Introduction to 8086 Assembly

Lecture 5

Jump, Conditional Jump, Looping, Compare instructions

Labels and jumping (the jmp instruction)

```
  mov eax, 1

  add eax, eax

  jmp label1

  xor eax, eax

label1:

  sub eax, 303
```

Labels and jumping (the jmp instruction)

```
   mov eax, 1

   add eax, eax

   jmp label1

   xor eax, eax

label1:

   sub eax, 303
```

label1: → address of `sub eax, 303`

Infinite loop

```
  mov eax, 0

loop1:

  call print_int
  call print_nl

  inc eax

  jmp loop1
```

infinite_loop.asm

Remember: the FLAGS Register

K. N. Toosi
University of Technology

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟩	🟥	🟩	🟥
				Overflow Flag (OF)	Direction Flag (DF)	Interrupt Flag (IF)	Trap Flag (TF)	Sign Flag (SF)	Zero Flag (ZF)		Auxiliary Carry Flag (AF)		Parity Flag (PF)		Carry Flag (CF)

CF: carry flag
OF: overflow flag
SF: sign flag
ZF: zero flag

PF: parity flag
DF: direction flag
IF: interrupt flag

Conditional loops

JZ	Jump if ZF=1
JNZ	Jump if ZF=0
JO	Jump if OF=1
JNO	Jump if OF=0
JS	Jump if SF=1
JNS	Jump if SF=0
JC	Jump if CF=1
JNC	Jump if CF=0
JP	Jump if PF=1
JNP	Jump if PF=0

Example

unsigned integer:

```
if (eax == ebx)
    esi = 0
```

JZ	Jump if ZF=1
JNZ	Jump if ZF=0
JO	Jump if OF=1
JNO	Jump if OF=0
JS	Jump if SF=1
JNS	Jump if SF=0
JC	Jump if CF=1
JNC	Jump if CF=0
JP	Jump if PF=1
JNP	Jump if PF=0

Example

unsigned integer:

```
if (eax == ebx)
    esi = 0
```

```
sub eax, ebx
jnz next

mov esi, 0

next:
```

JZ	Jump if ZF=1
JNZ	Jump if ZF=0
JO	Jump if OF=1
JNO	Jump if OF=0
JS	Jump if SF=1
JNS	Jump if SF=0
JC	Jump if CF=1
JNC	Jump if CF=0
JP	Jump if PF=1
JNP	Jump if PF=0

Example

signed integer:

```
if (eax == ebx)
    esi = 0
```

```
sub eax, ebx
jnz next

mov esi, 0

next:
```

JZ	Jump if ZF=1
JNZ	Jump if ZF=0
JO	Jump if OF=1
JNO	Jump if OF=0
JS	Jump if SF=1
JNS	Jump if SF=0
JC	Jump if CF=1
JNC	Jump if CF=0
JP	Jump if PF=1
JNP	Jump if PF=0

Example

signed integer:

```
if (eax == - ebx)
    edi = 4
```

JZ	Jump if ZF=1
JNZ	Jump if ZF=0
JO	Jump if OF=1
JNO	Jump if OF=0
JS	Jump if SF=1
JNS	Jump if SF=0
JC	Jump if CF=1
JNC	Jump if CF=0
JP	Jump if PF=1
JNP	Jump if PF=0

Example

signed integer:

```
if (eax == - ebx)
    edi = 4
```

```
add eax, ebx
jnz next

mov edi, 4

next:
```

JZ	Jump if ZF=1
JNZ	Jump if ZF=0
JO	Jump if OF=1
JNO	Jump if OF=0
JS	Jump if SF=1
JNS	Jump if SF=0
JC	Jump if CF=1
JNC	Jump if CF=0
JP	Jump if PF=1
JNP	Jump if PF=0

Example

unsigned integer:

```
if (eax >= ebx)
    esp -= 4
```

`JZ`	Jump if ZF=1
`JNZ`	Jump if ZF=0
`JO`	Jump if OF=1
`JNO`	Jump if OF=0
`JS`	Jump if SF=1
`JNS`	Jump if SF=0
`JC`	Jump if CF=1
`JNC`	Jump if CF=0
`JP`	Jump if PF=1
`JNP`	Jump if PF=0

Example

unsigned integer:

```
if (eax >= ebx)
    esp -= 4
```

```
sub eax, ebx
jc next

sub esp, 4

next:
```

