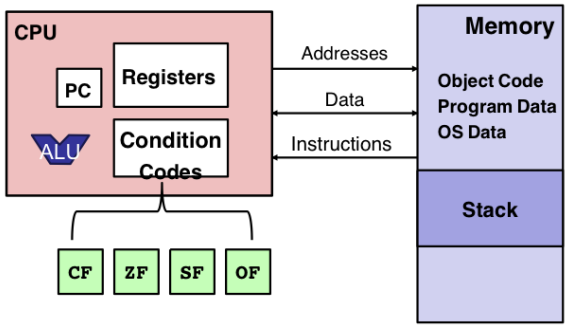
conditions in Assembly

uses condition codes & jump



single bit registers

flags

carry flag	CF	set if unsigned overflow
zero flag	ZF	all bits are zero
sign flag	SF	negative integer
overflow flag	OF	2's compl. signed overflow

assigned w/ different interpretations

not set by leaq

add

a	b	result t = a+b
1111 1111	1111 1111	1111 1111 + 1111 1111 = 1 1111 1110
1000 0000	1000 0000	1000 0000 + 1000 0000 = 1 0000 0000
0111 1111	0111 1111	0111 1111 + 0111 1111 = 1111 1110

condition codes

CF	ZF	SF	OF	
1	0	1	0	
				there's a carrier bit
				is bit all 0?
				most sig bit is 1
				a=-1, b=-1 t=-2
1	1	0	1	
				all 0's
				non negative
				negative + negative = 0
0	0	1	1	
				is negative
				pos + pos = neg.

explicitly by testq Src1, Src2

testq b,a , like a&b w/o destination

sets condition based on value of Src1 Src2

test sign of value

useful to have 1 of operands be mask

ZF set when a&b == 0

SF set when a&b < 0

OF & CF set to 0

Compare cmpq Src1, Src2

cmpq b,a like a - b

CF	set	if	carry out from most sig. bit
ZF	set	if	a == b
SF	set	if	(a-b) < 0
OF	set	if	2's compl. overflow

add

a: 1011 0011

b: 1011 0011

cmpq b,a a - b: 0000 0000

1111 1110 (-2)

1111 1111 (-1)

1111 1110 − 1111 1111 = 1 1111 1111

condition codes: CF 0, ZF 1, SF 0, OF 0

1000 0000

0111 1111

1000 0000 − 0111 1111 = 0000 0001

condition codes: CF 0, ZF 0, SF 0, OF 1

neg - pos = pos!?!

is a < b?

iff SF ^ OF

sign flag or overflow, not both

SetX Instructions

Does not alter remaining 7 bytes

SetX	Condition	Description
setl	(SF^OF)	Less (Signed)

```
long less (long x, long y)
{
  return x < y;
}
```

```
cmpq    %rsi,%rdi     # Compare x:y
setl    %al           # Set when <
movzbq  %al,%rax      # Zero rest of %rax
ret
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

x = 2, y = 3

SF = ? OF = ?

000...001

%rax %al

cmpq %rsi, %rdi x - y

SF: 1 negative

OF: 0 no overflow

```
  0010
- 0011
-------
  1111
```

setl %al SF ^ OF XOR

put result @ lowest %rax register 1, %al

movzbq %al, %rax set rest to 0

SetX Instructions

Does not alter remaining 7 bytes

SetX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF)&~ZF	Greater (Signed)
setge	~(SF^OF)	Greater or Equal (Signed)
setl	(SF^OF)	Less (Signed)
setle	(SF^OF)|ZF	Less or Equal (Signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	

a > b

Conditional Branches

jump

jX	Condition	Description
jl	(SF^OF)	

```
long absdiff
  (long x, long y)
{
  long result;
  if (x >= y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
absdiff:
    cmpq    %rsi, %rdi  # x:y
    jl      .L4         # jmp if x<y
    movq    %rdi, %rax
    subq    %rsi, %rax
    ret
.L4:                    # x < y
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

compare x:y

jl: jump if less

last compare instruction

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

ISC 15400

jX Instructions

Jump to different part of code depending on condition codes

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF)&~ZF	Greater (Signed)
jge	~(SF^OF)	Greater or Equal (Signed)
jl	(SF^OF)	Less (Signed)
jle	(SF^OF)|ZF	Less or Equal (Signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	

