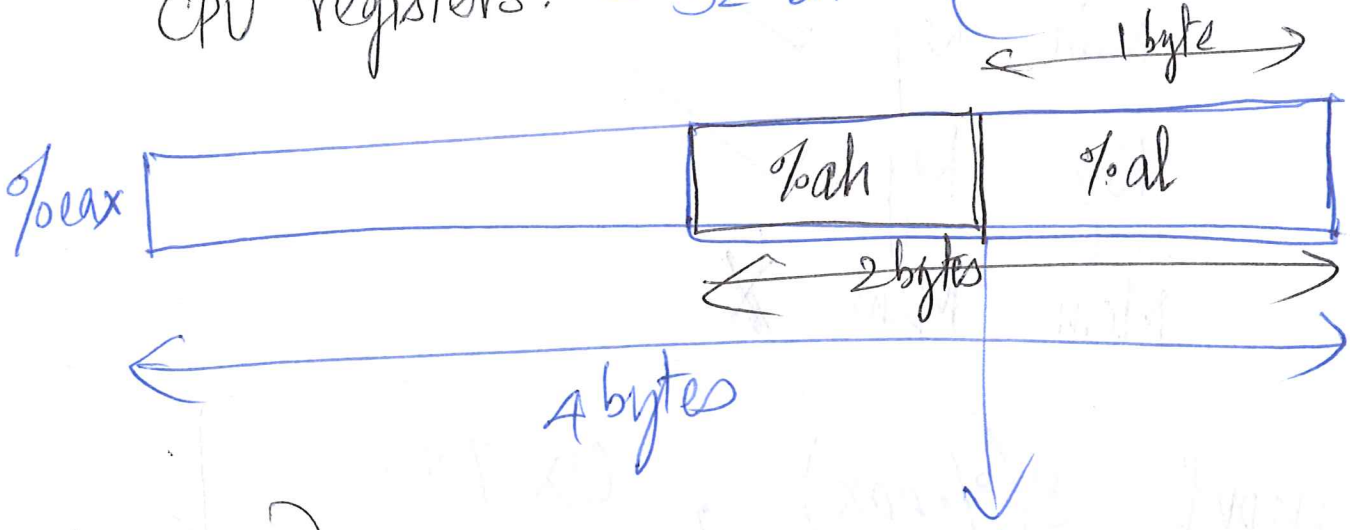
CS 354 - Lecture 7

Review

C → (1) → Assembly.

CPU registers. - 32-bits (32 bits machines)



%eax

%ebx

%ecx

%edx

%esi

%edi

} general purpose registers

X86

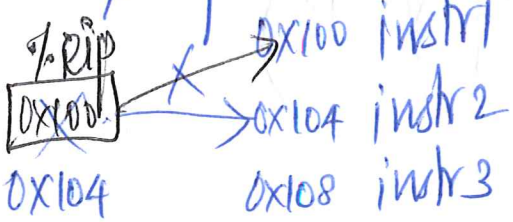
AT&T

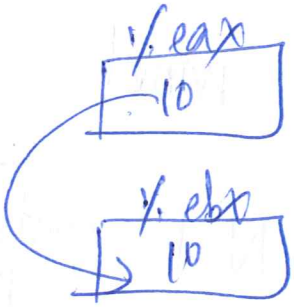
movl %eax, %ebx

S D

Special purpose registers

%eip - instruction pointer





(2)