`JZ`	Jump if ZF=1
`JNZ`	Jump if ZF=0
`JO`	Jump if OF=1
`JNO`	Jump if OF=0
`JS`	Jump if SF=1
`JNS`	Jump if SF=0
`JC`	Jump if CF=1
`JNC`	Jump if CF=0
`JP`	Jump if PF=1
`JNP`	Jump if PF=0

Example

signed integer:

```
if (eax < ebx)
     ebp += 8
```

x - y

x < y => SF = 1

x >= y => SF = 0

Example

signed integer:

```
if (eax < ebx)
    ebp += 8
```

`x - y`

OF=0	x < y => SF = 1 x >= y => SF = 0

Example

signed integer:

```
if (eax < ebx)
     ebp += 8
```

```
x - y
```

OF=0	$x < y \Rightarrow SF = 1$ $x >= y \Rightarrow SF = 0$
OF=1	$x < 0 < y \Rightarrow SF = 0$ $x > 0 > y \Rightarrow SF = 1$

Example

signed integer:

```
if (eax < ebx) ebp += 8

sub eax, ebx
jo  ovflow
jns endl
if_cond:
add ebp, 8
jmp endl
ovflow:
jns if_cond
endl:
```

x - y

OF=0	x < y => SF = 1 x >= y => SF = 0
OF=1	x < 0 < y => SF = 0 x > 0 > y => SF = 1

Example

signed integer:

```
if (eax < ebx)
    ebp += 8
else
    ebp -= 8
```

x - y

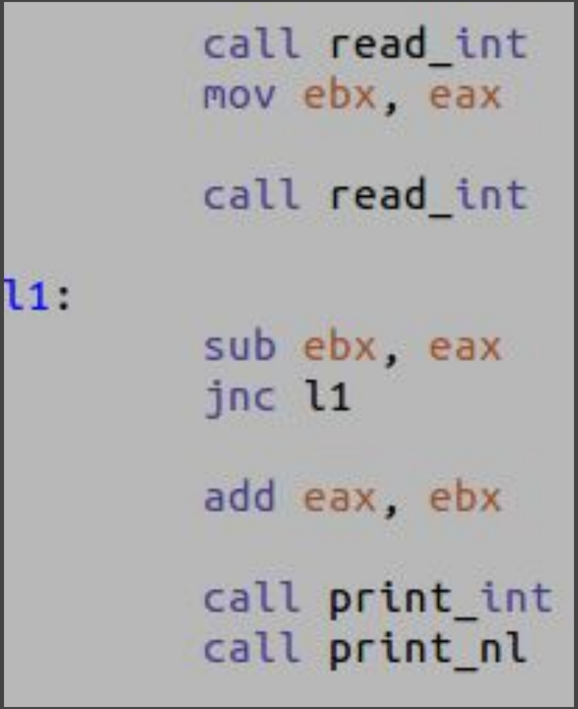
OF=0	x < y => SF = 1 x >= y => SF = 0
OF=1	x < 0 < y => SF = 0 x > 0 > y => SF = 1

Other conditional jump commands

```
sub x, y
```

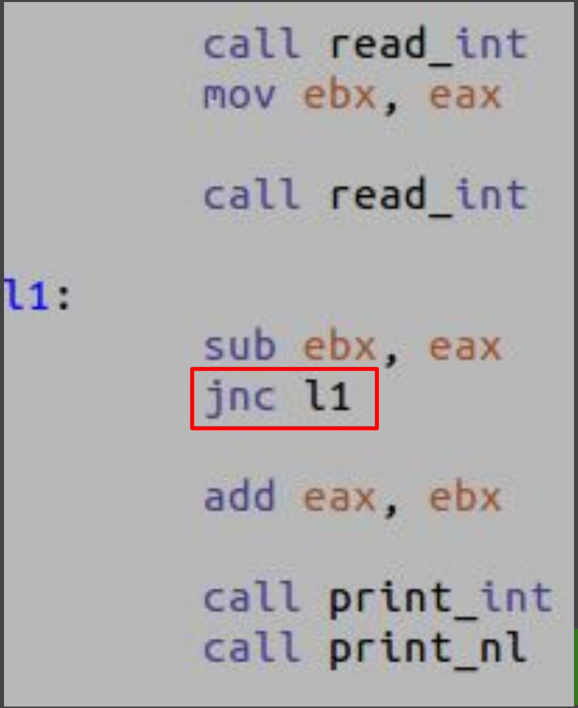
unsigned		signed	
JE label	jump if x == y	JE label	jump if x == y
JNE label	jump if x != y	JNE label	jump if x != y
JA label JNBE label	jump if x > y	JG label JNLE label	jump if x > y
JB label JNAE label	jump if x < y	JL label JNGE label	jump if x < y
JAE label JNB label	jump if x >= y	JGE label JNL label	jump if x >= y
JBE label JNA label	jump if x <= y	JLE label JNG label	jump if x <= y

