

Practice Exam 2

Prof. Tak Auyeung

November 8, 2004

Instructions: You may bring any material that is handwritten 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 Read the following program:

```
.lcomm ch 1
.global _start
.text
_start:
    movl $1,%edx
    movl $ch,%ecx
    pushl %edx
    pushl %ecx
    movl $0,%ebx
    movl $3,%eax
    int $0x80
    cmpl $0,%eax
    jz done
    cmpb $'\n',ch
    jz _start
    popl %ecx
    popl %edx
    movl $1,%ebx
    movl $4,%eax
    int $0x80
    jmp _start
done:
    movl $1,%eax
    movl $0,%ebx
    int $0x80
```

What is the output of this program, assuming the input file looks like the following? (Assume the input file is redirected to the program.) There are no empty lines, and no trailing spaces on each line.

5
6
7

- A 12 34
- B 1234567
- C 12 34 5 6 7
- D 12
- E 12 34567

2 What is the 16-bit value in `ax` at the `nop` instruction?

```
.global _start
.text
_start:
    movw $46,%ax
    pushw %ax
    pushw %ax
    call sub1
    popw %ax
    popw %ax
    nop
    movl $1,%eax
    movl $0,%ebx
    int $0x80

sub1:
    movw 4(%esp),%ax
    addw 6(%esp),%ax
    ret
```

- A this code does not assemble
- B the execute never gets to `nop`
- C 46
- D 92
- E we get to `nop`, but the value of `ax` cannot be determined

3 Sometimes, we just need to remove an item from the stack without restoring it to a register or memory location. Assuming that we can alter flag bits in the status register (carry, zero, overflow, etc.), which of the following instruction sequences releases 8 bytes from the stack?

- A `subl $8, (%esp)`
- B `subl 8, %esp`
- C `addl $8, %esp`
- D `popa`
- E `popf`

4 What happens when the following subroutine returns from a `call`? Assume the code in ... is correct and does not cause any problem by itself.

```
sub1:
    pushl %eax
    # ...
    popl %ebx
    popl %eax
    ret
```

- A this code does not assemble
- B this code *always* returns to the instruction immediately following the invoking `call` instruction
- C this code *may not* return to the instruction immediately following the invoking `call` instruction, depending on the values on the stack that is not controlled by the subroutine
- D this code *always* crashes (segmentation fault) at the `popl %ebx` instruction
- E this code *always* crashes (segmentation fault) at the `popl %eax` instruction

5 Assuming the stack is infinite, what happens when the following code is called?

```
sub1:
    cmpb $0,%al
    jz   done
    subb $1,%al
    call sub1
done:
    ret
```

- A this code does not assemble, regardless of the rest of the program
- B this code never returns, it keeps going and going and ...
- C this code always crashes (segmentation fault)
- D this code always returns with a zero in `al`
- E the behavior of this code cannot be determined

6 What is the value of `ax` at the `nop` instruction?

```
.global _start
.text
_start:
    movw $0,%ax
    call sub1
    call sub2
    call sub3
    nop

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

sub1: addw $1,%ax
sub2: addw $2,%ax
sub3: addw $3,%ax
    ret
```

- A 1
- B 2
- C 3
- D 6
- E 14

7 What happens when the following code executes?

```
pushl %eax
popl  %ebx
```

- A swaps the values of `eax` and `ebx`
- B copies the value of `eax` to `ebx`
- C copies the value of `ebx` to `eax`
- D this code does not assemble because the registers do not match
- E this code always crashes (segmentation fault)

8 Ignoring the side effect to the status flags, what does the following code do? Assume the capacity of the stack is infinite

```
subl $4,%esp
movl $523245,(%esp)
```

- A `pushl 523245`
- B pushes the constant 523245 on the stack
- C this code does not assemble
- D this code results in a segmentation fault
- E changes the stack pointer to point to location 523245

9 What is the value of `ax` after the following code executes?

```
movw $23, %ax
movw $67, %bx
movw $12, %dx
mulw %bx
```

- A the least significant 16 bits of 23×67
- B the least significant 16 bits of $(12 \times 256 + 23) \times 67$
- C the most significant 16 bits of 23×67
- D the most significant 16 bits of $(12 \times 256 + 23) \times 67$
- E the value of `ax` cannot be determined just by the code shown

10 It is a hassle to set up for a system call to read from the standard input file (`stdin`). The following subroutine is used to read from the standard input:

```
getStdin:
    movl $3,%eax
    movl $0,%ebx
    movl $1,%ecx
    pushl %ebx
    movl %esp,%ecx
    movl $1,%edx
    int $0x80
    popl %ebx
    ret
```

Describe the result after the following code executes:

```
call getStdin
```

- A the code crashes (segmentation fault) when it executes `ret`

- B the code crashes (segmentation fault) when it executes `int $0x80`
- C it reads one character, returned in `bl`
- D it reads one character, but it is not returned
- E it reads one character, returned in `eax`

11 Which of the following instructions does not alter the stack pointer `esp`?

- A `push %ebp`
- B `ret`
- C `call sub1`
- D `subl $4,%esp`
- E `movl var1,(%esp)`

12 What is the maximum number of bytes to be used in the following program? Each return address consumes four bytes. Do not count the bytes that may potentially be used by interrupts. Do not count the bytes that are used by the `int` instruction.

```
.text
.global _start
    call sub1
    call sub2
    call sub3
    movl $1,%eax
    movl $0,%ebx
    int $0x80
```

```
sub1:
    call sub2
    ret
```

```
sub2:
    call sub3
    ret
```

```
sub3:
    ret
```

- A 4
- B 8
- C 12
- D 16
- E 20