mov S, D

Reg Mem ✓

Mem Reg ✓

Reg Reg ✓

Mem Mem ✗

movl 8(%eax) , 0x7008

%eax
0x3000

0x3000
+ 8
0x3008

Memory

movl 0x1000, %eax

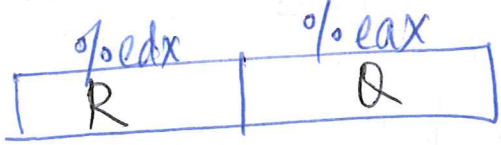
movl N(R1, R2, K), %eax

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

Arithmetic

addl S, D $D \leftarrow D + S$

subl, imull, idivl



Bitwise

andl S, D $D \leftarrow D \,\&\, S$

orl, notl, xorl

Unary

incl D $D \leftarrow D + 1$

decl, negl, notl

Shift

sarl K, D $D \leftarrow D \gg_A K$

shrl

sall / shll

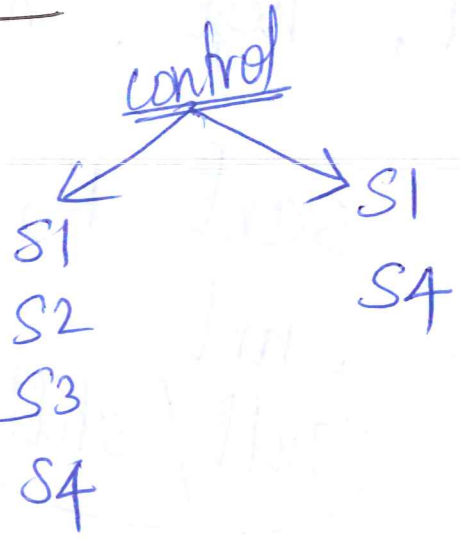
Today

1. Control flow
   - if / else
   - loops
   - functions.
2. Stack pointer (%esp)
3. Load Effective Address (leal).

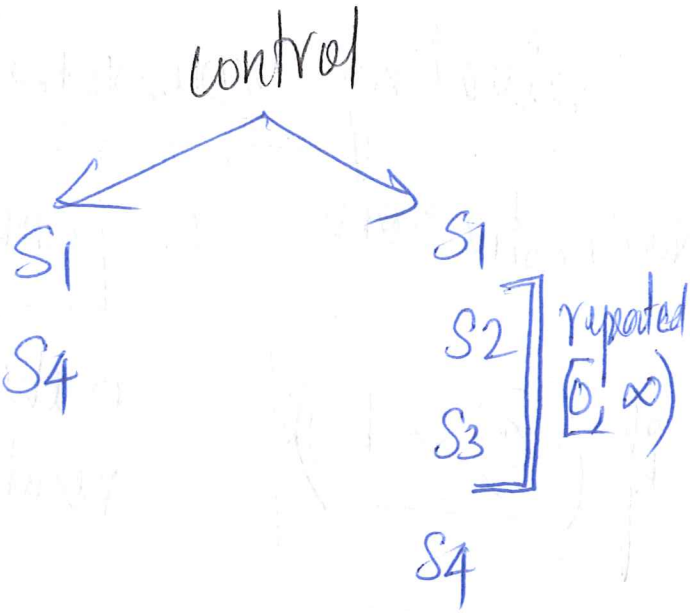
Control Flow

Decisions

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



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



What help do we need from Assembly to implement control flow?

1) need some mechanism to evaluate an expression. (eg boolean expr)

2). Ability to change the instruction pointer (program counter) to jump to some arbitrary instruction.

Evaluating expressions

New Instructions

1. `cmpl b, a`

evaluate : a − b

result of this subtraction is discarded.