Example:



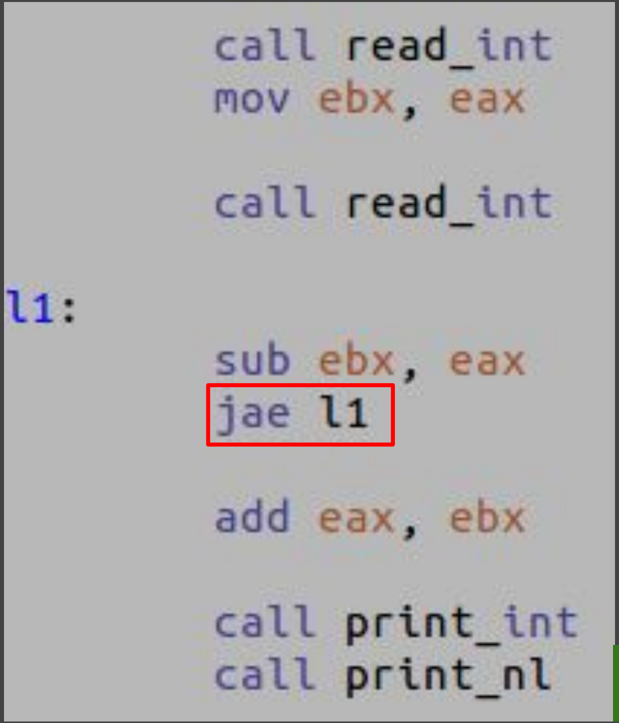
```
        call read_int
        mov ebx, eax

        call read_int

l1:
        sub ebx, eax
        jnc l1

        add eax, ebx

        call print_int
        call print_nl
```

Example:



```
        call read_int
        mov ebx, eax

        call read_int

l1:

        sub ebx, eax
        jnc l1

        add eax, ebx

        call print_int
        call print_nl
```

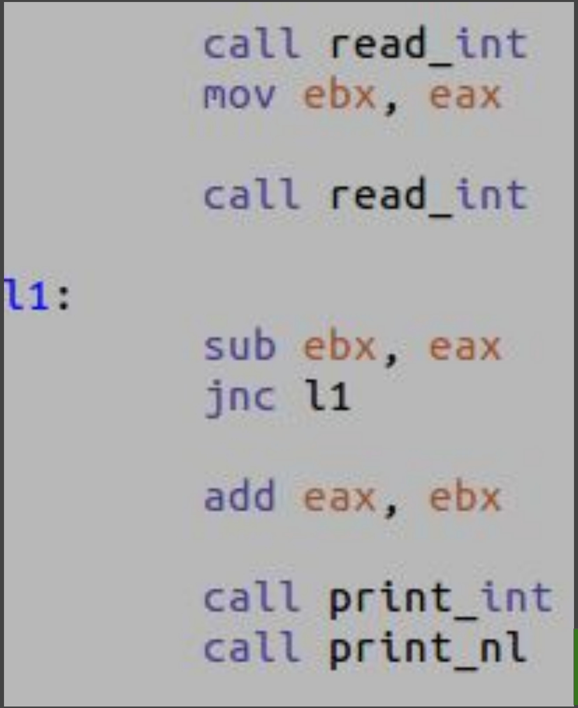
rem.asm



```
        call read_int
        mov ebx, eax

        call read_int

l1:

        sub ebx, eax
        jae l1

        add eax, ebx

        call print_int
        call print_nl
```

rem2.asm

Example:

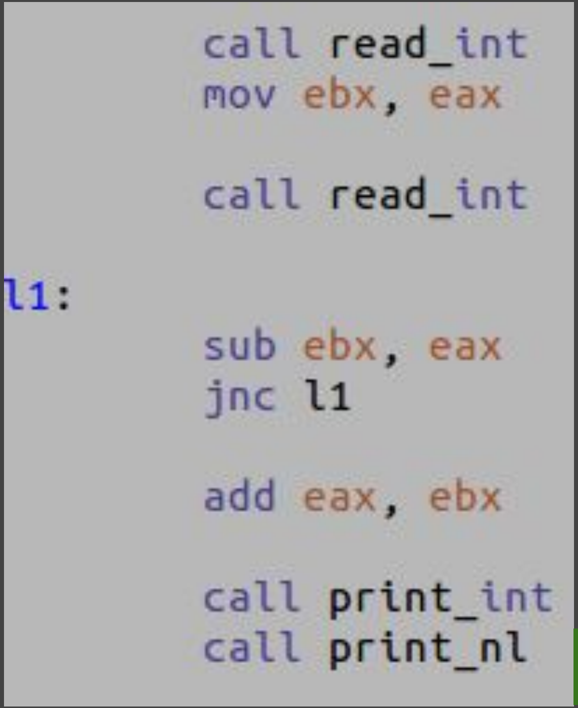


```
        call read_int
        mov ebx, eax

        call read_int

l1:

        sub ebx, eax
        jnc l1

        add eax, ebx

        call print_int
        call print_nl
```

