes the float value `-56.824` in the memory location 100 bytes after the beginning of `buff`. You should assume that floats are 32-bit values.

(1 point) (b) writes the string `"sleepy"` just after the float. You may use functions in the C library.

(1 point) (c) writes a struct 10 bytes before the end of buff. The struct definition is:

```
1  typedef struct {
2     int x;
3     int y;
4  } point;
```

and `point`'s x coordinate is 10, and its y coordinate is 40.

23. Given the following C function, write an expression in assembly indicating where I'd find the following values:

```
1  int func(int x, int y, int z)
2  {
3     int i;
4     char c[20];
5
6     for (i=0; i<20; i++)
7        c[i]=(char)i*5;
8
9     return z++;
10 }
```

(1 point) (a) i

(a) _____

(1 point) (b) c[0];

(b) _____

(1 point) (c) the instruction to execute after func finishes

(c) _____

(1 point) (d) z

(d) _____

(5 points) 24. For each of the following, suppose that %eax contains the value x, %ecx contains y. What's stored in %edx after the each operation?

expression	**result**
leal 0xC(%eax), %edx	
leal (%eax,%ecx), %edx	
leal (%eax,%ecx, 4), %edx	
leal 5(%eax,%eax,8), %edx	
leal 0xA(,%ecx,8), %edx	

(5 points) 25. I'm implementing a function in assembly. What are the first and last lines that I need to write in the function in order to handle the base and stack pointers? Explain each of the lines.

(5 points) 26. Write a function which is passed a string `s`. The function removes the last character from `s`.

(10 (bonus)) 27. Write a C function equivalent to the following assembly (no credit for an answer containing inline assembly). Hint: the `negl` instruction reverses a number's sign.

```
1           .text
2   .globl funkytown
3           .type    funkytown, @function
4   funkytown:
5           pushl    %ebp
6           movl     %esp, %ebp
7           subl     $4, %esp
8           cmpl     $0, 8(%ebp)
9           jle      .L2
10          movl     8(%ebp), %eax
11          movl     %eax, -4(%ebp)
12          jmp      .L3
13  .L2:
14          movl     8(%ebp), %eax
15          movl     %eax, %edx
16          negl     %edx
17          movl     %edx, -4(%ebp)
18  .L3:
19          movl     -4(%ebp), %eax
20          leave
21          ret
```

(10 points) 28. A common way of storing a spreadsheet is comma-separated text. For example, the following line in a spreadsheet:

apple	banana	cherry	some fruit beginning with d

could be stored as "apple, banana, cherry, some fruit beginning with d". Write a function which is given a string of comma-separated values and gives back an array of string containing the values in the line. Using our current example, we'd return

s[0]	apple
s[1]	banana
s[2]	cherry
s[3]	some fruit beginning with d

You may decide the arguments to the function, the return value, and how it is that we communicate to the caller the length of the returned value `s`.

(7 points) 29. Given the following definition for an int linked list node:

```
1 struct node {
2   int data;
3   struct node *next;
4 };
```

write a function which is passed a pointer to a node `p`, which points to the head of a list, and an int `n`. The function returns the address of a node which contains the data `n`, or returns NULL if `n` isn't found in the list.

(10 points) 30. Write a function which is passed an 8-byte long `l` and returns a char `c`, which contains the low order bit of each of the bytes of `l`.

(10 points) 31. Write a program in which a series of one or more filenames will be passed at the command line. The program prints the character which appears in the files the most frequently. You may assume that the files contain ASCII text only.

(extra space)