CS 354
Machine Organization and Programming
Lecture 17

Michael Doescher
Summer 2020

Assembly Languages
Control Flow
Conditionals

What happens when we run a program?
Compiling

<u>sum.c</u>

……
a = 1;
b = 2;
c = a + b;
……

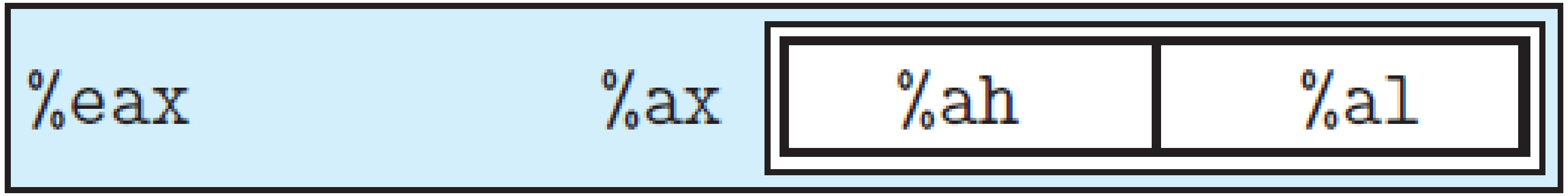
<u>sum.s</u>

……
movl 0x1000, %eax
movl0x1004, %ebx
addl %eax, %ebx
……

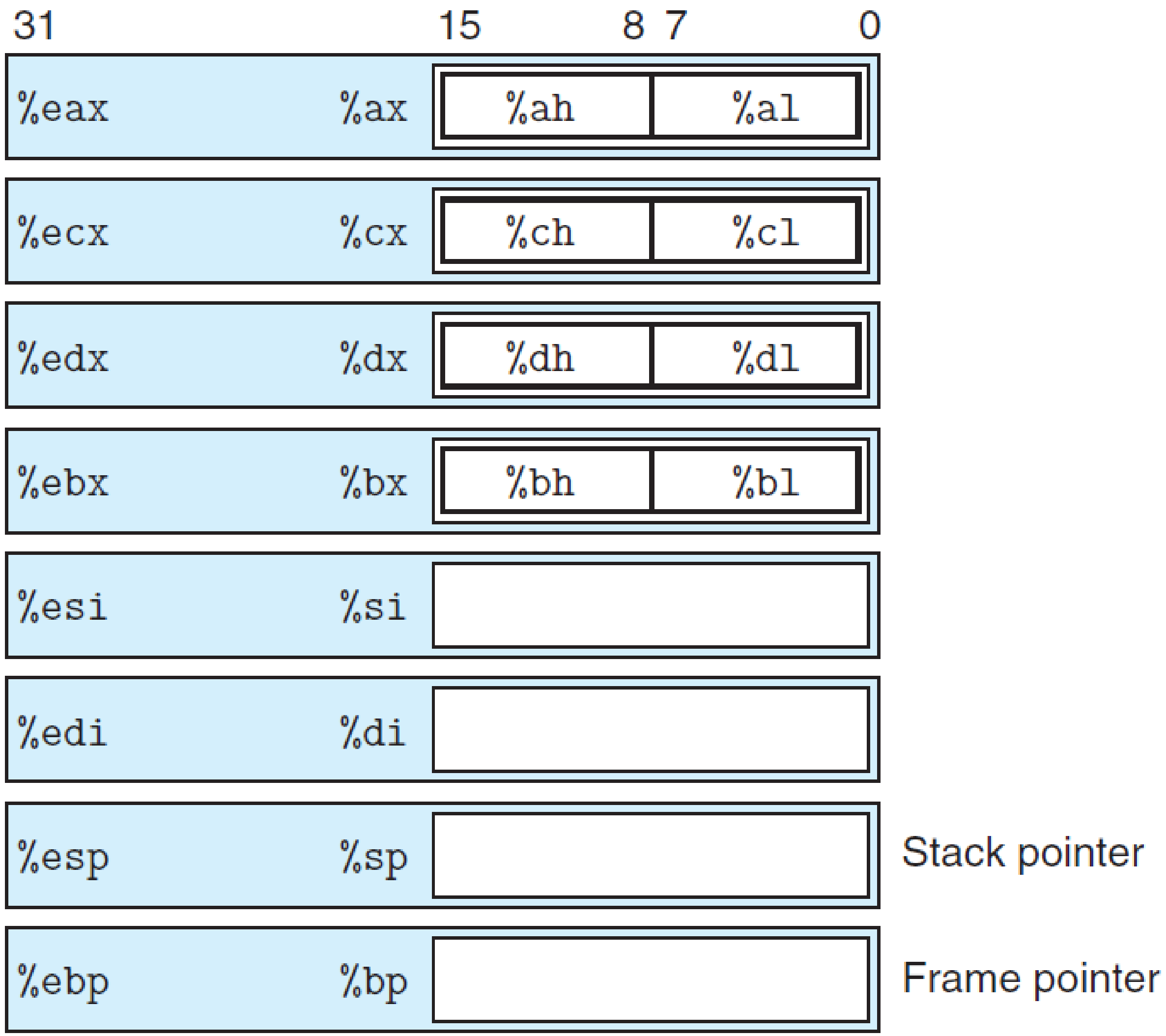
<u>sum</u>

00101010
10101010
10101010
10000110
11100110

Registers



Registers


31
15
8 7
0
%eax
%ax
%ah
%al
%ecx
%cx
%ch
%cl
%edx
%dx
%dh
%dl
%ebx
%bx
%bh
%bl
%esi
%si
%edi
%di
%esp
%sp
Stack pointer
%ebp
%bp
Frame pointer

Assembly Instructions : CS:APP 3.4

AT&T Syntax of x86

mov (movl, movb, movw)
Copies data from one location to another (source remains unchanged)

Assembly Instructions : CS:APP 3.4

AT&T Syntax of x86

mov (movl, movb, movw)
Copies data from one location to another (source remains unchanged)

movl imm, register
movl imm, memory
mov register, register
mov register, memory
mov memory, register

Assembly Instructions : CS:APP 3.4

AT&T Syntax of x86

mov (movl, movb, movw)
Copies data from one location to another (source remains unchanged)

movl imm, register
movl imm, memory
mov register, register
mov register, memory
mov memory, register

~~mov anything, immediate~~
~~mov memory, memory ::: movl 8(%eax), 0x7008~~

Assembly Instructions : CS:APP 3.4

AT&T Syntax of x86

mov (movl, movb, movw)
Copies data from one location to another (source remains unchanged)

movl imm, register
movl imm, memory
mov register, register