Think of C's "go to" statement

label)

```
long absdiff(long x, long y)
{
    long result;
    if (x >= y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
long absdiff_j(long x, long y)
{
    long result;
    int ntest = (x < y);
    if (ntest) goto Else;
    result = x-y;
    goto Done;
 Else:
    result = y-x;
 Done:
    return result;
}
```

```
absdiff:
    cmpq    %rsi, %rdi  # x:y
    jl      .L4         # jmp if x<y
    movq    %rdi, %rax #x -> rax
    subq    %rsi, %rax # rax-rsi ->rax
    ret
.L4:                    # x < y
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

%rdi = x
%rsi = y

"jX" of assembly equivalent to "goto" in C

C Code

```
val = Test ? Then_Expr : Else_Expr;
```

```
val = x>y ? x-y : y-x;
```

Goto Version

```
    ntest = !Test;
    if (ntest) goto Else;
    val = Then_Expr;
    goto Done;
Else:
    val = Else_Expr;
Done:
    . . .
```

Create separate code regions for then & else expressions

Execute appropriate one

Loops

easily translated: "do-while" loops turn to goto version

C Code

```
long pcount_do
  (unsigned long x) {
  long result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

Goto Version

```
long pcount_goto
  (unsigned long x) {
  long result = 0;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

Count number of 1's in argument x

Use conditional branch to either continue looping or to exit loop

Goto Version

```
long pcount_goto
  (unsigned long x) {
  long result = 0;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

Register	Use(s)
%rdi	Argument x
%rax	result

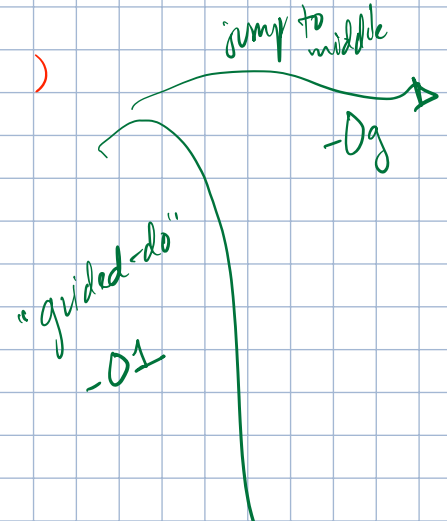
```
   movq    $0, %rax        #  result = 0
.L2:                       # loop:
   movq    %rdi, %rdx
   andq    $1, %rdx        #  t = x & 0x1
   addq    %rdx, %rax      #  result += t
   shrq    %rdi            #  x >>= 1
   jne     .L2             #  if (x) goto loop
   ret
```

Jump if ZF is not set to 1

do
  Body
  while (Test)

loop:
  Body
  if(Test) goto loop

while(Test)
  Body



goto test
loop:
  Body
test:
  if(Test) goto loop
done:

if(!Test) goto done

loop:
Body
if(Test) goto loop

done:

Example

C Code

```
long pcount_while
  (unsigned long x) {
  long result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Goto Version

```
long pcount_goto_jtm
  (unsigned long x) {
  long result = 0;
  goto test;
 loop:
  result += x & 0x1;
  x >>= 1;
 test:
  if(x) goto loop;
  return result;
}
```

Jump-to-middle

```
long pcount_goto_dw
  (unsigned long x) {
  long result = 0;
  if (!x) goto done;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
 done:
  return result;
}
```

Guided-do

for(init ; Test ; update)

Body

while loop

init
while(Test)
Body
update

goto

"For" Loop Form

General Form

```
for (Init; Test; Update)
    Body
```

Init `i = 0` **Test** `i < 64` **Update** `i++`

Body

```
unsigned bit =
    (x >> i) & 0x1;
result += bit;
```

While Version

```
Init;
while (Test) {
    Body
    Update;
}
```

```
long pcount_while (unsigned long x)
{
  size_t i;
  long result = 0;
  i = 0;
  while ( i < 64)
  {
    unsigned bit = (x >> 1) & 0x1;
    result += bit;
    i++;
  }
  return result;
}
```

C Code

```
long pcount_for
  (unsigned long x)
{
  size_t i;
  long result = 0;
  for (i = 0; i < 64; i++)
  {
    unsigned bit =
      (x >> 1) & 0x1;
    result += bit;
  }
  return result;
}
```

Initial test can be optimized away

Goto Version

```
long pcount_for_goto_dw
  (unsigned long x) {
  size_t i;
  long result = 0;
  i = 0;
  if (!(i < 64))
    goto done;
 loop:
  {
    unsigned bit =
      (x >> 1) & 0x1;
    result += bit;
  }
  i++;
  if (i )
    goto loop;
 done:
  return result;
}
```

Init

! Test

Body

Update

Test