[Chap.3-4] Machine-level Representation of Programs

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

SKK UNIVERSITY

Contents

- Introduction
- Program encodings
- Data formats
- Intel processors
- Accessing information
- Primitive instructions
- Data movement instructions
- Arithmetic and logic instructions
- Control instructions
- Procedures
- …

Control Instructions

- **C control structures**
  - Conditionals, loops, switches, …
    - **if**, **switch**, **for**, **while**, **do**, …
- **Machine-level control structures**
  - Jump instructions
    - Pass control to some other part of the program, possibly contingent on the result of some test

Control Instructions

Condition codes (CC)

- Single-bit registers
  - CF, ZF, SF, OF
- Implicitly set by arithmetic or logical operations
  - CF (carry flag)
    - ✓ Set if carry out from most significant bit
    - ✓ Used to detect unsigned overflow
  - ZF (zero flag)
    - ✓ Set if the most recent operation yielded 0
  - SF (sign flag)
    - ✓ Set if the most recent operation yielded a negative value
  - OF (overflow flag)
    - ✓ Set if the most recent operation caused 2's-complement overflow

Control Instructions

Condition codes (CC)

- Most arithmetic/logical instructions affects CCs
- The **leaq** instruction does not alter any CCs
- Some instructions affects the CCs in their own way
  - For logical operations such as **xor**,
    - ✓ The CF and OF are set to 0
  - For shift operations,
    - ✓ The CF is set to the last bit shifted out and OF is set to 0
  - For **inc** and **dec**,
    - ✓ The OF and ZF are set by the normal rule, leaving CF unchanged
  - Etc…

Control Instructions

Condition codes (CC)

- Explicit setting by compare (**CMP**) instructions
  - Example) **cmpq b, a**
    - ✓ Computes **(a – b)** without saving the result
    - ✓ CF set if carry out from most significant bit
      - Used for unsigned comparisons
    - ✓ ZF set if **a == b**
    - ✓ SF set if **(a – b) < 0**
    - ✓ OF set if two's complement overflow
      - **(a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0)**

Instruction	Based on	Description
CMP S1,S2 **cmpb** **cmpw** **cmpl** **cmpq**	**S2 – S1**	Compare Compare byte Compare word Compare double word Compare quad word

Control Instructions

Condition codes (CC)

- Explicit setting by test (**TEST**) instructions
  - Example) **testq b, a**
    - ✓ Sets CCs based on the value of **a** and **b**
      - Useful to have one of the operands as a mask
    - ✓ Computes **a & b** without setting destination
    - ✓ ZF set when **a & b == 0**
    - ✓ SF set when **a & b < 0**
    - ✓ CF and OF are cleared to **0**

Instruction	Based on	Description
TEST S1,S2 **testb** **testw** **testl** **testq**	**S1 & S2**	Test Test byte Test word Test double word Test quad word

Control Instructions

Accessing CCs

- **set** instructions
  - Set a single byte destination (register or memory) to 0 or 1 depending on some combination of CCs

Instruction		Synonym	Effect	Description
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 <=)

Control Instructions

Accessing CCs

- **set** instructions
  - Have a single-byte register or a single-byte memory location as its destination
  - Does not alter remaining 3 bytes
  - Typically uses **movzbl** or **movzbq** to finish job

Control Instructions

Accessing CCs

- Example) **set** instructions

```
[C code]
int comp(long a, long b){
   return a < b;
}
```

```
[Assembly code]
   int comp(long, long)
   a in %rdi, b in %rsi
comp:
   cmpq %rsi,%rdi      compare a:b
   setl %al            set low-order byte of %rax to 0 or 1
   movzbl %al,%eax     clear rest of %eax (and rest of %rax)
   ret
```

Note inverted ordering!

Control Instructions

Jump instructions

- Unconditional jump
  - Direct jump
    - ✓ Jump target encoded in the instruction (use label in assembly)
      - Eg) **jmp .L1**
  - Indirect jump
    - ✓ Jump target in register or memory location
    - ✓ Use * followed by an operand specifier
      - Eg) **jmp *%eax**
        jmp *(%eax)

```
   movq $0,%rax
   jmp .L1
   movq (%rax),%rdx
.L1:
   popq %rdx
```

- Conditional jump
  - Can only be direct

Control Instructions

Jump instructions

Instruction	Synonym	Condition	Description
jmp		1	Unconditional
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 (Signed >=)
jl	jnge	(SF ^ OF)	Less (Signed <)
jle	jng	(SF ^ OF) | ZF	Less or Equal (Signed <=)
ja	jnbe	~CF & ~ZF	Above (Unsigned >)
jae	jnb	~CF	Above or Equal (Unsigned >=)
jb	jnae	CF	Below (Unsigned <)
jbe	jna	CF | ZF	Below or Equal (Unsigned <=)

Control Instructions

Jump instructions

- Example) PC-relative addressing → ±128 byte 내의 범위에서 한정

```
   movq %rdi,%rax
   jmp .L2
.L3:
   sarq %rax
.L2:
   testq %rax,%rax
   jg .L3
   rep; ret
```

