Assembly

תרגול 5
תכנות באסמבלי, המשך

Condition Codes

- Single bit registers
  - ZF – zero flag
  - SF – sign flag
  - CF – carry flag
  - OF – overflow flag

- Relevant only for the most recent operation
- `leaq` does not alter any condition code
- In logical operations, CF and OF are set to 0
- For shift operations, OF is set to 0, CF is the last shifted out bit

ZF – Zero Flag

- Set if a result is zero
- Example:
  - Addb 0,0 $\rightarrow ZF = 1$

SF – Sign Flag

- Set if a result is negative (MSB of the result = 1)
- Example:
  - subb $00000000_b, 00000001_b \rightarrow 11111111_b \rightarrow SF = 1$

CF – Carry Flag

- Used for unsigned arithmetic
- CF gets 1 if result is larger than capacity of the target operand
- Examples:
  - $11111111_b + 00000001_b = 1\underbrace{00000000}_{0*8}{}_b = 0 \rightarrow CF = 1$
  - $00000000_b - 00000001_b = \underbrace{11111111}_{1*8}{}_b \rightarrow CF = 1$

unsigned arithm $(decimal)$ =
255 (decimal) → Got the wrong answer

00000000_b-$00000001_b \equiv$
100000000_b-$00000001_b \equiv 11111111_b$

signed arithmetic → $11111111_b = -1(decimal) \rightarrow$ ignore CF flag

OF – Overflow Flag

- Used for signed arithmetic
- sign-bit-off operands + sign-bit-on result and vice versa → OF gets '1'
- Examples:
  - $01111111_b + 01000000_b = 1\underbrace{0000000}_{0*7}{}_b \rightarrow OF = 1$
  - $10000000_b + 1000000_b = 1\underbrace{00000000_b}_{1*8} = 0 \rightarrow OF = 1$
- Otherwise OF gets '0'

Usage examples

- Check equality of two values:
  - subl %ecx, %edx and ZF = 1

But then, the subtraction result we be saved in %edx

Setting Condition Codes

Instruction		Based on	Description
cmpb	S_2, S_1	S_1 - S_2	Compare bytes
testb	S_2, S_1	S_1 & S_2	Test byte
cmpw	S_2, S_1	S_1 - S_2	Compare words
testw	S_2, S_1	S_1 & S_2	Test word
cmpl	S_2, S_1	S_1 - S_2	Compare double words
testl	S_2, S_1	S_1 & S_2	Test double word

- Similarly, cmpq and testq
- Incase we do not need to save the result

Test usage examples

- Check if a number is ‘0’:
  - testl %ecx, %ecx and check if ZF = 1

Setting Condition Codes (cont.)

Explicit Setting by Test instruction

`testl` *Src2*,*Src1*

- Sets condition codes based on value of *Src1* `&` *Src2*
  - Useful to have one of the operands be a mask
- `testl b,a` like computing `a&b` without setting destination
- ZF set when `a&b == 0`
- SF set when `a&b < 0`

Accessing the Condition Codes

Instruction		Synonym	Effect	Set Condition
sete	*D*	setz	*D* ← ZF	Equal / Zero
setne	*D*	setnz	*D* ← ~ZF	Not Equal / Not Zero
sets	*D*		*D* ← SF	Negative
setns	*D*		*D* ← ~SF	Nonnegative
setg	*D*	setnle	*D* ← ~(SF ^ OF) & ~ZF	Greater (Signed >)
setge	*D*	setnl	*D* ← ~(SF ^ OF)	Greater or Equal (Signed >=)
setl	*D*	setnge	*D* ← SF ^ OF	Less (Signed <)
setle	*D*	setng	*D* ← (SF ^ OF) | ZF	Less or Equal (Signed <=)
seta	*D*	setnbe	*D* ← ~CF & ~ZF	Above (Unsigned >)
setae	*D*	setnb	*D* ← ~CF	Above or Equal (Unsigned >=)
setb	*D*	setnae	*D* ← CF	Below (Unsigned <)
setbe	*D*	setna	*D* ← CF & ~ZF	Below or Equal (Unsigned <=)

Figure 3.9: **The** `set` **Instructions.** Each instruction sets a single byte to 0 or 1 based on some combination of the condition codes. Some instructions have "synonyms," i.e., alternate names for the same machine instruction.

