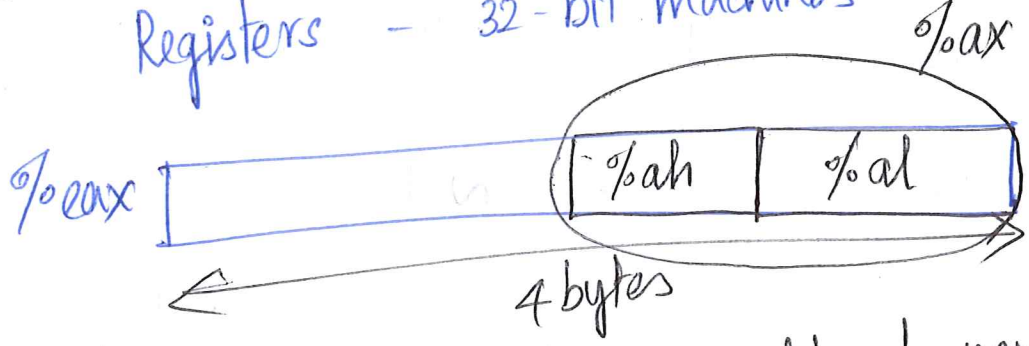
CS 354 - Lecture 7

①

Review

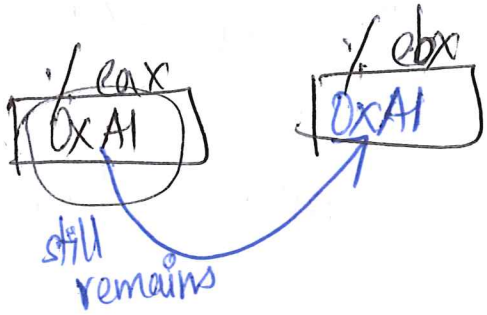
Registers - 32-bit machines



%eip - Instr. pointer - addr of next instr to be executed.

movl S, D

movl %eax, %ebx



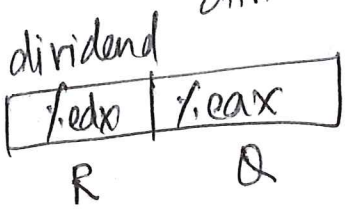
Arithmetic

addl S, D D ← D + S

subl S, D D ← D - S

imull S, D D ← D * S

idivl (op) divisor



andl S, D D ← D & S

xorl S, D D ← D ^ S

orl S, D D ← D | S

Unary

incl D D ← D+1

negl D D ← -D

notl D ← ~D

X = 1010

~X = 0101

Memory

movl 0x1000, %eax

movl N(R1, R2, K), %eax

addr = N + contents of R1 + (K * contents of R2)

Plan for today

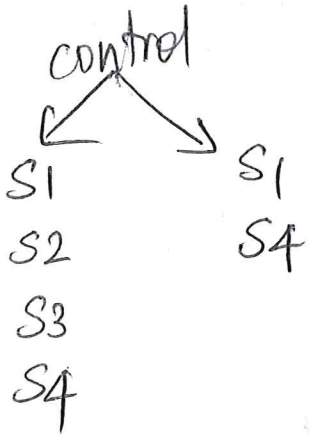
1. Control flow (X)
   - If / Then
   - Loops
   - Functions
2. Stack Pointer (%esp)
3. Load Effective Addr (LEAL)

