

Practice Final

Prof. Tak Auyeung

December 12, 2004

Instructions: You may bring any material that is hand-written or printed *prior* to the examination to help you. You can also bring a calculator if you think it may help you. However, you can only use the calculator for numerical computations only.

You, as an individual, are expected to do your own work. This means you cannot seek, receive or otherwise acquire any assistance except clarifications from the professor during an examination. Any communication involving the contents of the subject matter or the examination is considered cheating. Do not initiate or accept such communication, or the result of your examination is automatically voided.

Each correct answer is worth one point, each wrong answer is worth zero point, and each unanswered question is also worth zero point. This means you *should* guess and leave no question unanswered.

Make sure you write down your name on the upper right corner *first*, otherwise I cannot give points to anonymous students!

- 1 The following is the C code representation of a record (structure):

```
struct X
{
    int i;
    int j;
};
```

Assume that an `int` is a 32-bit integer. How many bytes should be allocated for a variable of this type?

- A 32
- B 4
- C 8
- D 2
- E 64

- 2 Refer to question 1, we add a global variable definition as follows:

```
struct X *p;
```

In a program, we need to perform the following:

```
p->j = 0;
```

How do we implement this in assembly? Assume `offset_X_field` is the offset in number of bytes from the base of a `struct X` object.

Also, assume that `p` is declared as follows:

```
.data
p: .fill 1,4,0

A movl p,%eax
   movl $0,%eax

B movl p+offset_X_j,%eax
   movl $0,%eax

C movl p,%eax
   movl $0,offset_X_j(%eax)

D movl $0,offset_X_j(p)

E movl $0,offset_X_j+p
```

- 3 Refer to question 1. Now we add an array definition in C:

```
struct X a[10];
```

Assume that `sizeof_X` is a symbol defined to the number of bytes of a `struct X` object.

How do we implement the following statement in assembly:

```
a[3].j = 23;
```

- A `movl $23,a+offset_X_j[3]`
- B `movl $23,a+3*sizeof_X+offset_X_j`
- C `movl $3*sizeof_X,%eax`
`movl $23,offset_X_j(%eax)`
- D `movl a,%eax`
`movl $23,3*sizeof_X+offset_X_j(%eax)`
- E All of the above implement the C statement

- 4 Refer to question 1. Assume we allocate a local variable from the stack. Symbols are defined as follows:

```
oldEbp = 0
x = oldEbp - sizeof_X
```

The matching code to perform the allocation at run time is as follows:

```

pushl %ebp
movl %esp,%ebp
addl $x,%esp

```

Given these assumptions, how do we perform the following statement in assembly?

```
x.j = 12;
```

- A movl \$12,x
- B movl \$12,x+offset_X_j
- C movl \$12,x+offset_X_j(%ebp)
- D movl \$x,%eax
movl \$12,offset_X_j(%eax)
- E movl x,%eax
addl %ebp,%eax
movl \$12,offset_X_j(%eax)

5 Refer to question 1. Assume we allocate a global (static) variable. Symbols are defined as follows:

```

.data
x: .fill 1, sizeof_X, 0

```

Now, we need to call a subroutine expecting a single parameter that is of struct X type. How do we pass x as a parameter?

- A pushl x
- B pushl \$x
- C subl \$sizeof_X,%esp
movl %esp,%ebx
movl \$x,%eax
L0:
cmpl \$x+sizeof_X, %eax
jnb L1
movl (%eax),%ecx
movl %ecx,(%ebx)
addl \$4,%eax
addl \$4,%ebx
jmp L0
- D pushl \$x+sizeof_X
- E pushl x+sizeof_X

6 Refer to 1. ptr is a local variable pointer to a struct X. How do we implement the C code ++ptr so that ptr points to the next struct X object?

- A addl sizeof_X,ptr
- B addl \$sizeof_X,ptr
- C addl \$sizeof_X,ptr(%ebp)
- D movl ptr(%ebp),%eax
addl \$sizeof_X,(%eax)

E Cannot be determined because we don't know if ptr is pointing an object on the stack or not.

7 In a subroutine, we need to allocate an array of 10 32-bit integers as the only (auto) local variable. This is done by the following code:

```

oldEbp = 0
localArray = oldEbp - (10 * 4)
sub1:
pushl %ebp
movl %esp,%ebp
addl $localArray,%esp
...

```

Which C expression will corrupt the return address?

- A localArray[0] = 0;
- B localArray[10] = 0;
- C localArray[11] = 0;
- D None of the above because localArray has nothing to do with the stack.
- E None of the above because localArray has an address higher than that of the return address of sub1.

8 Describe the result of executing the following code.

```

pushl %eax
pushl %ebx
pushl %ecx
pushl %edx
movl $4,%eax
movl $1,%ebx
pushl $'\n'
movl %esp,%ecx
movl $1,%edx
int $0x80
addl $4,%esp
popl %edx
popl %ecx
popl %ebx
popl %eax

```

- A Nothing
- B It writes a newline character to the standard output file. There is no problem.
- C It reads a character from the standard input file. There is no problem.
- D It crashes.
- E It does something useful, and it does not crash immediately. But, the stack is left unbalanced.

9 Using our standard method to allocate parameters and local variables from the stack, how many bytes are required every time we call the following subroutine? Assume each int is a 32-bit integer.

```
void f(int x, int *y)
{
    int i;
    int j;

    // statements
}
```

- A 4
- B 8
- C 16
- D 24
- E Cannot be determined because we don't know the size of the array that y points to.