if (a > b) {

}

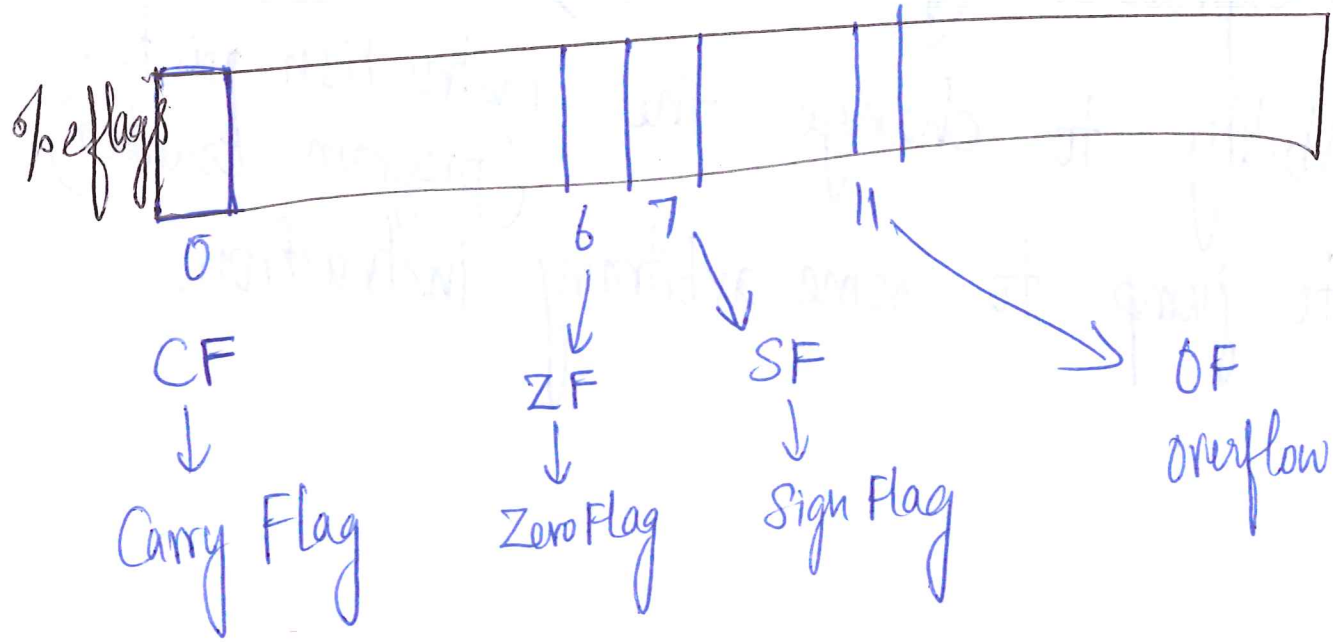
→ update a special register %eflags (hidden register)

(or)

Condition Code registers

(or)

PSW - Program Status Word.



Addition:

$$t = a + b$$

%eflags register are updated implicitly.

vs

%eflags gets updated explicitly w/ cmpl instr.

int → 3 bits

1) ZF - Zero Flag

if $(t == 0) \Rightarrow ZF = 1$ (set)

else $\Rightarrow ZF = 0$ (unset).

eg. $t = 3 + (-3) \Rightarrow$ ZF = 1

2) CF - Carry Flag. (unsigned numbers)

000 - 0
001
010
011
100
101
110
111 - 7

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

No carry

CF = 0

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

CF = 1

3) SF - Sign Flag (~~unsigned~~ signed numbers)

$t = a + b$

if $(t < 0) \Rightarrow SF = 1$

else $\Rightarrow SF = 0$.

eg $t = 3 + (-4) = -1 \Rightarrow$ SF = 1

1 1 1

4) OF - Overflow flag

$t = a + b$

① 2 010
+1 001
+3 011

② 3 011
+1 001
-4 100

-4 -3 -2 -1 0 +1 +2 +3

Overflow

if $((a > 0 \;\&\&\; b > 0 \;\&\&\; t < 0)$
$\|\; (a < 0 \;\&\&\; b < 0 \;\&\&\; t > 0))$

$\Rightarrow$ OF = 1

(9)

cmpl b, a

a - b

1) ZF: $a - b == 0 \Rightarrow ZF = 1$

2) CF: $a - b < 0 \Rightarrow CF = 1$

(or)

$a < b$

a: 1 ⇒ 001

-b: 2 ⇒ 010

(7) ⇒ 111

3) SF: $a - b < 0 \Rightarrow SF = 1$

~~t = a - b~~ (a - b)

4) OF:

if ((a > 0 && b < 0 && (a-b) < 0)

|| (a < 0 && b > 0 && (a-b) > 0))

⇒ OF = 1

New instruction:

testl b, a

evaluate: a & b

→ expliatly sets the % eflags.