Why does it work?

Let's take `cmpl b,a` for example

- If there is no overflow and a≥b → SF=0, OF=0
- If there is no overflow and a<b → SF=1, OF=0
- If there is a negative overflow (a>b) → SF=1, OF=1
- If there is a positive overflow (a<b) → SF=0, OF=1

a < b in assembly

- Translate the line: return (a<b);
- Suppose a is in %edx, b is in %eax:

```
cmpl %eax,%edx          # compare a to b
```

setl	D	setnge	$D \leftarrow$ SF ^ OF	Less (Signed <)

```
movzbl %al,%eax         # set %eax to 0 or 1
```

Unconditional Jump

- Direct jump

```
jmp L1
```

- Indirect jump
  - `jmp *%rax`
  - `jmp *(%rax)`

```
L1:
   movq %rdx, %rcx
   movq %rdx, %rcx
   jmp L1
```

Unconditional Jump

- Direct jump

  `jmp L1`

- Indirect jump
  - `jmp *%rax`
  - `jmp *(%rax)`

```
%rax = L2

 L1:
    movq %rdx, %rcx
 L2:
    movq %rdx, %rcx
    jmp *%rax
```

Unconditional Jump

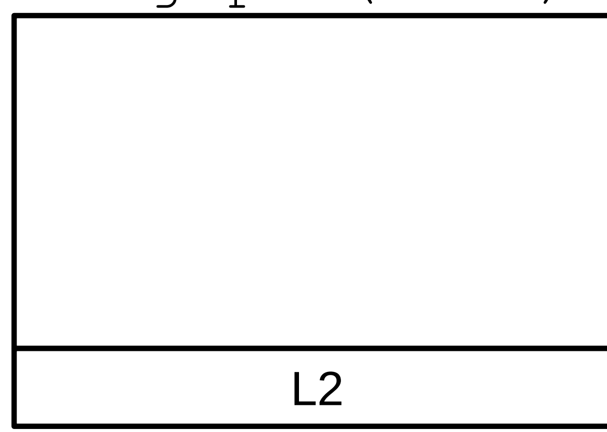
- Direct jump

  `jmp L1`

- Indirect jump
  - `jmp *%rax`
  - `jmp *(%rax)`

```
%rax = 1000

 L1:
   movq %rdx, %rcx
 L2:
   movq %rdx, %rcx
   jmp *(%rax)
```



1000

Jump Instructions

Instruction	Synonym	Jump Condition	Description
`jmp` *Label*		1	Direct Jump
`jmp` **Operand*		1	Indirect Jump
`je` *Label*	`jz`	`ZF`	Equal / Zero
`jne` *Label*	`jnz`	`~ZF`	Not Equal / Not Zero
`js` *Label*		`SF`	Negative
`jns` *Label*		`~SF`	Nonnegative
`jg` *Label*	`jnle`	`~(SF ^ OF) & ~ZF`	Greater (Signed >)
`jge` *Label*	`jnl`	`~(SF ^ OF)`	Greater or Equal (Signed >=)
`jl` *Label*	`jnge`	`SF ^ OF`	Less (Signed <)
`jle` *Label*	`jng`	`(SF ^ OF) | ZF`	Less or Equal (Signed <=)
`ja` *Label*	`jnbe`	`~CF & ~ZF`	Above (Unsigned >)
`jae` *Label*	`jnb`	`~CF`	Above or Equal (Unsigned >=)
`jb` *Label*	`jnae`	`CF`	Below (Unsigned <)
`jbe` *Label*	`jna`	`CF & ~ZF`	Below or Equal (Unsigned <=)

Figure 3.10: **The `jump` Instructions.** These instructions jump to a labeled destination when the jump condition holds. Some instructions have "synonyms," alternate names for the same machine instruction.