Control Flow

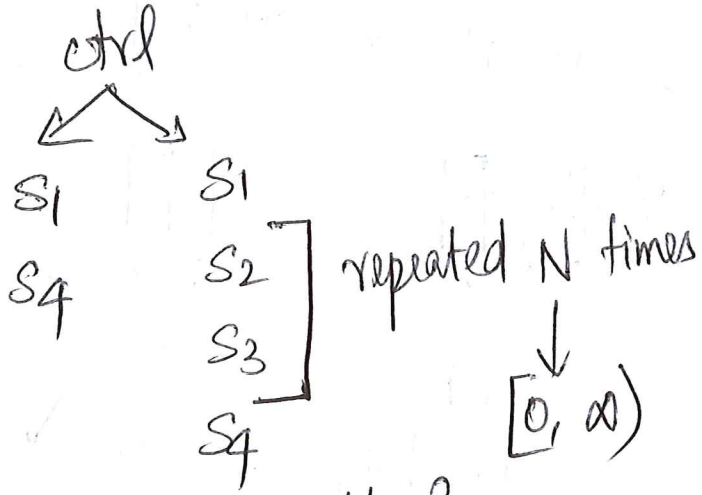
Decisions

③

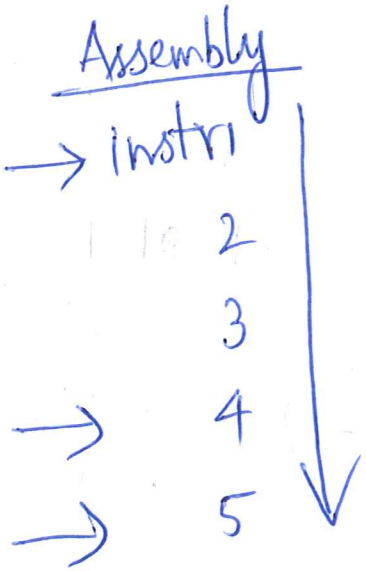
```
S1;
if (expr) {
    S2;
    S3;
}
S4;
```




```
S1;
while (expr) {
    S2;
    S3;
}
S4;
```



What help do we need from Assembly?



① jump to some arbitrary instr.

② evaluate an expression

Evaluating Expressions ④

if (a > b) {

}

Instructions

1. `cmpl b, a`

→ evaluate: a − b

result of sub. is discarded.

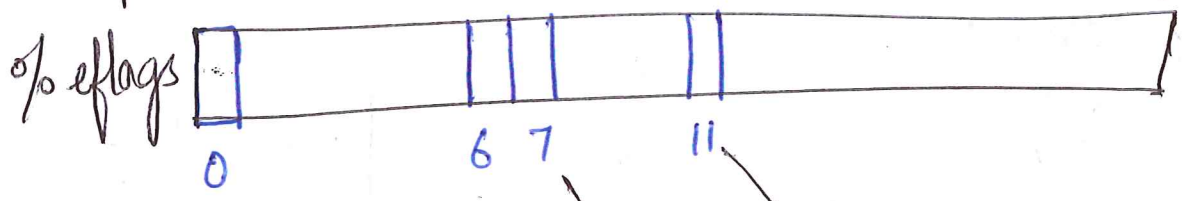
`subl S, D`

D ← D − S

→ updates a spl. registed %eflags (Condition Code)

OR

PSW



"state" on CPU.

CF Carry Flag

ZF Zero Flag

SF Sign Flag

OF overflow flag.

ZF a=1 b=1

cmpl b, a (a − b)

if a − b == 0 ⇒ ZF = 1.

SF a=1 b=2

cmpl b, a (a − b)

if a − b < 0 ⇒ SF = 1.

MSB

$\frac{1}{-2^2}$ $\frac{0}{2^1}$ $\frac{1}{2^0}$

1 − 2 = (−1) = −4 + 2 + 1

= (−1)

3-bits → int

	unsigned	signed
000	0	0
001	1	+1
010	2	+2
011	3	+3
100	4	−4
101	5	−3
110	6	−2
111	7	−1

5

Signed

1 +2 010
+ +1 001
+3 011 ✓

No carry.
No overflow.

2 −2 110
+ −1 111
−3 ~~1~~ 101 ✓

Carry!
No overflow.

Carry → unsigned

3 011
+ 4 100
7 111 ✓

No carry!

3 +3 011
+ +1 001
−4 100 ✗

No carry!
Overflow.

6 ~~1~~ 110
4 100
2 010 ✗

Carry!

4 −4 100
+ −1 111
+3 ~~1~~ 011 ✗

Carry
Overflow

6

CF

a b
1 2
cmpl b, a

a − b

1 − 2

1
~~2~~ 2
~~1~~ 001 (1)
− 010 (2)
1011

If carryout from MSB
or
carryin to MSB
} unsigned comparisons.