mov register, memory
mov memory, register

~~mov anything, immediate~~
~~mov memory, memory ::: movl 8(%eax), 0x7008~~

General Form
Imm(%R1, %R2, Scale) ::: computes address as Imm + %R1 + %R2*scale

Assembly Instructions : CS:APP 3.5

AT&T Syntax of x86

mov (movl, movb, movw)
lea : load effective address (& operator in C)

Assembly Instructions : CS:APP 3.5

AT&T Syntax of x86

mov (movl, movb, movw)
lea : load effective address (& operator in C)

Arithmetic
add, sub, imul, idiv
    (addl, addb, addw)
    destination <- source – destination
    integer division: dividend stored in %edx:%eax
    remainder in %edx and quotient in %eax

Assembly Instructions : CS:APP 3.5

AT&T Syntax of x86

mov (movl, movb, movw)
lea : load effective address (& operator in C)

Arithmetic
add, sub, imul, idiv
    (addl, addb, addw)
    destination <- source – destination
    integer division: dividend stored in %edx:%eax
    remainder in %edx and quotient in %eax
inc, dec

Assembly Instructions : CS:APP 3.5

AT&T Syntax of x86

mov (movl, movb, movw)
lea : load effective address (& operator in C)

Arithmetic
add, sub, imul, idiv
(addl, addb, addw)
destination <- source – destination
integer division: dividend stored in %edx:%eax
remainder in %edx and quotient in %eax
inc, dec

Bitwise operations
and, or, not, shifting

Control Flow : CS:APP 3.6

- Sequential

```
Stmt 1;
Stmt 2;
Stmt 3;
Stmt 4;
```

Control Flow : CS:APP 3.6

- Sequential
- Conditional

```
Stmt 1;
if (expr) {
    Stmt 2;
    Stmt 3;
}
Stmt 4;
```