Conditional Jump

- Can’t use indirect jump

- Use it to implement
  - if conditions
  - loops
  - switch statements

Goto in C

```
if (test-expr)
   then-statement
else
   else-statement
```

```
   t = test-expr;
   if (t)
      goto true;
   else-statement
   goto done;
true:
   then-statement
done:
```

If Condition in Assembly

```c
int absdiff(int x, int y)
{
    if (x < y)
        return y - x;
    else
        return x - y;
}
```

```
  # x in %edi, y in %esi
  cmpl    %esi,%edi   # Compare x:y
  jl .L3              # if <, go to less:
  subl    %esi,%edi   # Compute x-y as return value
  movl    %edi,%eax   # Set as return value
  jmp .L5             # Goto done:
.L3:                 # less:
  subl    %edi,%esi   # Compute y-x as return value
  movl    %esi,%eax   # Set as return value
.L5:                 # done: Begin completion code
```

If Condition in Assembly

```c
int absdiff(int x, int y)
{
    if (x < y)
        return y - x;
    else
        return x - y;
}
```

```
  # x in %edi, y in %esi
  cmpl    %esi,%edi    # Compare x:y
  jl    Label    jnge    SF ^ OF    Less (Signed <)
  subl    %esi,%edi    # Compute x-y as return value
  movl    %edi,%eax    # Set as return value
  jmp .L5              # Goto done:
.L3:                   # less:
  subl    %edi,%esi    # Compute y-x as return value
  movl    %esi,%eax    # Set as return value
.L5:                   # done: Begin completion code
```

Do-While Loops

```
do
    body-statement
    while (test-expr);
```

⟹

```
loop:
    body-statement
    t = test-expr;
    if (t)
        goto loop;
```

Do-While Loops in Assembly

```c
long fib_dw(long n)
{
    long i = 0;
    long val = 0;
    long nval = 1;

    do {
        long t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while(i < n);

    return val;
}
```

```
.L6:
  leaq    (%rdx,%rbx),%rax
  movq    %rdx,%rbx
  movq    %rax,%rdx
  incq    %rcx
  cmpq    %rdi,%rcx
  jl .L6
  movq    %rbx,%rax
```

Register Usage		
Register	Variable	Initially
%rcx	i	0
%rdi	n	n
%rbx	val	0
%rdx	nval	1
%rax	t	–

While Loops

```
while (test-expr)
   body-statement
```

```
if (!test-expr)
   goto done;
do
   body-statement
   while (test-expr);
done:
```

```
   t = test-expr;
   if (!t)
      goto done;
loop:
   body-statement
   t = test-expr;
   if (t)
      goto loop;
done:
```

Exercise

```c
int loop_while(int a, int b)
{
    int i = 0;
    int result = a;
    while (i < 256) {
        result += a;
        a -= b;
        i += b;
    }
    return result;
}
```

```asm
  # a in %edi, b in %esi
  xorl     %ecx,%ecx
  movl     %edi,%edx
  .p2align    4,,7
.L5:
  addl     %edi,%edx
  subl     %esi,%edi
  addl     %esi,%ecx
  cmpl     $255,%ecx
  jle .L5
```

Register	Variable	Initially
%eax		
%ebx		
%ecx		
%edx		

Exercise

```c
int loop_while(int a, int b)
{
    int i = 0;
    int result = a;
    while (i < 256) {
        result += a;
        a -= b;
        i += b;
    }
    return result;
}
```

```
  # a in %edi, b in %esi
  xorl     %ecx,%ecx
  movl     %edi,%edx
  .p2align    4,,7
.L5:
  addl     %edi,%edx
  subl     %esi,%edi
  addl     %esi,%ecx
  cmpl     $255,%ecx
  jle .L5
```

Register	Variable	Initially
%eax	a	a
%ebx	b	b
%ecx	i	0
%edx	result	a

Exercise’s Solution

```
  # a in %edi, b in %esi
  xorl     %ecx,%ecx    # i = 0
  movl     %edi,%edx    # result = 0
  .p2align    4,,7
.L5:                    # loop:
  addl     %edi,%edx    # result += a
  subl     %esi,%edi    # a -= b
  addl     %esi,%ecx    # i += b
  cmpl     $255,%ecx    # Compare i:255
  jle .L5               # If <= goto loop
  movl     %edx,%eax    # Set result as return value
```

Note the optimization done by the compiler!

