CPSC 313, 06w Term 2— Midterm 1 — Solutions

Date: February 9, 2007; Instructor: Norm Hutchinson

1. **(8 marks)** Short answers.

1a. **(2 marks)** Is the address of a global variable in a C program determined *statically* (before the program runs) or *dynamically* (while the program is executing)? Briefly explain.

It is determined *statically*, by the linker in particular.

1b. **(2 marks)** Is the content of the jump table used in the execution of a switch statement determined *statically* or *dynamically*? Briefly explain.

It is *statically* determined, since the address of each piece of code is determined by the linker.

1c. **(2 marks)** Does the IA32 instruction-set architecture require that `%eax` be used to hold the value returned from a C function call? Briefly explain.

It does not require it - any register or even the stack could be used for this purpose. The use of `%eax` is a convention.

1d. **(2 marks)** If `%ecx` holds the address of an integer array `a`, and `%edx` holds the integer `i`, give a single assembly-language statement that computes `&a[i]` and places it in the register `%eax`.

```
    leal    (%ecx,%edx,4), %eax     # eax = &a[i]
```

2. **(8 marks)** Consider the following C source file.

```
int g;
void f (int t) {
    int a; int *b;
    /* consider each statement as if it were here */
}
```

Give an assembly-code implementation of each of the following statements of function `f()`. Consider each statement in isolation (i.e., as if it were the only statement of f). Do not assume that variables start out in registers. Be sure to write results to the appropriate location in memory. Assume that the local variables are in memory on the stack (not in a register). A fully correct answer will use as few instructions as possible. **Comment your code.**

2a. **(2 marks)** `b = &g;`

```
    movl    #g, -8(%ebp)     # b = &g
```

or

```
    leal    g, %eax          # eax = &g
    movl    %eax, -8(%ebp)   # b = &g
```

2b. **(2 marks)** `a = *b;`

```
    movl    -8(%ebp), %eax   # eax = b
    movl    (%eax), %eax     # eax = *b
    movl    %eax, -4(%ebp)   # a = *b
```

2c. **(2 marks)** `*b = 3;`

```
    movl   -8(%ebp), %eax  # eax = b
    movl   $3, (%eax)      # *b = 3
```

2d. **(2 marks)**

```
do {
  a = a - t;
} while (a > 0);
```

```
        movl    -4(%ebp), %eax
.L0:    subl    8(%ebp), %eax
        jg  .L0
        movl    %eax, -4(%ebp)
```

or

```
        movl    8(%ebp), %eax
.L0:    subl    %eax, -4(%ebp)
        jg  .L0
```

3. **(8 marks)** Assume that the following registers hold the indicated C language variables with the indicated types:

Register	Variable name	type
%ebx	a	int
%ecx	b	int
%edx	c	int *

For each of the indicated snippets of assembly code, write C-language statements that have the same effect. If your solution is significantly longer than mine, you will lose marks.

3a. **(2 marks)**

```
    leal    (%ebx, %ecx, 8), %eax
    ret
```

```
    return a + 8 * b;
```

3b. **(2 marks)**

```
    addl    (%edx), %ecx
```

```
    b += *c;
```

3c. **(2 marks)**

```
    movl    (%edx), %eax
    addl    (%edx), %eax
    movl    %eax, (%edx)
```

```
    *c += *c;
```

3d. **(2 marks)**

```
        cmpl    %ebx, %ecx
        jg  .L1
        movl    %ebx, %eax
        jmp .L0
.L1:    movl    %ecx, %eax
.L0:    ret
```

```
    return b > a ? b : a;
```

or

```
        if (b > a)
            return b;
        else
            return a;
```

4. **(6 marks)** Describe the input to and the work performed by each of the following three components of the process of converting a program in a high level language into an executable program.

4a. **(2 marks)** Compiler

The compiler takes as input a text file containing a source program in a high level language, and translates the program into an assembly language source program. This process involves checking for errors in the program as well as constructing an equivalent assembly language program.

4b. **(2 marks)** Assembler

The assembler takes as input a text file containing an assembly language program and translates it into machine code in a relocatable object file.

4c. **(2 marks)** Linker

The linker takes as input one or more object files containing pieces of a complete program and any libraries. It determines final locations for all the code and data in the program and resolves all references in order to make an executable program.

5. **(6 marks)**

5a. **(3 marks)** Describe the three major tasks accomplished by a function's prologue.