1) ZF: a & b == 0

2) SF: a & b < 0

testl %eax, %eax

1) if %eax = 0 : ZF = 1, SF = 0 } ⇒ value in %eax = 0

2) if %eax < 0 : ZF = 0, SF = 1 } ⇒ value in %eax < 0.

eg. 111
&111
1111

3) if %eax > 0 : 001
&001
001

ZF = 0, SF = 0 } ⇒ value in %eax > 0

Jump instructions


```
        instr 1
        jmp  target
        instr 2
        instr 3
target: instr 4
        instr 5
```

```
        instr 1
        jz  target
        instr 2
        instr 3
target: instr 4
        instr 5
```

cmpl b, a

Jumps

- unconditional: jmp <target>
- conditional
  - je (jz)
  - jne (jnz)
  - signed numbers: jl, jle, jg, jge
  - jb, jbe, ja, jae

```
if ( a < b ) {
}
```

%eax %ebx

```
cmpl %ebx, %eax
jl Label
```

a - b

Instruction	Jump condition
jb	CF == 1
jbe	CF == 1 or ZF == 1
ja	CF == 0 and ZF == 0
jae	CF == 0
jl	SF ∧ OF

cmpl b, a → a − b

CF

SF == 1 ∧ OF == 0 = 1

SF == 0 ∧ OF == 1 = 1

cmpl b, a

a = 1 ; b = 2

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

a = -4 b = 3

```
  -4 : 100
-  3 : 011
-------
   001
```

De Morgan's Law

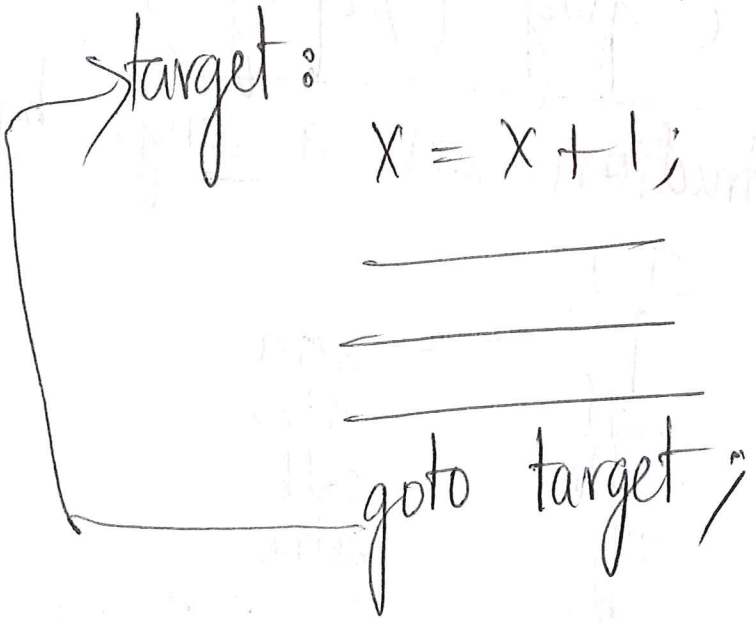
$$\sim(A \mid B) = \sim A \;\&\; \sim B$$

Loops

for while do-while

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

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



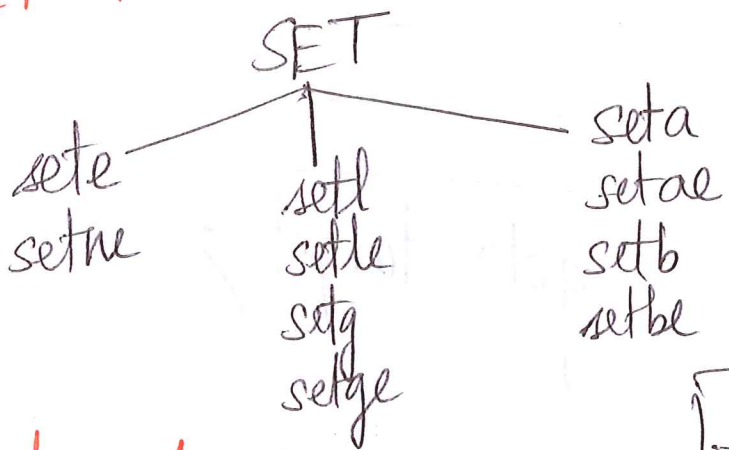
(14)