rem.asm

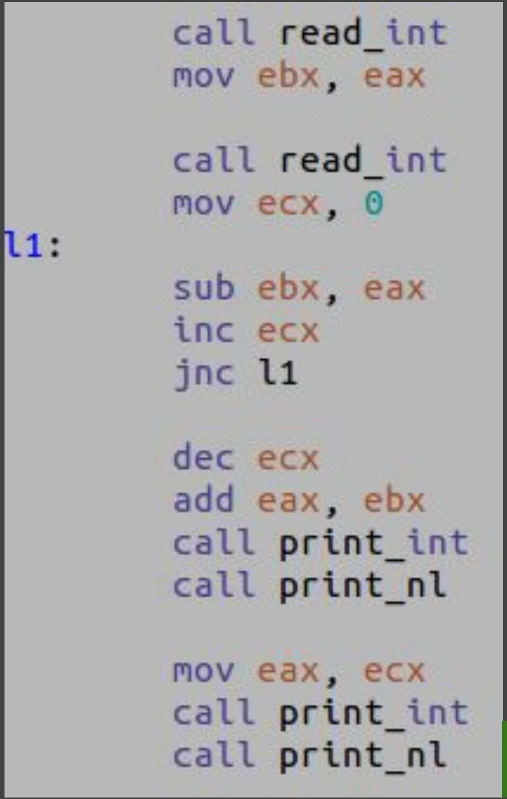
Practice: Also print quotient

Example:



```
        call read_int
        mov ebx, eax

        call read_int

l1:

        sub ebx, eax
        jnc l1

        add eax, ebx

        call print_int
        call print_nl
```

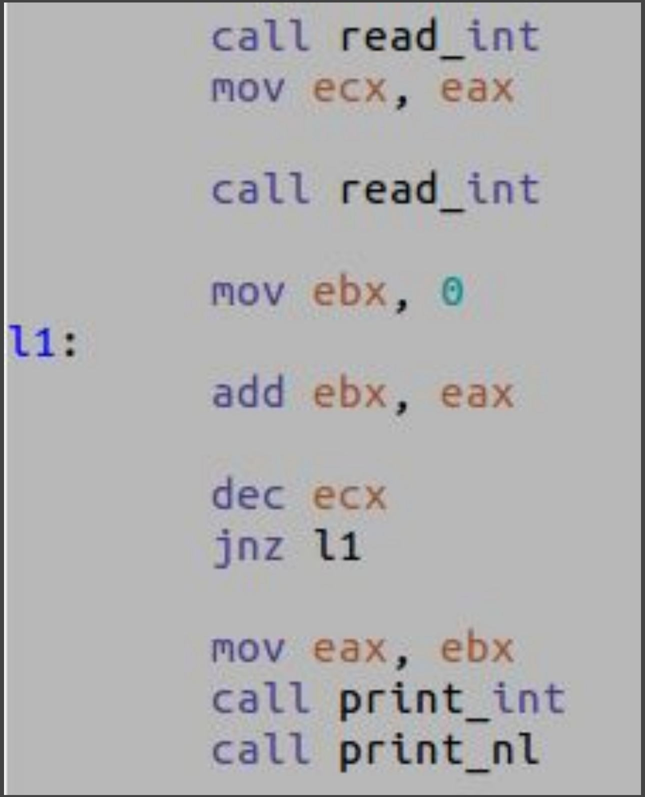
rem.asm



```
        call read_int
        mov ebx, eax

        call read_int
        mov ecx, 0
l1:
        sub ebx, eax
        inc ecx
        jnc l1

        dec ecx
        add eax, ebx
        call print_int
        call print_nl

        mov eax, ecx
        call print_int
        call print_nl
```