Control Flow : CS:APP 3.6

- Sequential
- Conditional

```
Stmt 1;
if (expr) {
    Stmt 2;
    Stmt 3;
}
Stmt 4;
```

```
Stmt 1;
Stmt 2;
Stmt 3;
Stmt 4;
```

```
Stmt 1;
Stmt 4;
```

Control Flow : CS:APP 3.6

- Sequential
- Conditional

```
Stmt 1;
if (expr) {
    Stmt 2;
    Stmt 3;
}
else {
    Stmt 4;
    Stmt 5;
}
Stmt 6;
```

Control Flow : CS:APP 3.6

- Sequential
- Conditional

```
Stmt 1;
if (expr) {
    Stmt 2;
    Stmt 3;
}
else {
    Stmt 4;
    Stmt 5;
}
Stmt 6;
```

```
Stmt 1;
Stmt 2;
Stmt 3;
Stmt 6;
```

```
Stmt 1;
Stmt 4;
Stmt 5;
Stmt 6;
```

Control Flow : CS:APP 3.6

- Sequential
- Conditional
- Iteration

```
Stmt 1;
while (expr) {
    Stmt 2;
    Stmt 3;
}
Stmt 4;
```

Control Flow : CS:APP 3.6

- Sequential
- Conditional
- Iteration

```
Stmt 1;
while (expr) {
    Stmt 2;
    Stmt 3;
}
Stmt 4;
```

```
Stmt 1;
Stmt 4;
```

```
Stmt 1;
Stmt 2;
Stmt 3;
Stmt 4;
```

```
Stmt 1;
Stmt 2;
Stmt 3;
Stmt 2;
Stmt 3;
Stmt 2;
Stmt 3;
Stmt 2;
Stmt 3;
Stmt 4;
```

Control Flow : CS:APP 3.6

- Sequential
- Conditional
- Iteration
- Functions

Control Flow : CS:APP 3.6

- Sequential
- Conditional
- Iteration
- Functions

What do we need at the assembly level to implement these?

Control Flow : CS:APP 3.6

- Sequential
- Conditional
- Iteration
- Functions

What do we need at the assembly level to implement these?

Boolean Operators

<, <=, >, >=, !=, ==

Control Flow : CS:APP 3.6

- Sequential
- Conditional
- Iteration
- Functions

What do we need at the assembly level to implement these?
Remember we have a bunch of registers, most importantly

- Instruction Pointer %eip
- Condition Code Register, %eflags

Condition Codes: Addition

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

Condition Codes: Addition: ZF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

```
T = a + b
T = 3 + (-3)
if (T == 0)
    ZF -> 1 (set)
else
    ZF -> 0 (not set)
            (unset)
```

Condition Codes: Addition: CF

%eflags
- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

```
T = a + b

 3   011
+4   100
--------
 7   111

No carry out
Required
CF = 0
```

Condition Codes: Addition : CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

Unsigned Numbers

```
T = a + b

 3  011
+4  100
--------
 7  111

No carry out
Required
CF = 0
```

```
T = a + b

 3    011
+5    101
---------
```

Condition Codes: Addition : CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

Unsigned Numbers

```
T = a + b

 3  011
+4  100
-------
 7  111

No carry out
Required
CF = 0
```

```
T = a + b

        1
 3    011
+5    101
---------
        0
```

Condition Codes: Addition : CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

```
T = a + b

 3  011
+4  100
-------
 7  111

No carry out
Required
CF = 0
```

```
T = a + b

         11
 3      011
+5      101
-----------
         00
```

Condition Codes: Addition : CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

```
T = a + b


 3  011
+4  100
-------
 7  111

No carry out
Required
CF = 0
```

```
T = a + b

        111
 3      011
+5      101
-----------
        000
```

Condition Codes: Addition : CF

%eflags
- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

Unsigned Numbers

```
T = a + b



 3   011
+4   100
----------
 7   111

No carry out
Required
CF = 0
```

```
T = a + b

         111
 3       011
+5       101
------------
      (1)000

Carry out
CF = 1
```

Condition Codes: Subtraction: CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

`T = a + b`

```
 1  001
-2  010
```

Condition Codes: Subtraction: CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

T = a + b

```
 1  001
-2  010
-------
      1
```

Condition Codes: Subtraction: CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

T = a + b

```
     1
  1  001
-2  010
---------
       1
```

Condition Codes: Subtraction: CF

%eflags
- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

```
T = a + b

    02
 1  001
-2  010
-------
      1
```