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

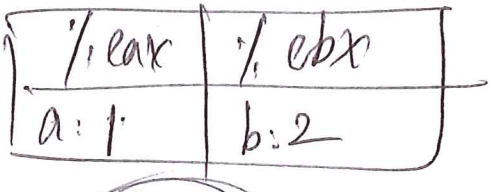
```
if (!expr)
    goto end;
do {
    body of loop;
} while (expr);
end:
```

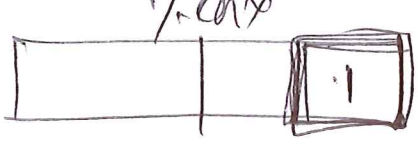
Access the CC flags (%eflags)

SET instruction - sets a single byte to 0 or 1.

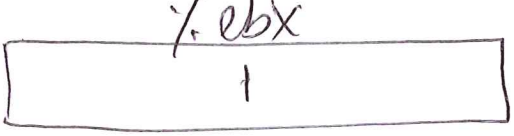



```
cmpl %ebx, %eax
setl %al
movzbl %al, %ebx
```

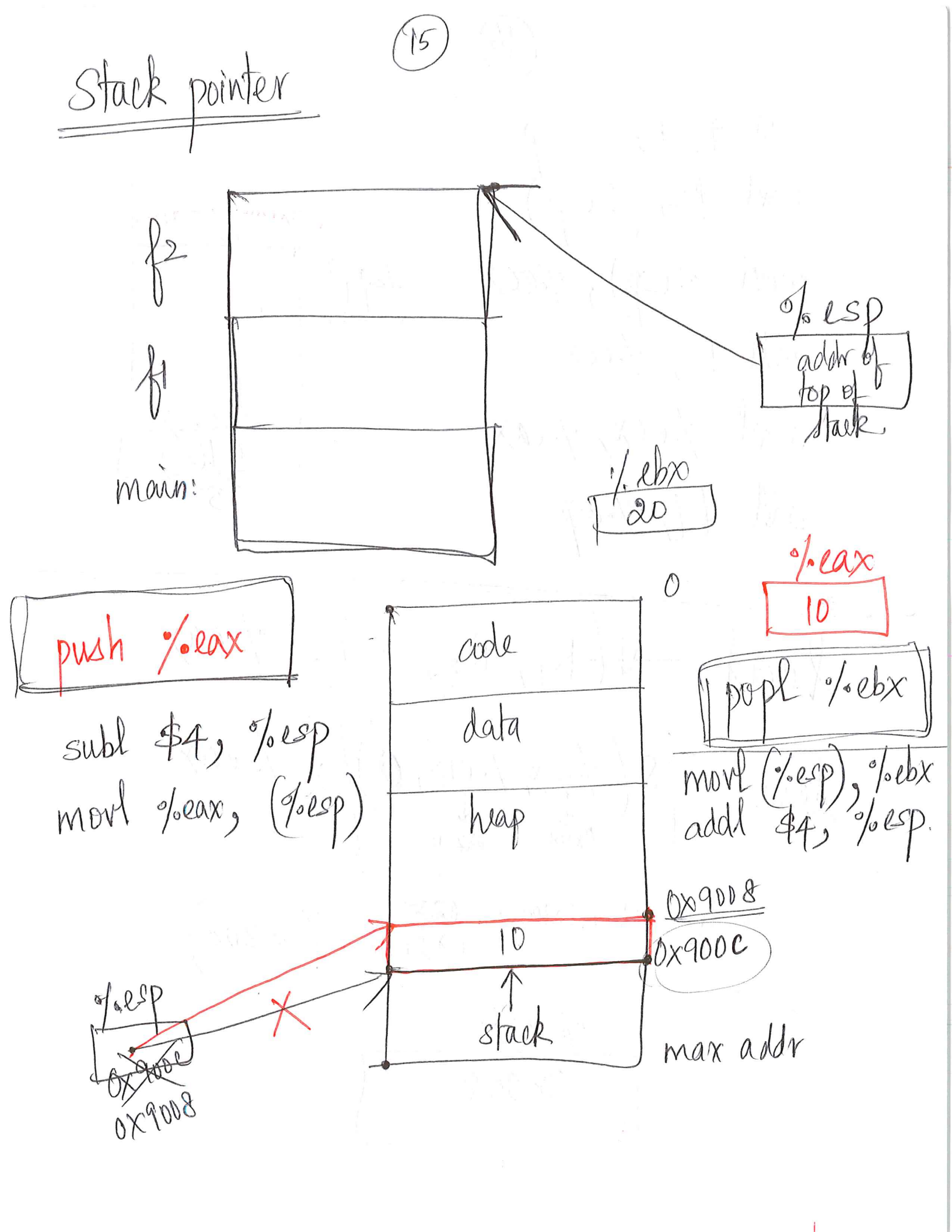




a - b

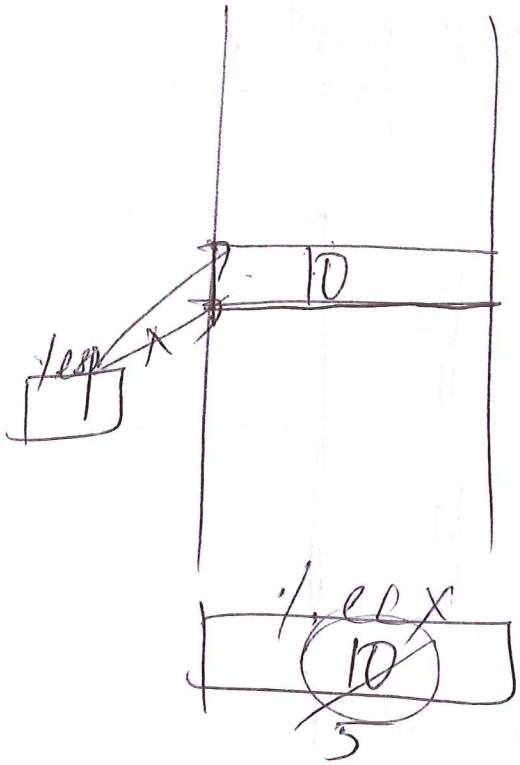


Stack pointer



(16)

```
add $-4, %.esp
movl $10, (%.esp)
movl (%.esp), %.ecx
addl $5, %.ecx
movl %.ecx, %.eax
add $4, %.esp
```



---

leal N(R1, R2, K), %.eax

leal 8(%.eax, %.ebx, 0), %.eax

0X100 0X200

addr: 8 + 0x100 + 0X200 = 0X308

%.eax

0X308

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      (a-b)
jg do
jmp dont
do: addl $1, %eax
dont: instr.
```

```
cmpl %ebx, %eax      (a-b)
jle dont
addl $, %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;
}
```

```
cmpl %ebx, %eax      (a-b)
jle dont
addl $1, %eax
jmp end
dont: movl %eax, %ebx
end: instr after if/else
```

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        (b-0)
jle end
incl %eax
decl %ebx
jmp loop
end:
instr(s)
```

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== _____

INSTRUCTION: **jg** TARGET %eip=TARGET if _____

INSTRUCTION: **jge** TARGET %eip=TARGET if _____

INSTRUCTION: **jl** TARGET %eip=TARGET if _____

INSTRUCTION: **jle** TARGET %eip=TARGET if _____

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++;
}
```

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;
}
```

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--;
}
```

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 $\neg(SF \wedge OF) \;\&\; \neg ZF$

INSTRUCTION: **jge** TARGET %eip=TARGET if $\neg(SF \wedge OF)$

INSTRUCTION: **jl** TARGET %eip=TARGET if $SF \wedge OF$

INSTRUCTION: **jle** TARGET %eip=TARGET if $(SF \wedge OF) \mid ZF$