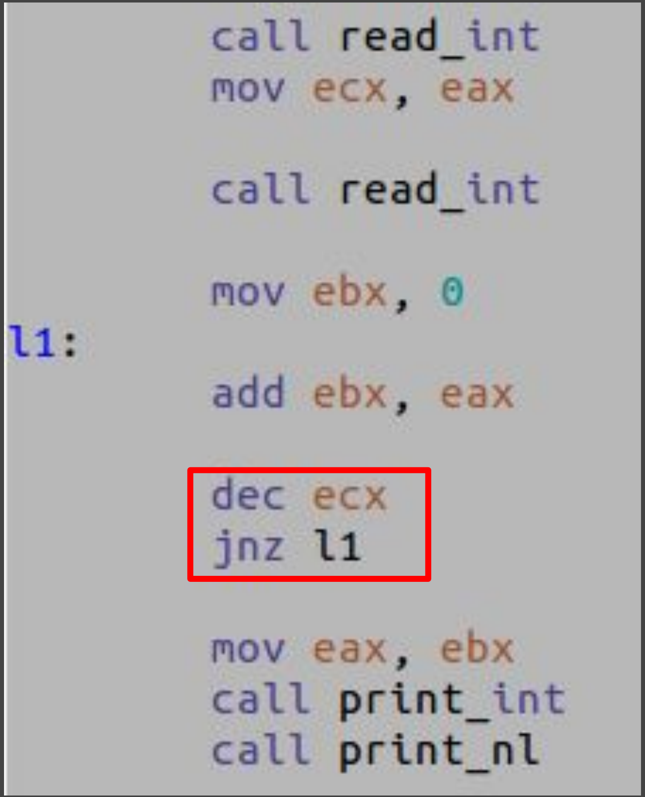
div.asm

Example:

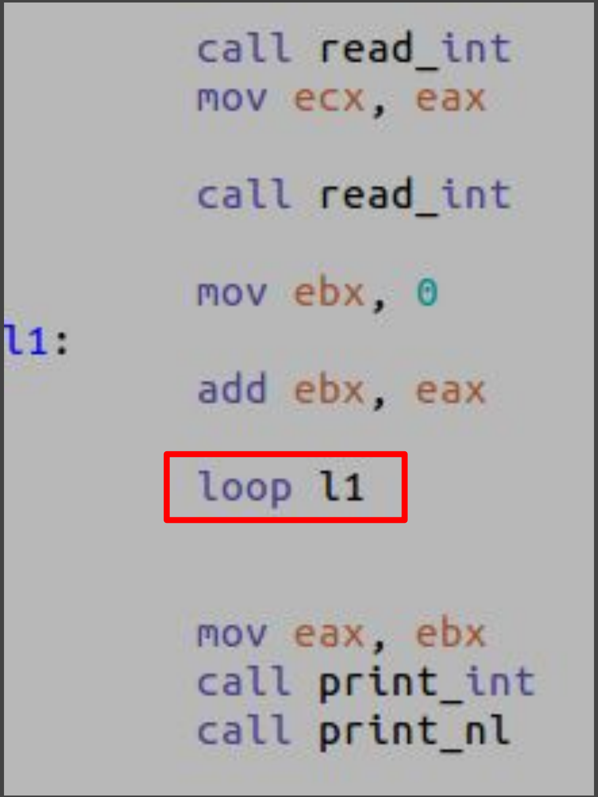


```
        call read_int
        mov ecx, eax

        call read_int

        mov ebx, 0
l1:
        add ebx, eax

        dec ecx
        jnz l1

        mov eax, ebx
        call print_int
        call print_nl
```

Example:



```
        call read_int
        mov ecx, eax

        call read_int

        mov ebx, 0
l1:
        add ebx, eax

        dec ecx
        jnz l1

        mov eax, ebx
        call print_int
        call print_nl
```

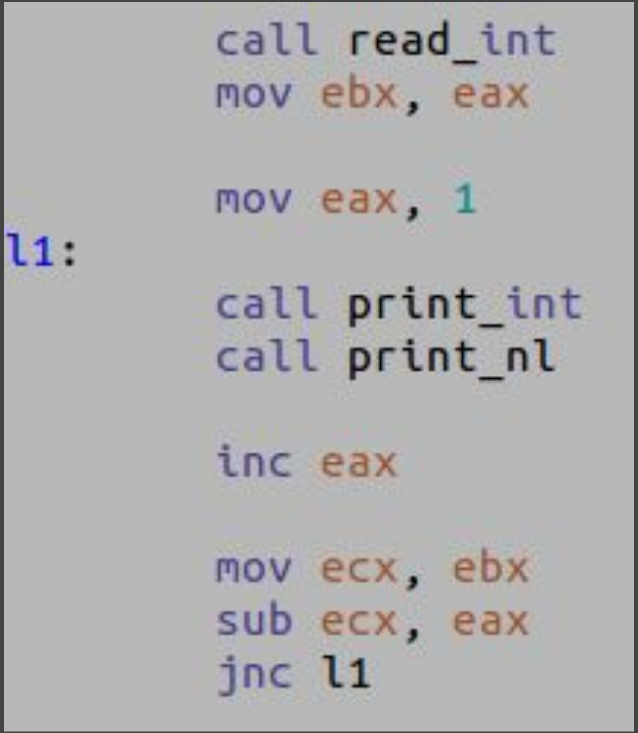


```
        call read_int
        mov ecx, eax

        call read_int

        mov ebx, 0
l1:
        add ebx, eax

        loop l1


        mov eax, ebx
        call print_int
        call print_nl
```

