Get to know th
condition codes

Understand how computer control program flow

Understand looping and branching

TRUONG DAI HOC KHOA HOC TU NHIEN
KHTN
TP. HO CHI MINH

x86 Transfer Control

① Conditional Codes

② Jumping

Three Basic Kinds of Instructions

- Transfer data
  - MOV, LEA
- Arithmetic function
  - ADD, SUB, IMUL, SAL, SAR, SHR, XOR, AND, OR
  - INC, DEC, NEG, NOT
- Transfer control
  - JMP, JE, JNE, JS, JNS, JG, JGE, JL, JLE, JA, JB

Condition Codes

Implicitly set by arithmetic or logical operations (~~LEA~~)

CF	Carry Flag	Carry out of the MSB unsigned
ZF	Zero Flag	0
SF	Sign Flag	Negative value signed
OF	Overflow Flag	Two’s complement overflow signed

Condition Codes Examples

t = a + b

CF	`(unsigned) t < (unsigned) a`
ZF	`t == 0`
SF	`t < 0`

OF	`(a>0 && b>0 && t<0) ||`
	`(a<0 && b<0 && t>=0)`

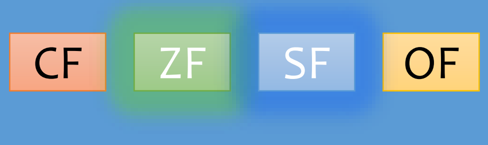
Set Condition Codes

CMP S1, S2

cmpb cmpw cmpl cmpq

Sets condition codes based on S2 – S1

```
cmpq %rax, %rbx
```

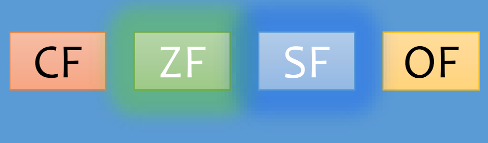


TEST S1, S2

testb testw testl testq

Sets condition codes based on S2 & S1

```
testq %rax, %rax
```



Accessing the Condition Codes

1. Set a byte to 0 or 1

2. Conditionally jump to other program part

3. Conditionally transfer data

The SET Instructions

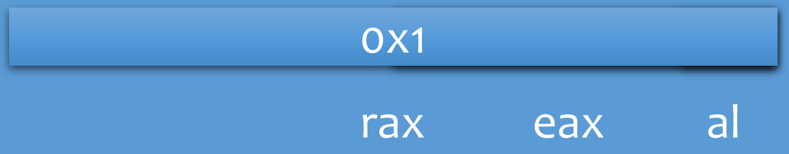
1. Set a byte to 0 or 1

SET D

sete setne sets setns
setg setge setl setle
seta setae setb setbe

```
comp:
   cmpq  %rsi, %rdi
   setl  %al
   movzbl %al, %eax
   ret
```

a < b



0x1

rax eax al

SET Instructions

Instruction	Synonym	Condition	Description
sete	setz	ZF	Equal / Zero
setne	setnz	~ZF	Not Equal / Not Zero
sets		SF	Negative
setns		~SF	Nonnegative
setg	setnle	~(SF^OF)&~ZF	Greater (signed)
setge	setnl	~(SF^OF)	Greater or Equal
setl	setnge	(SF^OF)	Less (signed)
setle	setng	(SF^OF)|ZF	Less or Equal
seta	setnbe	~CF&~ZF	Above (unsigned)
setae	setnb	~CF	Above or Equal
setb	setnae	CF	Below (unsigned)
setbe	setna	CF | ZF	Below or Equal

Accessing the Condition Codes

2. Conditionally jump to other program part

J Label

je	jne	js	jns
jg	jge	jl	jle
ja	jae	jb	jbe

jmp label

jmp *Operand

Unconditional jumps

Conditional Jumps

Instruction	Synonym	Condition	Description
je	jz	ZF	Equal / Zero
jne	jnz	~ZF	Not Equal / Not Zero
js		SF	Negative
jns		~SF	Nonnegative
jg	jnle	~(SF^OF)&~ZF	Greater (signed)
jge	jnl	~(SF^OF)	Greater or Equal
jl	jnge	(SF^OF)	Less (signed)
jle	jng	(SF^OF)|ZF	Less or Equal
ja	jnbe	~CF&~ZF	Above (unsigned)
jae	jnb	~CF	Above or Equal
jb	jnae	CF	Below (unsigned)
jbe	jna	CF | ZF	Below or Equal

Jump Instruction Example

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

x in %rdi
y in %rsi

```
absdiff:
   cmpq    %rsi, %rdi
   jle     .L4
   movq    %rdi, %rax
   subq    %rsi, %rax
   ret
.L4:
   movq    %rsi, %rax
   subq    %rdi, %rax
   ret
```