Condition Codes: Subtraction: CF

%eflags
- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

```
T = a + b

     012
  1  001
 -2  010
 --------
       1
```

Condition Codes: Subtraction: CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

`T = a + b`

```
    012
 1  001
-2  010
-------
     11
```

Condition Codes: Subtraction: CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Unsigned Numbers

```
T = a + b

   012
 1  001
-2  010
--------
    111
```

Condition Codes: Subtraction: CF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

Unsigned Numbers

T = a + b

```
    012
 1  001
-2  010
-------
 7  111
```

Carry Flag is set when we Borrow during subtraction Also.

CF = 1

Condition Codes: Addition: SF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Signed Numbers

```
T = a + b

if T < 0 SF = 1
else     SF = 0

   3   011
+(-4)  100
```

Condition Codes: Addition: SF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Signed Numbers

```
T = a + b

if T < 0 SF = 1
else     SF = 0

    3   011
+(-4)   100
-----------
   -1   111
```

Condition Codes: Addition: OF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Signed Numbers

```
T = a + b

 2   010
+1   001
---------
```

```
T = a + b

 3   011
+1   001
---------
```

Condition Codes: Addition: OF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Signed Numbers

```
T = a + b

 2   010
+1   001
---------
 3   011
```

```
T = a + b

 3   011
+1   001
---------
```

Condition Codes: Addition: OF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Signed Numbers

```
T = a + b

 2   010
+1   001
---------
 3   011
```

```
T = a + b

 3   011
+1   001
---------
-4   100
```

Condition Codes: Addition: OF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Signed Numbers

```
T = a + b

 2   010
+1   001
---------
 3   011

 OF = 0
```

```
T = a + b

 3   011
+1   001
---------
-4   100

 OF = 1
```

Condition Codes: Addition: OF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

Signed Numbers

```
T = a + b      T = a + b

 2   010        3   011
+1   001       +1   001
---------      ---------
 3   011       -4   100

 OF = 0         OF = 1

-4, -3, -2, -1, 0, 1, 2, 3
```

Condition Codes: Addition: OF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly
after every operation

Signed Numbers

```
T = a + b      T = a + b

 2   010        3   011
+1   001       +1   001
---------      ---------
 3   011       -4   100

 OF = 0         OF = 1
```

```
-4, -3, -2, -1, 0, 1, 2, 3
Any positive number + any negative
number yield a number from this set
```

Condition Codes: Addition: OF

%eflags

- ZF = Zero Flag
- CF = Carry Flag
- SF = Sign Flag
- OF = Overflow Flag

Updated implicitly after every operation

Signed Numbers

```
T = a + b        T = a + b

 2   010          3   011
+1   001         +1   001
---------        ---------
 3   011         -4   100

 OF = 0           OF = 1
```

```
-4, -3, -2, -1, 0, 1, 2, 3
if (a>0 && b>0 && t<0) OF = 1
or (a<0 && b<0 && t>0) OF = 1
else OF = 0
```

COMPARE INSTRUCTION

cmpl b,a

Evaluates a-b
But does not store
the result

%eflags

- ZF =
- CF =
- SF =
- OF =

COMPARE INSTRUCTION

cmpl b,a

Evaluates a-b
But does not store
the result

%eflags

- ZF = a-b == 0
- CF =
- SF =
- OF =

COMPARE INSTRUCTION

cmpl b,a

Evaluates a-b
But does not store the result

%eflags

- ZF = a-b == 0
- CF = a<b
- SF =
- OF =

- CF is set when we borrow in during subtraction of unsigned numbers.
- Borrowing is required when we subtract a bigger number from a smaller number

COMPARE INSTRUCTION

cmpl b,a

Evaluates a-b
But does not store
the result

%eflags

- ZF = a-b == 0
- CF = a-b < 0 or a<b
- SF = a-b < 0 or a<b
- OF =

- SF similar to CF but for signed numbers
- Subtracting a bigger number from a smaller number results in a negative number
- Same as saying a < b

COMPARE INSTRUCTION

cmpl b,a

Evaluates a-b
But does not store
the result

%eflags

- ZF = a-b == 0
- CF = a-b < 0 or a<b
- SF = a-b < 0 or a<b
- OF =

```
    (a>0 && b<0 && (a-b)<0)
or  (a<0 && b>0 && (a-b)>0)
```