The loop commands

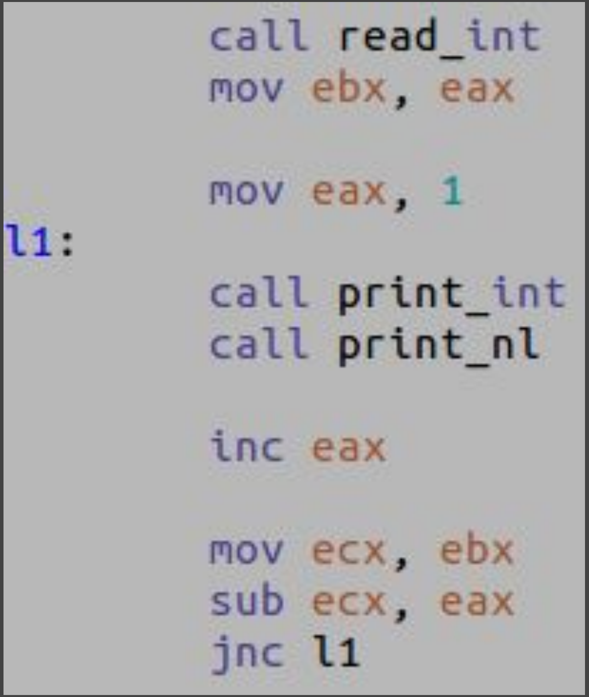
```
loop     lbl       ecx--; if (ecx!=0) goto lbl
loopz    lbl       ecx--; if (ecx!=0 && ZF=1) goto lbl
loopnz   lbl       ecx--; if (ecx!=0 && ZF=0) goto lbl
```

Example: Count up to N

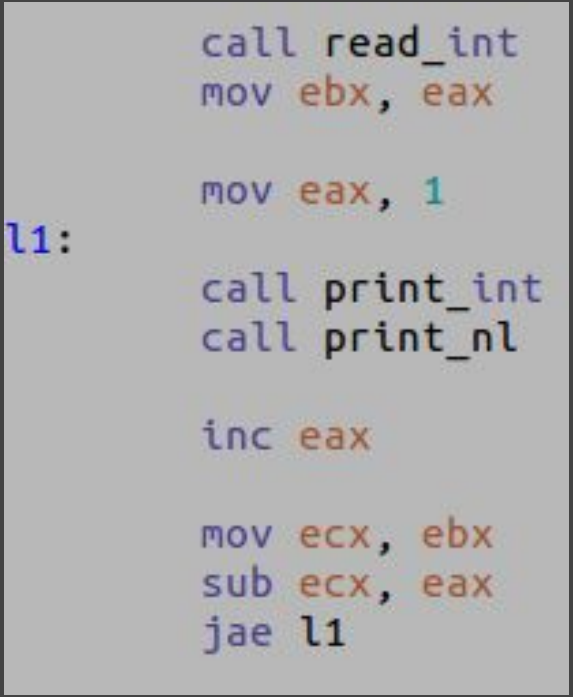


```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jnc l1
```