10 j and k are both local variables of int type. What assembly code passes j+k as a parameter to subroutine s1? In other words, how do we implement the following C code?

```
s1(j+k);
```

- A pushl j+k
- B pushl j+k(%ebp)
- C pushl j(%ebp)
movl k(%ebp),%eax
addl %eax,(%esp)
- D movl j(%ebp),%eax
addl k(%ebp),%eax
pushl %eax
- E 10c and 10d

11 Some architectures do not have the direct addressing mode. How else can we implement the following instruction?

```
movl someVar,%eax
```

- A movl \$someVar,%ebx
movl (%ebx),%eax
- B movl \$someVar,%ebx
- C movl someVar(%ebx),%eax
- D movl \$someVar,(%eax)
- E movl \$someVar,(%ebx)
movl (%ebx),%eax

12 Consider the following instruction. In which case will both the O flag and Z be set (don't worry about the other flags)?

```
cmpl %eax,%ebx
```

- A %eax = 0x80000000 and %ebx = 0x80000000
- B %eax = 0x80000000 and %ebx = 0x7fffffff

- C %eax = 0x7fffffff and %ebx = 0x80000000
- D None of the above, but there is such a case
- E None of the above, but such a case does not exist

13 The instruction ja often does not exist on most other architectures. How can we implement it using other instructions? For example, how do we implement ja L1 for all cases?

- A jb L1
- B jz L1
- C jg L1
- D jz L0
jnc L1
L0:
- E jnc L1
jnz L1

14 Refer to question 1. Assume the following:

- struct X *p1 is a parameter
- struct X *p2 is a parameter
- int num is an auto local variable

Assuming that in assembly, the name of each parameter or variable is the displacement from %ebp, how do we implement the following statement?

```
p1->i = p2->j + num;
```

- A movl p2+offset_X_j(%ebp),%eax
addl num(%ebp),%eax
movl %eax,p1+offset_X_i
- B movl p2+offset_X_j(%ebp),p1+offset_X_i(%ebp)
addl num(%ebp),p1+offset_X_i(%ebp)
- C movl p2(%ebp),%eax
movl offset_X_j(%eax),%ebx
addl num(%ebp),%ebx
movl p1(%ebp),%eax
movl %ebx,offset_X_i(%eax)
- D movl p2+offset_X_j,%eax
addl num(%ebp),%eax
movl %eax,p1+offset_X_i
- E 14a and 14b

15 If the S flag is set after the following instruction, what do we know about the registers?

```
cmpl %eax,%ebx
```

- A %eax is (unsigned) greater than %ebx
- B %eax is (signed) greater than %ebx
- C %ebx is (unsigned) greater than %eax
- D %ebx is (signed) greater than %eax

- E not one of the above can be confirmed
- 16 We suspect that a subroutine `bart` modifies register `eax`. In order to confirm this, we try to write some code so that we jump to label `yikes` if and only if register `eax` is modified by subroutine `bart`. Which of the following invocation code does this?
- We assume subroutine `bart` does not have any parameters, and it does not return any value. Also, the stack needs not be balanced when we get to `yikes`.
- A `pushl %eax`
`call bart`
`popl %eax`
`cmpl %eax,%eax`
`jnz yikes`
`addl $4,%esp`
- B `pushl %eax`
`call bart`
`cmpl %eax,%eax`
`jnz yikes`
`addl $4,%esp`
- C `pushl %eax`
`call bart`
`cmpl %eax,(%esp)`
`jnz yikes`
`addl $4,%esp`
- D `pushl %eax`
`call bart`
`cmpl %eax,4(%esp)`
`jnz yikes`
`addl $4,%esp`
- E all of the above code will do it
- 17 An operand of which of the following addressing modes always provides the same value?
- A immediate
 B register
 C direct
 D indirect
 E indexed
- 18 Which of the following is not a valid instruction?
- A `movl $1,%eax`
 B `movl %eax,(%ebx)`
 C `movl 1(%ebx),%eax`
 D `movl 1,(%eax)`
 E `movl 1,%eax`
- 19 Which conditional jump is not supported by the 386 architecture?
- A `ja`
 B `jb`
 C `jz`
 D `jo`
 E all of the above are supported
- 20 If the following instruction is repeated 33 times, what will be the value of register `%eax`? The answer has to work for all possible initial values of `%eax`.
- `addl %eax,%eax`
- A 0
 B the most positive value of a 32-bit signed number
 C the most positive value of a 32-bit unsigned number
 D the most negative value of a 32-bit signed number
 E 32
- 21 Assume that `%eax` has a value of 23 (in decimal) initially. What is the value of `%eax` after the following instructions?
- `pushl %eax`
`addl %eax,(%esp)`
`addl %eax,%eax`
`addl %eax,%eax`
`addl %eax,%eax`
`addl %eax,(%esp)`
`popl %eax`
- A 23
 B 230
 C 46
 D 184
 E 0
- 22 Instead of using the `ja` instruction, which other replacement instruction can be used in all cases for the following situation?
- `cmpl $0,%eax`
`ja L0`
- A `jb`
 B `jnb`
 C `jg`
 D `jnz`
 E None of the above can replace `ja`.
- 23 After the following instruction, the carry flag is set. What can we know for sure?
- `cmpl %ebx,%eax`
- A `%eax` is less than `%ebx` unsigned

- B `%ebx` is less than `%eax` signed
- C `%eax` is not less than `%ebx` unsigned
- D `%ebx` is no less than `%eax` signed
- E `%eax` is less than `%ebx` signed

24 Which of the following instructions does not access (read or write) memory (other than the instructions)?

- A `call sub1`
- B `ret`
- C `movl $0,0`
- D `popl %eax`
- E `addl $4,%esp`