

Exam 1

Prof. Tak Auyeung

October 13, 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 Which of the following instructions potentially changes the value of a general purpose register? Assume labels are defined so that all lines assemble.

- A `cmpl $0xff, (%eax)`
- B `cmpl $0xff, %eax`
- C `cmpl %eax, 6324523`
- D `cmpb (%ebx), %eax`
- E None of the choices modify a general purpose register

2 Observe the following code:

```
movb $0x80,%al
subb $____,%al
```

Select a value for `____` such that the sign (S) flag is set after the `subb` instruction executes.

- A `0x00`
- B `0x10`
- C `0x7f`
- D `0x80`
- E `0xff`

3 The following are byte values at location `somevar`:

```
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08
```

Assume that `%eax` has the address of `somevar`. What is the content of `somevar` after the following code executes?

```
movl (%eax),%ebx
addl $1,%eax
movb %bl,(%eax)
```

- A `0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08`
- B `0x01 0x01 0x03 0x04 0x05 0x06 0x07 0x08`
- C `0x01 0x01 0x02 0x03 0x04 0x05 0x06 0x07`
- D `0x01 0x01 0x02 0x03 0x04 0x06 0x07 0x08`
- E `0x01 0x02 0x03 0x04 0x01 0x02 0x03 0x04`

4 Which of the following code compares two 64-bit number, and branches to `label1` if and only if `val1` equals `val2`? Assume `val1` and `val2` are signed, and they are properly allocated space. Select an answer that works in general.

- A `movl val1,%eax`
`cmpl val2,%eax`
`jz label1`
`movl val1+4,%eax`
`cmpl val2+4,%eax`
`jz label1`
- B `movl val1+4,%eax`
`cmpl val2+4,%eax`
`jnz label2`
`movl val1,%eax`
`cmpl val2,%eax`
`jz label1`
`label2:`
- C `movl val1,%eax`
`cmpl val2,%eax`
`jnz label2`
`movl val1+4,%eax`
`cmpl val2+4,%eax`
`jz label1`
`label2:`
- D 4a and 4b
- E 4b and 4c

5 Assume that `var1` and `var1_end` are defined as follows:

```
.data
var1: .org +20
      var1_end = .
```

The direction `.org +20` means increase the memory location counter by 20 (bytes). `var1_end` is effectively defined to be the address of the byte immediately following the 20 bytes allocated by the first line.

What does the following code do to the allocated memory locations?

```
      movl $var1,%eax
L1:   cmpl $var1_end,%eax
      jnb L2
      movl $0,(%eax)
      addl $4,%eax
      jmp L1
L2:
```

- A starting with the byte at offset 0, every fourth byte is initialized to zero with no change to all the other bytes.
- B this loop has a run-time error because it accesses locations that is not a part of the 20 allocated bytes
- C all 20 allocated bytes are initialized to 0
- D this is an infinite loop, `%eax` always alternates between 0 and 4
- E this code has an assemble-time error

6 Assume the bytes starting at location `var1` are as follows:

```
0x01 0x02 0x03 0x04 0x00 0x05 0x06 0x07 0x08 0x09
```

What is the value of `%ecx` when the following code completes?

```
      movl $0,%ecx
      movl $var1,%eax
L1:   cmpb $0,(%eax)
      jz L2
      addl (%eax),%ecx
      addl $1,%eax
      jmp L1
L2:
```

- A 4
- B 6
- C 10
- D 20
- E the value is none of the above

7 Select a pseudocode for the following assembly code. Assume `var1` is an array of characters.

```

movl $0, %eax
movb $0,%b1
L1:
  cmpl $4,%eax
  jnb L2
  addl $1,%eax
  addb (%eax),b1
  jmp L1

```

L2:

- A `eax` \leftarrow 4
`b1` \leftarrow 0
while (`eax`) < 4 **do**
 `var1[eax]` \leftarrow `var1[eax] + 1`
 `b1` \leftarrow `b1 + 1`
end while
- B `eax` \leftarrow 4
`b1` \leftarrow 0
while (`eax`) \leq 4 **do**
 `eax` \leftarrow `eax + 1`
 `b1` \leftarrow `b1 + var1[eax]`
end while
- C `eax` \leftarrow 4
`b1` \leftarrow 0
while (`eax`) < 4 **do**
 `eax` \leftarrow `eax + 1`
 `b1` \leftarrow `b1 + var1[eax]`
end while
- D `eax` \leftarrow 4
`b1` \leftarrow 0
while (`eax`) > 4 **do**
 `eax` \leftarrow `eax + 1`
 `b1` \leftarrow `var1[b1]`
end while
- E `eax` \leftarrow 4
`b1` \leftarrow 0
while (`eax`) < 4 **do**
 `eax` \leftarrow `eax + 1`
 `b1` \leftarrow `var1[b1]`
end while

8 What is the value of register `eax` after the following code completes?

```

movl $1, %eax
addb %a1, %ah
addb %ah, %a1

```

- A 0x00020002
- B 0x00000102
- C 0x00020001
- D 0x00000201
- E 0x00000202

9 How do we interpret the following assembly instructions?

```

cmpl %eax, %ebx
jb label1
jz label1

```

- A jump to `label1` iff register `eax` is not greater than `ebx`, signed interpretation
- B jump to `label1` iff register `ebx` is not greater than `eax`, unsigned interpretation
- C jump to `label1` iff register `ebx` is not greater than `eax`, signed interpretation
- D jump to `label1` iff register `eax` is not greater than `ebx`, unsigned interpretation
- E jump to `label1` iff register `eax` is greater than `ebx`, unsigned interpretation

10 Which flag(s) (Carry, Sign, Overflow and Zero) is/are set after the following code?

```

movl $1, %eax
movl $-1,%ebx
subl %eax,%ebx

```

- A C, S
- B C, S, O
- C S, O
- D C, O
- E S

11 What are the byte values at `outthere` when the following code is about to execute the `nop` instruction? The byte values are expressed in hexadecimal representation, from low address (left) to high address (right).

```
.lcomm outthere 4
.text
.global _start
_start:
    movl $outthere, %eax
    movb $0x04, %c1
    movb $0x10, %b1
L1:
    movb %b1, (%eax)
    addb $2, %b1
    addl $1, %eax
    subb $1, %c1
    jnz L1

    nop
# the usual exit code
```

- A this is an infinite loop, we'll never execute `nop`
- B the byte values cannot be determined
- C 03 02 01 00
- D 10 10 10 10
- E 10 12 14 16

12 If you are only allowed to use the byte-wide instruction `addb`, how do you add the value of `var1` to the value of `var2`, then store the result in `var2`? You can use register `%a1`. Recall that the x86 architecture is little endian. Assume the variables are defined as follows:

```
.lcomm var1 2
.lcomm var2 2
```

- A


```
movb var1, %a1
addb %a1, var2
movb var1+1, %a1
jnc l1
addb $1,%a1
l1:
addb %a1,var2+1
```
- B


```
movb var1, %a1
addb %a1, var2
jnc l1
addb $1,%a1
l1:
movb var1+1, %a1
addb %a1,var2+1
```
- C


```
movb var1,%a1
addb %a1,var2
movb $1,%a1
addb var1+1,%a1
addb %a1,var2+1
```
- D


```
movb var1, %a1
addb %a1, var2
jnc l1
movb var1+1, %a1
addb $1,%a1
l1:
addb %a1,var2+1
```

E It is impossible to add more than 8 bits without some kind of add-with-carry instruction.