Example: Count up to N

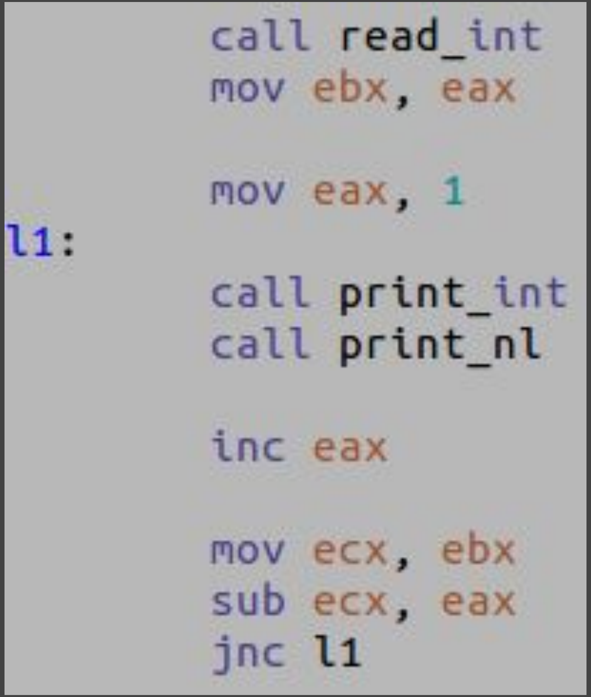


```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jnc l1
```

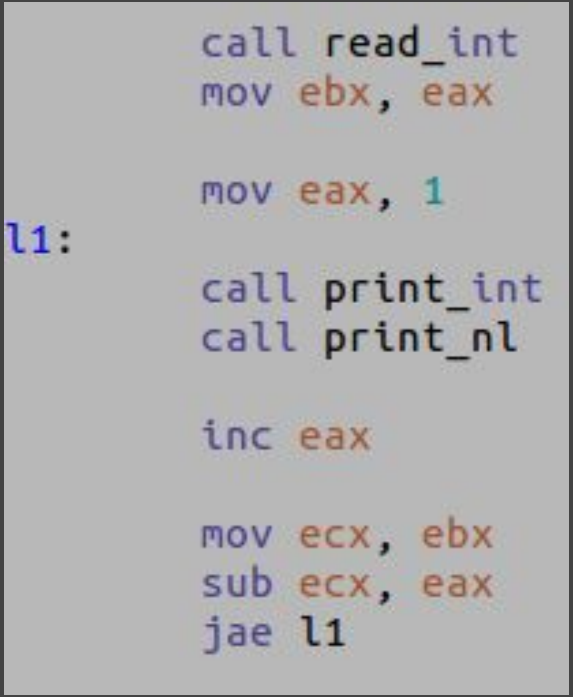


```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jae l1
```

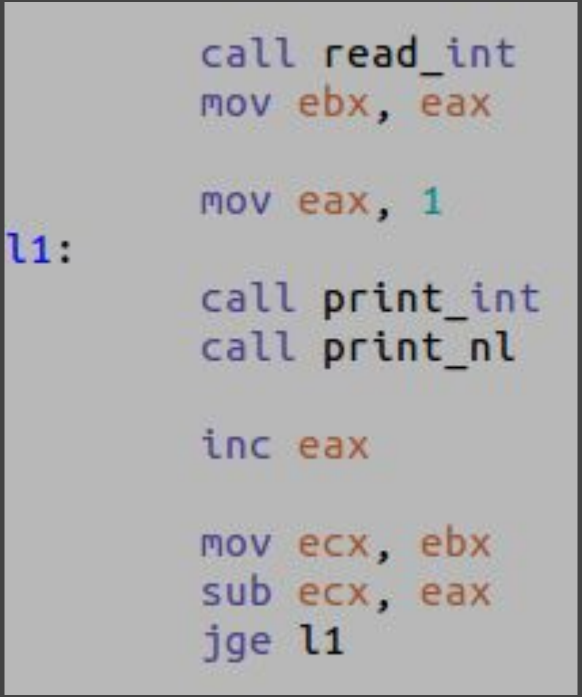
Example: Count up to N



```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jnc l1
```

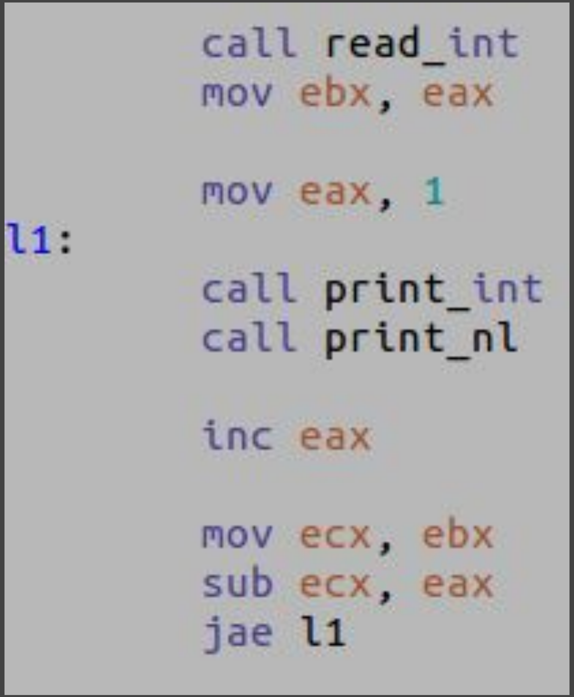


```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jae l1
```