Accessing the Condition Codes

3. Conditionally transfer data

```
CMOV  S, R

      cmove   cmovne   cmovs   cmovns
      cmovg   cmovge   cmovl   cmovle
      cmova   cmovae   cmovb   cmovbe
```

not require control transfer

Conditional Move

Instruction	Synonym	Condition	Description
cmove	cmovz	ZF	Equal / Zero
cmovne	cmovnz	~ZF	Not Equal / Not Zero
cmovs		SF	Negative
cmovns		~SF	Nonnegative
cmovg	cmovnle	~(SF^OF)&~ZF	Greater (signed)
cmovge	cmovnl	~(SF^OF)	Greater or Equal
cmovl	cmovnge	(SF^OF)	Less (signed)
cmovle	cmovng	(SF^OF)|ZF	Less or Equal
cmova	cmovnbe	~CF&~ZF	Above (unsigned)
cmovae	cmovnb	~CF	Above or Equal
cmovb	cmovnae	CF	Below (unsigned)
cmovbe	cmovna	CF | ZF	Below or Equal

CMOV Instruction Example

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

x in %rdi
y in %rsi

```
absdiff:
   movq    %rdi, %rax
   subq    %rsi, %rax
   movq    %rsi, %rdx
   subq    %rdi, %rdx
   cmpq    %rsi, %rdi
   cmovle  %rdx, %rax
   ret
```

Do-While Loops

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

x in %rdi

```
pcount_do:
    movl    $0, %eax
.L2:
   movq    %rdi, %rdx
   andl    $1, %edx
   addq    %rdx, %rax
   shrq    %rdi
   jne     .L2
   rep; ret
```

Do-While Loops

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

x in %rdi

```
pcount_do:
    movl     $0, %eax
.L2:
   movq     %rdi, %rdx
   andl     $1, %edx
   addq     %rdx, %rax
   shrq     %rdi
   jne       .L2
   rep; ret
```

“Do-While” Translation

```
do
  Body
  while (Test);
```

```
loop:
  Body
  if (Test)
    goto loop
```

```
while (Test)
  Body;
```

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

While Loop Example 1

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

```
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;
}
```

“While” Translation

```
while (Test)
  Body
```

```
  if (!Test)
    goto done;
  do
    Body
    while(Test);
done:
```

```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

While Loop Example 2

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

```
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;
}
```

For Loops

```
#define WSIZE 8*sizeof(int)
long pcount_for (unsigned long x)
{
  size_t i;
  long result = 0;
  for (i = 0; i < WSIZE; i++)
  {
    unsigned bit = (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```

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

```
void switch_eg
(long x, long n, long *dest)
{
    long val = x;
    switch(n) {
    case 100:
        val *= 13;          Block 0
        break;
    case 102:
        val += 10;          Block 1
        /* Fall Through */
    case 103:
        val += 11;          Block 2
        break;
    case 104:
    case 106:
        val *= val;         Block 3
        break;
    default:
        val = 0;            Block 4
    }
    *dest = val;
}
```

```
switch_eg:
        subq    $100,%rsi
        cmpq    $6,%rsi
        ja      .L8
        jmp     *.L4(,%rsi,8)
.L3:
        leaq    (%rdi,%rdi,2),%rax
100     leaq    (%rdi,%rax,4),%rdi      Block 0
        jmp     .L2
.L5:
102     addq    $10,%rdi                Block 1
.L6:
        addq    $11,%rdi
103                                     Block 2
        jmp     .L2
.L7:
104     imulq   %rdi,%rdi
                                        Block 3
106     jmp     .L2
.L8:
        movl    $0,%edi                 Block 4
.L2:
        movq    %rdi,(%rdx)
        ret
```