TEST INSTRUCTION

testl b,a

Evaluates a&b
But does not store
the result

Almost always used with
the same operands

testl a,a

%eflags

- ZF =
- CF =
- SF =
- OF =

TEST INSTRUCTION

testl b,a

Evaluates a&b
But does not store
the result

Almost always used with
the same operands

testl a,a

%eflags

- ZF = a&b = 0 (a&a ==0 -> a ==0)
- CF =
- SF =
- OF =

TEST INSTRUCTION

testl b,a

Evaluates a&b
But does not store
the result

Almost always used with
the same operands

testl a,a

%eflags

- ZF = a&b = 0 (a&a == 0 -> a ==0)
- CF =
- SF = a&b<0 (a&a -> a < 0)
- OF =

TEST INSTRUCTION

testl b,a

Evaluates a&b
But does not store
the result

%eflags

- ZF = a&b = 0 (a&a == 0 -> a ==0)
- CF =
- SF = a&b<0 (a&a -> a < 0)
- OF =

```
If %eax == 0  |  testl %eax, %eax  | ZF = 1, SF = 0
If %eax  < 0  |  testl %eax, %eax  | ZF = 0, SF = 1
If %eax  > 0  |  testl %eax, %eax  | ZF = 0, SF = 0
```

JUMP INSTRUCTION

```
Instr 1
jmp Target       // unconditional jump
Instr 2
Instr 3
Target:          // a label
Instr 4
Instr 5
```

Conditional JUMP INSTRUCTIONS

```
Instr 1          // set the flags
jz Target        // jump if ZF
Instr 2          // if block
Instr 3
Target:          // label no else block
Instr 4
Instr 5
```

Conditional JUMP INSTRUCTIONS

```
Instr 1         // set the flags
jz Target       // jump if ZF
Instr 2         // if block
Instr 3
Target:         // label no else block
Instr 4
Instr 5
```

```
We also have jump instructions
jz, jnz  (jump if zero, jump if not zero)
jl, jle, jg, jge, je, jne  (<, <=, >, >=, ==, !=)
jb, jbe, ja, jae (above, below – for unsigned numbers)
```

JUMP CONDITIONS

```
jmp        No Flag Requirements
je         ZF
jne        ~ZF
jl         SF^OF
jle        SF^OF | ZF
jg         ~(SF^OF)
jge        ~(SF^OF) | ZF

jb         CF
jbe        CF | ZF
ja         ~CF & ~ZF
jae        ~CF
```

Worksheet Problem 6

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
}
```

Worksheet Problem 6

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
}

Condition
Jump
DO:

DONT:
```

Worksheet Problem 6

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
}
```

```
  cmpl %ebx, %eax
  jle DONT
DO:
  addl $1, %eax
DONT:
```

Worksheet Problem 6

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
}
```

```
  cmpl %ebx, %eax
  jle DONT
  addl $1, %eax
DONT:
```

Worksheet Problem 7

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
} else {
      b = a;
}
```

Worksheet Problem 7

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
} else {
      b = a;
}

  condition
  jump
DO:

DONT:

END:
```

Worksheet Problem 7

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
} else {
      b = a;
}
```

```
  cmpl %ebx, %eax
  jle DONT
DO:
  addl $1, %eax
  jmp END
DONT:
  movl %eax, %ebx
END:
```

Worksheet Problem 7

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
if(a>b) {
      a++;
} else {
      b = a;
}
```

```
  cmpl %ebx, %eax
  jle DONT
  addl $1, %eax
  jmp END
DONT:
  movl %eax, %ebx
END:
```

Worksheet Problem 8

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
while (b > 0) {
      a++;
      b--;
}
```

Worksheet Problem 8

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
while (b > 0) {
      a++;
      b--;
}

TOP:
  condition
  jump BOTTOM
  statements
  jump TOP
BOTTOM:
```

Worksheet Problem 8

Assume the value of a is in %eax, and the value of b is in %ebx
Write x86 assembly code for:

```
while (b > 0) {
      a++;
      b--;
}
```

```
TOP:
  testl %ebx, %ebx
  jle BOTTOM
  incl %eax
  decl %ebx
  jmp TOP
BOTTOM:
```