

Exam 2

Prof. Tak Auyeung

November 17, 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!

The baseline is 10, there are 12 questions.

1 The following code calls a subroutine:

```
.data
a: .long 0x3, 0x6, 0x31, 0x0, 0x5, 0x7, 0x3, 0x6
   n = (. - a) / 4
.text
.global _start
_start:
    movl %esp,%ebx
    movl $a, %eax
    pushl $0
L1:
    cmpl $0,(%eax)
    jz   L2
    pushl %eax
    addl $4,%eax
    jmp  L1
L2:
    call sub1
    movl %ebx,%esp
```

When subroutine `sub1` gets control, how many bytes are pushed on the stack by the caller, not counting the return address?

- A 4
- B 12
- C 16
- D 24
- E 32

2 Refer to question 1. Besides the obvious long word zero pushed on the stack, what else is pushed (not counting the return address)?

- A nothing else is pushed on the stack
- B the values of some items defined in the `.data` section
- C the addresses of some items defined in the `.data` section
- D the values and addresses of some items defined in the `.data` section
- E something other than the values or addresses of items defined in the `.data` section

3 Refer to question 1. The following is the code of the subroutine `sub1`. Assuming `sub1` is called from the code in question 1, what is the value of `eax` right before the `ret` instruction?

```
sub1:
    pushl   %ebp
    movl    %esp,%ebp
    pushl   %ebx
    pushl   %ecx

    movl    %ebp,%ebx
    addl    $8,%ebx
    movl    $0,%eax
```

```
L3:
    cmpl    $0,(%ebx)
    jz      L4
    movl    (%ebx),%ecx
    addl    (%ecx),%eax
    addl    $4,%ebx
    jmp     L3
```

```
L4:
    popl    %ecx
    popl    %ebx
    popl    %ebp
    ret
```

- A 0
- B $0x5+0x7+0x3+0x6$
- C $4*a+4+4+4$ (definition of `a` in 1)
- D $0x3+0x6+0x31$
- E $0x3+0x6+0x31+0x0+0x5=0x7+0x3+0x6$

4 Refer to question 3. *In general*, what is the number of parameters (on the stack) expected by subroutine `sub1`?

- A 0
- B at least 1 long word
- C 4 long words
- D 3 long words
- E 5 long words

5 Read the following program, and choose one of the descriptions.

```
.text
.global _start
_start:
    pushl   $L1
    ret
```

```
L1:
    nop

    movl    $1,%eax
    movl    $0,%ebx
    int     $0x80
```

- A assemble time error: `ret` must be in a subroutine
- B run time error at the `ret` instruction because there is no matching `call` instruction
- C no error, but the stack is not balanced at the `nop` instruction
- D there is no problem
- E run time error: the program crashes after `int $0x80`

6 Which of the following instructions write(s) to memory? Assume all labels are properly defined.

- A `pushl $0`
- B `popl %eax`
- C `call sub1`
- D `ret`
- E 6a and 6c

7 Select one of the options to describe the behavior of this program.

```
.data
str: .asciz "abcde"
.text
.global _start
_start:
    movl $str,%eax

    call sub1

    movl $1,%eax
    movl $0,%ebx
    int $0x80

sub1:
    cmpb $0,(%eax)
    jz L1
    pushl %eax
    movl %eax,%ecx
    movl $4,%eax
    movl $1,%ebx
    movl $1,%edx
    int $0x80
    popl %eax
    pushl %eax
    addl $1,%eax

    call sub1

    popl %ecx
    movl $4,%eax
    movl $1,%ebx
    movl $1,%edx
    int $0x80
L1:
    ret
```

- A prints abcdedcba
- B prints abcdeedcba
- C prints abcde
- D prints eeddcbbaa
- E prints aabbccdde

8 Refer to question 7. Describe what happens when we swap the order of the instructions (right before call sub1 in the subroutine) from:

```
pushl %eax
addl $1,%eax

to

addl $1,%eax
pushl %eax
```

- A sub1 keeps calling itself until there is no room left on the stack, the program then crashes
- B the program does not crash, but it does not exit anymore
- C the program still exits without any problem, but what it prints is different from the output of question 7
- D the program has the same behavior as in question 7
- E the behavior of the program does not match any of the above descriptions

9 Refer to question 7. What is the *maximum* number of bytes that is used on the stack when the program executes? In other words, what is the minimum number of bytes available on the stack so the program executes without any problem? Do not include the stack usage of the int instruction.

- A 40
- B 44
- C 48
- D 60
- E 64

10 What are the values of registers `eax` and `ebx` immediately after the following instructions? Assume the stack is set up correctly *before* these four instructions, and it has at least eight bytes remaining.

```
movl $12,%eax
movl $678,%ebx
pushl %eax
popl %ebx
pushl %ebx
popl %eax
```

- A unknown, it depends on what was the on the stack before these instructions
- B `eax` has 12, and `ebx` has 678
- C `eax` has 12, and `ebx` has 12
- D `eax` has 678 and `ebx` has 12
- E `eax` has 678 and `ebx` has 678

11 We are given the following code (incomplete):

```
.data
num1: .long 0x1011fe73
num2: .long 0x25039237
.text
.global _start
_start:
    pushl  num1
    pushl  num2
    call   sub1
    addl   $8,%esp
```

In the subroutine `sub1`, how do we compute the sum of `0x1011fe73` and `0x25039237` and store the sum in register `eax`? Assume all registers are at your disposal, and the beginning of `sub1` is as follows:

```
sub1:
    pushl %ebp
    movl  %esp,%ebp

A    movl 8(%ebp),%eax
    addl 12(%ebp),%eax

B    movl 8(%ebp),%ebx
    movl (%ebx),%eax
    movl 12(%ebp),%ebx
    addl (%ebx),%eax

C    movl 12(%ebp),%eax
    addl 8(%ebp),%eax

D    11a and 11b

E    11a and 11c
```

12 When a multi-byte operand is pushed, it follows the little-endian convention (least significant byte at the lowest address) convention. The “pop” operation also follows this convention, the byte at the lowest address becomes the least significant byte. What is the 32-bit value of the 32-bit register `eax` after the following instructions?

```
pushw $0x1234
pushw $0x5678
popl  %eax
```

- A 0x1234
- B 0x12345678
- C 0x5678
- D 0x56781234
- E 0x34127856