

Practice Exam 2

Prof. Tak Auyeung

May 11, 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 *cannot* let your calculator compile a program, or to communicate with others.

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.

New rules, read this! As of 2003.09.22, I no longer deduct points for wrong answers. 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.

As a result, I also need to adjust the letter grade assignment break points. For your individual examination, “A” means at least 95%, “B” means at least 75%, “C” means at least 55%, “D” means at least 35% and “F” means below 35%. This exam. makes up 20% of your final grade.

Please note that this change does not affect your letter grade at all, it is just a number game to make some people feel better about guessing.

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

The baseline is 10, there are 12 questions.

1 Assume that `int` is 32-bit. Observe the following C code:

```
int f(int x)
{
    int i;

    // ...
    i = 100;
    // ...
}
```

The frame is set up as usual, and assume variable names are defined to be the displacement from the frame pointer `ebp` to the local variable or parameter.

f:

```
pushl %ebp
movl %esp,%ebp
addl $i,%esp

# ...
?????
# ...

movl %ebp,%esp
popl %ebp
ret
```

Which of the following initializes local variable `i` to 100 (and replaces `???`)?

- A `movl $100,i(%ebp)`
- B `movl 100,i(ebp)`
- C `movl $100,-i(%esp)`
- D `movl 100,i(%esp)`
- E `movl $100,i(%esp)`

2 The following is a C subroutine:

```
void f(void)
{
    char x[50];

    // ...
    write(1, x, 5); // write first five char of x to std
    // ...
}
```

When we implement this in assembly and use system service 4 to write the characters to the standard output file, how should we initialize register `%ecx`? Assume the name of a local variable defines the displacement from the frame pointer to its base address. Not all of the choices assemble.

- A `movl $x, %ecx`
- B `movl $x(%ebp), %ecx`
- C `movl %ebp,%ecx`
`addl $x, %ecx`
- D `movl %ebp, %ecx`
- E `movl x(%ebp), %ecx`

3 Assume an assembly subroutine implements the following C function:

```
void f(void)
{
    char ch;
    char *ptr;

    // ...
}
```

Assume the name of a local variable defines the displacement from the frame pointer to the base of the variable. Choose a C statement implemented by the following assembly code (some choice(s) may not compile):

```
movl %ebp, %eax
addl $ch, %eax
movl %eax, ptr(%ebp)
```

- A *ptr = ch;
- B ptr = ch;
- C ptr = &ch;
- D ch = *ptr;
- E ch = ptr;

4 Assume an assembly subroutine implements the following C code:

```
void g(int x);

void f(void)
{
    int x;

    // ...
    g(x);
    // ...
}
```

Assume the name of a local variable or parameter defines the displacement from the frame pointer to the base of the variable. Also assume that each file only defines one subroutine (so names of local variables or parameters are not shared). Which of the following correctly calls the subroutine `g` from `f` in assembly? Some of the choices may not assemble.

- A `pushl x`
`call g`
`subl $4,%esp`
- B `pushl x(%esp)`
`call g`
`subl $4,%esp`

- C `pushl x(%esp)`
`call g`
`addl $4,%esp`
- D `pushl x(%ebp)`
`call g`
`addl $4,%esp`
- E `pushl $x+%ebp`
`call g`
`addl $4,%esp`

5 Read the following C subroutine:

```
void f(void)
{
    struct {
        i : int;
        j : int;
        k : char;
    } x;

    x.i = 20;
    x.j = x.i + 3;
    x.k = 'b';
    // ...
}
```

When we implement this in assembly, we can allocate and initialize at the same time. Assume field `i` has the lowest address and `k` has the highest address. Also, assume two bytes are allocated for a `char`, but only the least significant 8 bits are used.