- Save `%ebp` on the stack and make it point to this stack frame
- Allocate space for local variables
- Save any callee-save registers

5b. **(3 marks)** Describe the three major tasks that must be accomplished by the code that makes a call to another function.

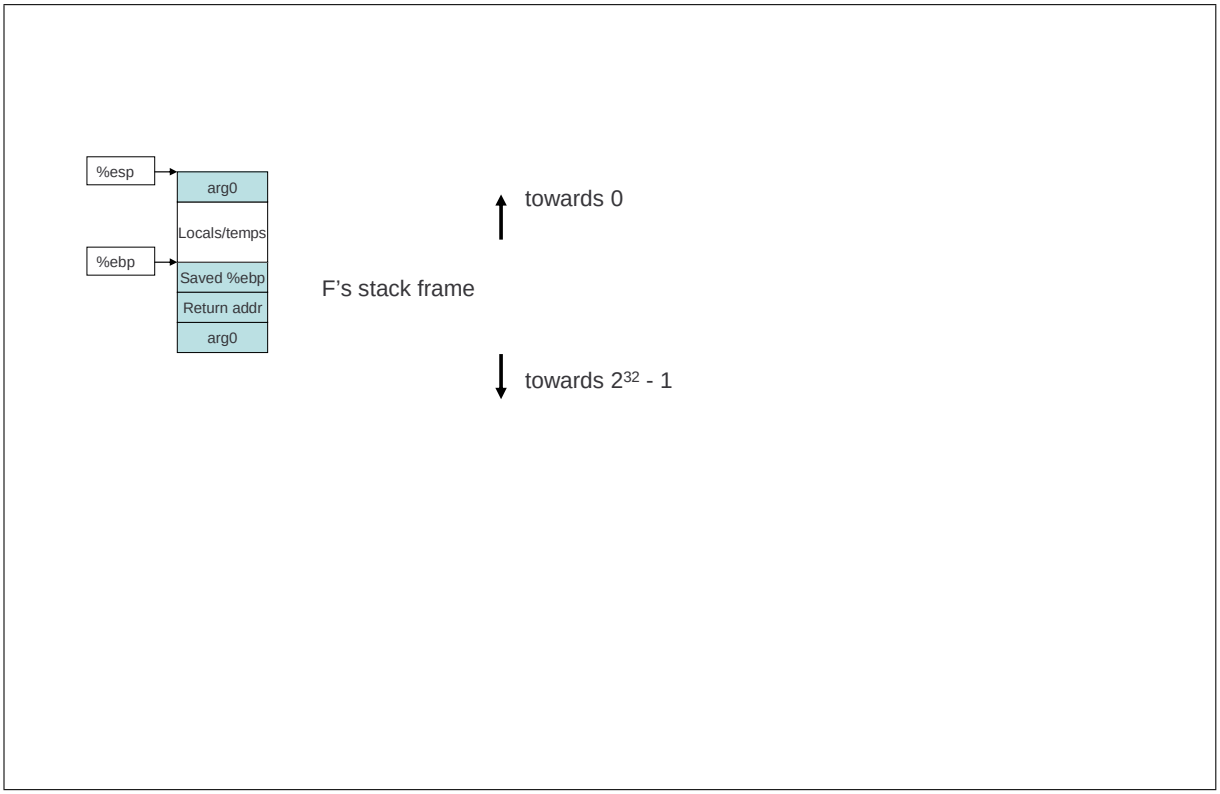
- Save any caller-save registers
- Place the arguments onto the stack (in the correct order)
- Execute the `call` instruction

6. **(6 marks)** A C program contains a function `F` with one integer parameter which calls a function `G` with one integer parameter.

6a. **(4 marks)** Draw a picture of the runtime stack immediately before the `call` instruction in function `F` executes. Your stack should contain all of the activation record for the function F including its one integer parameter.

In your picture, clearly indicate the location of the saved frame pointers, parameters, and return addresses. You should indicate the general location of local variables, but need not show them in detail.

Clearly indicate exactly where in the stack the registers `%esp` and `%ebp` point. Orient your stack so that lower addresses are at the top of the page.



6b. **(2 marks)** Give a single machine instruction that will load the address of the frame pointer for F's caller into the register `%eax`.

```
  leal  (%ebp), %eax
```

or

```
  movl  %ebp, %eax
```