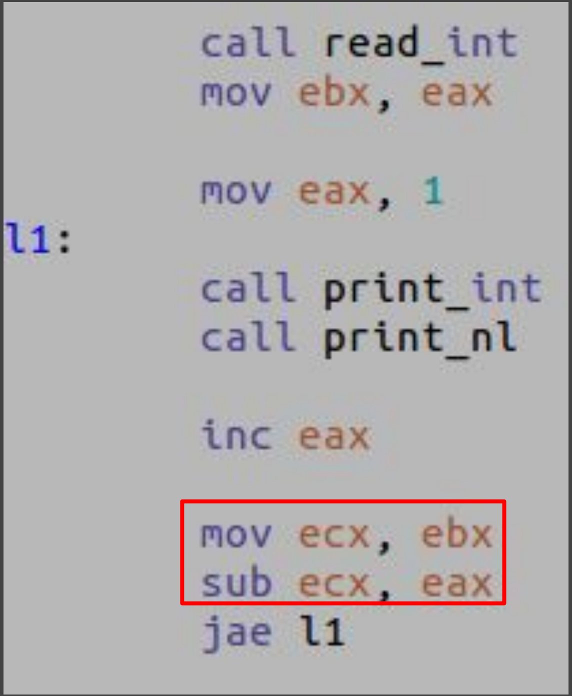
```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jge l1
```

using sub before jump; what's wrong?

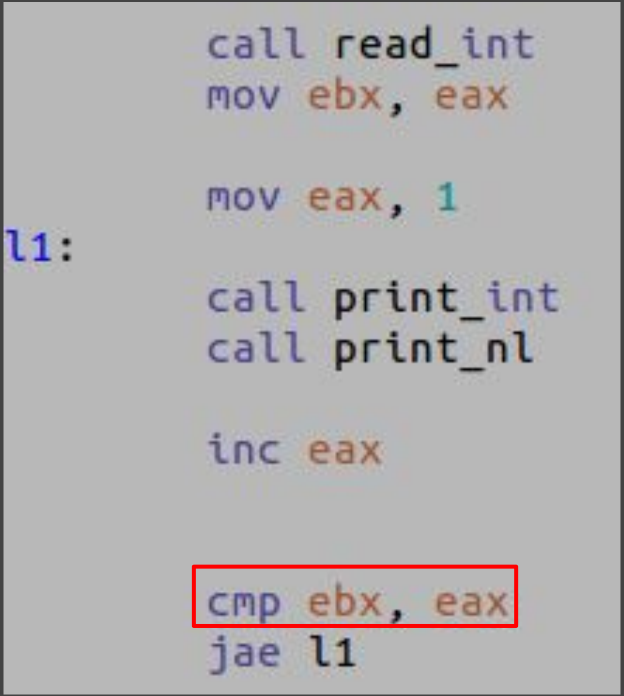


```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jae l1
```

the cmp instruction



```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax

        mov ecx, ebx
        sub ecx, eax
        jae l1
```



```
        call read_int
        mov ebx, eax

        mov eax, 1
l1:
        call print_int
        call print_nl

        inc eax


        cmp ebx, eax
        jae l1
```

The cmp instruction

```
sub eax, ebx
cmp eax, ebx
```

- `cmp x, y`
- subtracts y from x (like `sub x,y`)
- does not store the result (x is not changed)
- flags are set (as though a subtraction has taken place)

The cmp instruction

```
cmp x, y
```

unsigned		signed	
JE label	jump if x == y	JE label	jump if x == y
JNE label	jump if x != y	JNE label	jump if x != y
JA label JNBE label	jump if x > y	JG label JNLE label	jump if x > y
JB label JNAE label	jump if x < y	JL label JNGE label	jump if x < y
JAE label JNB label	jump if x >= y	JGE label JNL label	jump if x >= y
JBE label JNA label	jump if x <= y	JLE label JNG label	jump if x <= y

Practice

if (eax > ebx) {edi=1} else {edi=2}   (signed)

Practice

if (eax > ebx) {edi=1} else {edi=2}   (signed)

```
   cmp eax, ebx
   jle else_lbl
   mov edi, 1
   jmp endif

else_lbl:
   mov edi, 2

endif:
```