Furthermore, assume the name of a local variable defines the displacement from the frame pointer to its base address. Choose a sequence of instructions to replace `---` so we allocate and initialize local variable `x`. Your selection only needs to be functionally the same as the C code in terms of the end result. Pushing constants is allowed.

```
f:
pushl %ebp
movl %esp,%ebp
---
# ...
movl %ebp,%esp
popl %ebp
ret
```

- A `movl $20,%eax`
`pushl %eax`
`addl $3,%eax`
`pushl %eax`
`pushw $'b'`
- B `pushl $20`
`pushl +3`
`pushw $'b'`

- C `pushw $'b'`
`pushl $23`
`pushl $20`
- D `pushw $'b'`
`pushl $20`
`pushl $23`
- E None of the above has the same effect as the C code.

6 Consider the following C code:

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

    // ...
}
```

Choose a C statement that is implemented by the following assembly code. Assume the name of a local variable is defined to the displacement from the frame pointer to its base address.

```
f:
pushl i(%ebp)
movl i(%ebp),%eax
addl j(%ebp),%eax
addl %eax,%eax
addl %eax,(%esp)
call g
addl $4,%esp
```

- A `g(i);`
- B `g(i+j);`
- C `g(i+i+j);`
- D `g(i+2*(i+j));`
- E `g(i+(i*=2)+j);`

7 A C compiler can enforce the number and type of parameters. However, this is not possible in assembly. Assume the following prototype:

```
void f(int x);
```

What happens when the following assembly code is used to call `f`? Assume the frame is set up the usual way as discussed in the class, in which the first parameter is pushed last.

```
pushl $45
pushl $59
call f
addl $8,%esp
```

- A `f` crashes because it accesses invalid locations, regardless of the definition of `f`.
- B `f` does not crash, but the caller does, regardless of the rest of the code of the caller.
- C this is the same as `f(45)`
- D this is the same as `f(59)`
- E somewhere, this program as a whole will crash, regardless of the code.

8 In C (any many other programming languages), a two dimensional array is like an array of arrays. In other words, `char x[23][17];` means `x` is an array of 23 (array of 17 `char`). Parentheses were used in the previous sentence to indicate order. Alternatively, you can also see `char x[23][17];` the same as `char (x[23])[17];` if this helps.

What is the displacement of the memory location of `x[3][4]` from the base address of `x`? Remember that all arrays are zero-oriented in C (first element has an index of 0). Each `char` requires one byte, and all elements of `x` are allocated contiguously.

- A 3×4
- B $4 \times 23 + 4$
- C $3 \times 23 + 4 \times 17$
- D $3 \times 17 + 4$
- E $4 \times 17 + 5$

9 Using the usual method to construct a frame, what is the *minimum* number of bytes required on the stack for *each* invocation of the following function? In other words, what is the size of each frame of the following function? Note that a frame includes parameters (if any) and auto local variables (if any).

```
void f(int n)
{
    int i;
    char x[8];

    // ...
}
```

- A -12
- B 0
- C 4
- D 16
- E 24

10 Observe the following C definition:

```
struct Node {
    struct Yada y;
    struct Node *next;
};
```

```

};

void g(struct Yada *y);

void f(struct Node *n)
{
    // ...
    g(&(n->y));
    // ...
}

```

Let `Yada_size` define the size of each `struct Yada` object. `Node_y` defines the displacement of field `y` from the base of a `struct Node` object. `Node_next` defines the displacement of field `next` from the base of a `struct Node` object.

Assume the name of a local variable or parameter defines the displacement of the variable or parameter from the frame pointer. Also, assume each subroutine is defined in a separate file, and there is no mixing of names of local definitions.

How do we implement `g(&(n->y))`?

- A `movl n(%ebp),%eax`
 `addl $Node_y, %eax`
 `pushl %eax`
 `call g`
 `addl $4,%esp`
- B `movl n+Node_y(%ebp),%eax`
 `pushl %eax`
 `call g`
 `addl $4,%esp`
- C `pushl n(%ebp)`
 `addl $Node_y, (%esp)`
 `call g`
 `addl $4,%esp`
- D 10a and 10c
- E All of the above