Jump table

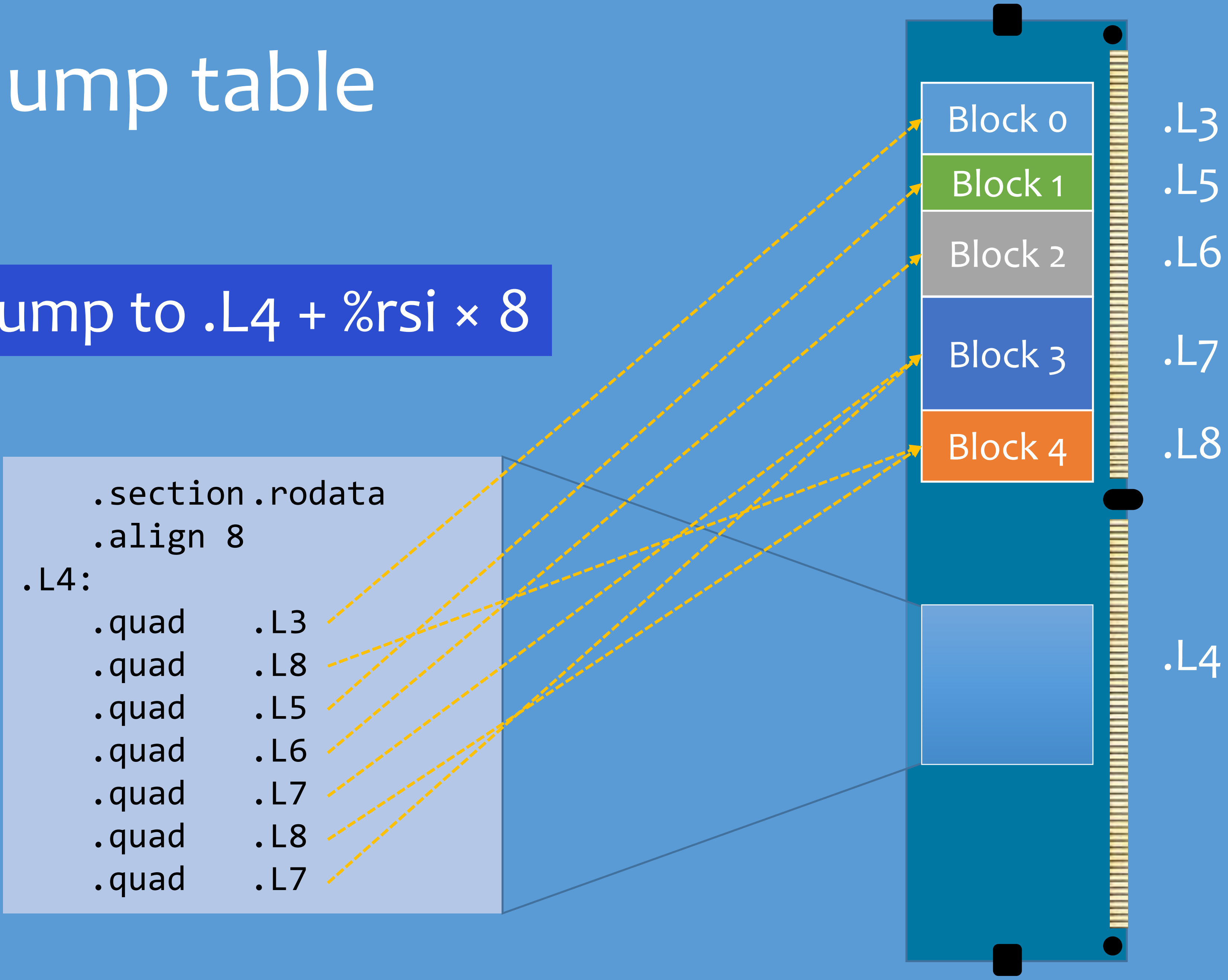
Jump to .L4 + %rsi × 8

```
    .section .rodata
    .align 8
.L4:
    .quad    .L3
    .quad    .L8
    .quad    .L5
    .quad    .L6
    .quad    .L7
    .quad    .L8
    .quad    .L7
```

```
switch_eg:
        subq     $100,%rsi
        cmpq     $6,%rsi
        ja       .L8
        jmp      *.L4(,%rsi,8)
.L3:
        leaq     (%rdi,%rdi,2),%rax
100     leaq     (%rdi,%rax,4),%rdi      Block 0
        jmp      .L2
.L5:
102     addq     $10,%rdi                Block 1
.L6:
        addq     $11,%rdi
103                                      Block 2
        jmp      .L2
.L7:
104     imulq    %rdi,%rdi
                                         Block 3
106     jmp      .L2
.L8:
        movl     $0,%edi                 Block 4
.L2:
        movq     %rdi,(%rdx)
        ret
```

Jump table

Jump to .L4 + %rsi × 8



```
    .section .rodata
    .align 8
.L4:
    .quad   .L3
    .quad   .L8
    .quad   .L5
    .quad   .L6
    .quad   .L7
    .quad   .L8
    .quad   .L7
```

Summary

- Transfer control
  - JMP
  - JE, JNE, JS, JNS, JG, JGE, JL, JLE, JA, JB
- Condition Codes
  - CF, ZF, SF, OF
  - CMP, TEST, SET
  - CMOV

Charles Petzold

American programmer, Microsoft MVP

> “Programming in machine code is like eating with a toothpick.”