⇒ $\boxed{CF = 1}$

OF - signed numbers.

cmpl b, a a − b

$\boxed{t = a + b}$

if ((a > 0 && b > 0 && t < 0) ||
(a < 0 && b < 0 && t > 0))
⇒ OF = 1

−3
− −4
100

⇒ −3
+ (−4)

−3
−(−4)
+1

−3 + 4

3 → −3
101

~~X = 100~~
~~~X = 011~~
~~+1~~
~~100~~

X = 011
~X = 100
+1
101
~~~

2. Test

testl b, a ⟹ a & b

- update the %eflags

testl %eax, %eax

andl S, D

D ← D & S

10010101

& 10010101

(1)00101010

ZF = 0

SF = 1

if num in %eax is 0 : ZF = 1, SF = 0

+ve : ZF = 0, SF = 0

-ve : ZF = 0, SF = 1

Jump instructions

jmp target

instr 0
jmp Label
instr 1
instr 2
label:
instr 3
instr 4

Label in assembly

instr 0
instr 3
instr 4

top: Idiom
cmpl b, a
jne top

Jumps

unconditional jump.
(jmp)

conditional jumps

je
jne

jl
jle
jg
jge
signed comparisons.

jb
jbe
ja
jae
unsigned comparison

cmpl b, a

a - b

a : b

Loops ①

C

do-while

```
do {
    instr 1;
    2
    3
} while (expr);
```

while

```
init-expr;
while (expr) {

}
```

for

```
for (init-expr; condition; post-expr) {
    body of loop;
}
```

→

```
init-expr;
while (condition) {
    body of loop;
    post-expr;
}
```

goto

```
target:
    x = x + 1;

    goto target;
```

```
while (expr) {
    loop-body;
}
```

(10) →

```
if (!expr)
    goto end;
do {
    body of loop;
} while (expr);
end:
```

SET

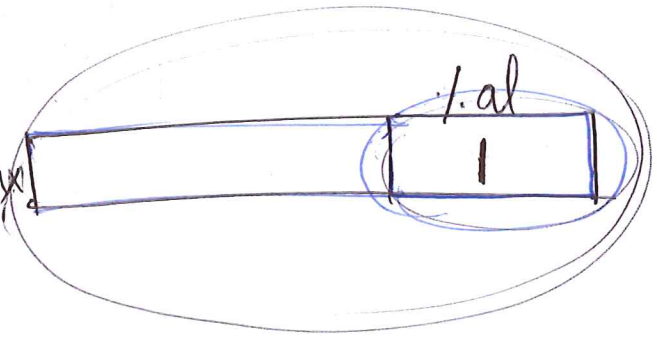
sete, setne, setl, setle, setg, setge

sets 1 byte to 0 or 1 based on the eflags.

setl (dst) → 1 byte

eg: %al

```
setl %al
movzbl %al, %eax
```



%eax
a

%ebx
b

cmpl b, a

a-b

a : b

Problem #6

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

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

```
    cmpl %ebx, %eax
    jg do
    jmp dont
do: addl $1, %eax
dont: instr
```

```
    cmpl %ebx, %eax
    jle dont
    addl $1, %eax
dont: instr
```

Problem #7

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

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

(D) (S)

```
    cmpl %ebx, %eax
    jle dont
    addl $1, %eax
    jmp end
dont: movl %eax, %ebx
end:
```

```
    cmpl $0, %ebx
    jle end
```

b : 0

Problem #8

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

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

```
loop: cmpl $0, %ebx
      jle end
      addl $1, %eax
      subl $1, %ebx
      jmp loop
end:
```

jmp comp

```
body: addl $1, %eax
      subl $1, %ebx
comp: cmpl $0, %ebx
      jg body
```

```
Condition codes: new bits in hidden %eflags register.

Some instructions set those bits based on comparisons:
  cmp, test
Other instructions change control flow (%eip) based on results:
  jmp family
```

```
INSTRUCTION: cmpl B, A

  computes A-B (but doesn't put result anywhere)

condition codes (incomplete):
  zero flag   : ZF=1 if (A-B) == 0  otherwise ZF=0
  signed flag : SF=1 if (A-B) < 0   otherwise SF=0
```

```
INSTRUCTION: jmp TARGET    always changes %eip to TARGET

INSTRUCTION: je  TARGET    %eip=TARGET if ZF==1

INSTRUCTION: jne TARGET    %eip=TARGET if ZF== 0

INSTRUCTION: jg  TARGET    %eip=TARGET if ~(SF∧OF) & ~ZF

INSTRUCTION: jge TARGET    %eip=TARGET if ~(SF∧OF)

INSTRUCTION: jl  TARGET    %eip=TARGET if SF=1 ~~AND ZF=0~~

INSTRUCTION: jle TARGET    %eip=TARGET if SF=1 OR ZF=1
```

SF ∧ OF

(SF∧OF) | ZF

cmpl b, a

a-b

CF ZF SF OF

OF = 1
SF = 0

-4	100
-3	011
	001

a < b

1	2
-4	3

	001
	-010
	111

SF = 1
OF = 0