For Loops

```
for (init-expr; test-expr; update-expr)
   body-statement
```

```
init-expr;
while (test-expr) {
   body-statement
   update-expr;
}
```

```
init-expr;
if (!test-expr)
   goto done;
do {
   body-statement
   update-expr;
} while (test-expr);
done:
```

```
   init-expr;
   t = test-expr;
   if (!t)
      goto done;
loop:
   body-statement
   update-expr;
   t = test-expr;
   if (t)
      goto loop;
done:
```

Exercise

```
  # x in %rdi, y in %rsi, n in %rdx
  xorq    %rax,%rax
  decq    %rdx
  js  .L4
  imulq   %rdi,%rsi
.L6:
  addq    %rsi,%rax
  subq    %rdi,%rdx
  jns .L6
.L4:
```

```c
long loop(long x, long y, long n)
{
    long result = 0;
    long i;
    for (i = ____ ; i ____ ; i = ____) {
        result += ____ ;
    }
    return result;
}
```

Exercise's Solution

```c
long loop(long x, long y, long n)
{
    long result = 0;
    long i;
    for (i = n - 1 ; i >= 0 ; i = i - x) {
        result += y * x ;
    }
    return result;
}
```

Note the optimization done by the compiler!

Switch Statements in C

```
long switch_eq(long x)
{
  long result = x;
  switch(x) {

    case 100:
      result *= 13;
      break;

    case 102:
      result += 10;
      /* Fall through */

    case 103:
      result += 11;
      break;

    case 104:
    case 106:
      result *= result;
      break;

    default:
      result = 0;
  }
  return result;
}
```

Switch Statements in Assembly

Building the jump table:

```
  .section .rodata
  .align 8  # Align address to multiple of 8
.L10:
  .quad .L4 # Case 100: loc_A
  .quad .L9 # Case 101: loc_def
  .quad .L5 # Case 102: loc_B
  .quad .L6 # Case 103: loc_C
  .quad .L8 # Case 104: loc_D
  .quad .L9 # Case 105: loc_def
  .quad .L8 # Case 106: loc_D
```

Switch Statements in Assembly

```
# Set up the jump table access
leaq -100(%rdi),%rsi       # Compute xi = x-100
cmpq $6,%rsi               # Compare xi:6
ja .L9                     # if >, goto default-case
jmp *.L10(,%rsi,8)         # Goto jt[xi]

# Case 100
.L4:                       # loc_A:
  leaq (%rdi,%rdi,2),%rax  # Compute 3*result
  leaq (%rdi,%rax,4),%rdi  # Compute x+4*3*result
  jmp .L3                  # Goto done

# Case 102
.L5:                       # loc_B:
  addq $10,%rdi            # result += 10, Fall through:

# Case 103
.L6:                       # loc_C:
  addq $11,%rdi            # result += 11
  jmp .L3                  # Goto done
```

Switch Statements in Assembly

```
# Cases 104,106
.L8:                    # loc_D:
  imulq %rdi,%rdi         # result *= result
  jmp .L3                 # Goto done

# Default case
.L9:                    # loc_D:
  xorq %rdi,%rdi          # result = 0

# Return result
.L3:                    # done:
  movq %rdi,%rax          # Set result as return value
```

Exercise

```c
long switch2(long x)
{
	long result = 0;
	switch(x) {
		/* Body of switch statement omitted */
	}
	return result;
}
```

```
# Setting up the jump table access
leaq $2(%rdi),%rsi  # x in %rdi
cmpq $6,%rsi
ja .L10
jmp *.L11(,%rsi,8)
```

```
# Jump table for switch2
.L11:
  .quad .L4
  .quad .L10
  .quad .L5
  .quad .L6
  .quad .L8
  .quad .L8
  .quad .L9
```

Q & A

- Q:
  - What were the values of the case labels in the switch statement body?
  - What cases had multiple labels in the C code?

- A:
  - The case labels had values -2,0,1,2,3,4
  - The case with the labels 2 and 3