Assembly code

Disassembled version

```
0:   48 89 f8                mov %rdi,%rax
3:   eb 03          +3       jmp 8 <loop+0x8>
5:   48 d1 f8                sar %rax
8:   48 85 c0                test %rax,%rax
b:   7f f8          -8       jg 5 <loop+0x5>
d:   f3 c3                   repz retq
```

Control Instructions

Translating conditional branches

- Uses conditional and unconditional branches
- Typical translations

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

goto version

```
   t = test_expr;
   if (!t)
      goto false;
   then-statement
   goto done;
false:
   else-statement
done:
```

Control Instructions

Translating conditional branches

- Example)

```
[Original C code]
long lt_cnt = 0;
long ge_cnt = 0;

long adiff_se(long x, long y)
{
   long result;
   if (x < y) {
      lt_cnt++;
      result = y - x;
   }
   else {
      ge_cnt++;
      result = x - y;
   }
   return result;
}
```

```
[goto version]
long lt_cnt = 0;
long ge_cnt = 0;

long gdiff_se(long x, long y)
{
   long result;
   if (x >= y)
      goto x_ge_y;
   lt_cnt++;
   result = y - x;
   return result;
x_ge_y:
   ge_cnt++;
   result = x - y;
   return result;
}
```

Control Instructions

Translating conditional branches

- Example)

```
[Assembly code]
   long adiff_se(long x, long y)
   x in %rdi, y in %rsi
adiff_se:
   cmpq %rsi,%rdi              Compare x:y
   jge .L2                     If >= goto x_ge_y
   addq $1,lt_cnt(%rip)        lt_cnt++
   movq %rsi,%rax
   subq %rdi,%rax              result = y - x
   ret                         Return
.L2:                           x_ge_y:
   addq $1,ge_cnt(%rip)        ge_cnt++
   movq %rdi,%rax
   subq %rsi,%rax              result = x - y
   ret                         Return
```

Control Instructions

Conditional move instructions

- Conditional transfer of data
  - Either copy a value to a register or do nothing, depending on the values of the CCs
- Introduced from Pentium Pro (1995)

- **gcc** generates code using conditional moves, if possible (64-bit versions of Linux and Windows)

Control Instructions

Conditional move instructions

- Two operands (16/32/64-bits long)
  - First operand: source register or memory location
  - Second operand: destination register

Instruction		Synonym	Move condition	Description
cmove	S, R	cmovz	ZF	Equal / zero
cmovne	S, R	cmovnz	~ZF	Not equal / not zero
cmovs	S, R		SF	Negative
cmovns	S, R		~SF	Nonnegative
cmovg	S, R	cmovnle	~(SF ^ OF) & ~ZF	Greater (signed >)
cmovge	S, R	cmovnl	~(SF ^ OF)	Greater or equal (signed >=)
cmovl	S, R	cmovnge	SF ^ OF	Less (signed <)
cmovle	S, R	cmovng	(SF ^ OF) | ZF	Less or equal (signed <=)
cmova	S, R	cmovnbe	~CF & ~ZF	Above (unsigned >)
cmovae	S, R	cmovnb	~CF	Above or equal (Unsigned >=)
cmovb	S, R	cmovnae	CF	Below (unsigned <)
cmovbe	S, R	cmovna	CF | ZF	below or equal (unsigned <=)

Control Instructions

Conditional move instructions

- Typical translations

```
v = test_expr ? then_expr : else_expr
```

```
   if (!test-expr)
      goto false;
   v = then-expr;
   goto done;
false:
   v = else-expr;
done:
```

Conditional assignment version

```
   v = then_expr;
   ve = else-expr;
   t = test_expr;
   if (!t) v = ve;
```

Control Instructions

Conditional move instructions

- Example)

[Original C code]

```
long adiff(long x, long y)
{
   long result;
   if (x < y)
      result = y - x;
   else
      result = x – y;
   return result;
}
```

[conditional assignment version]

```
long cdiff(long x, long y)
{
   long rval = y - x;
   long eval = x – y;
   long ntest = x >= y;
   if (ntest) rval = eval;
   return rval;
}
```

Control Instructions

Conditional move instructions

- Example)

```
[Assembly code]
   long adiff(long x, long y)
   x in %rdi, y in %rsi

adiff:
   movq %rsi,%rax
   subq %rdi,%rax              rval = y - x
   movq %rdi,%rdx
   subq %rsi,%rdx              eval = x - y
   cmpq %rsi,%rdi              Compare x : y
   cmovge %rdx,%rax            if >=, rval = eval
   ret                         Return rval
```

Control Instructions

Conditional move instructions

- Not all conditional expressions can be compiled using conditional moves
  - The abstract code evaluates both *then-expr* and *else-expr* regardless of the test outcome
  - If one of those two expressions could possibly generate an error condition or a side effect,
    - ✓ This could lead to invalid behavior

Control Instructions

Conditional move instructions

- Not all conditional expressions can be compiled using conditional moves

```
[C code]
long cread(long *xp){
   return (xp ? *xp : 0);
}
```

```
[Assembly code]
   long cread(long *xp)
   xp in %rdi

cread:
   movl $0,%edx              ve = 0
   movq (%rdi),%rax          v = *xp
   testq %rdi,%rdi           Test x
   cmove %rdx,%rax           If x==0, v = ve
   ret                       Return v
```

Null pointer dereferencing when xp == 0

Control Instructions

Conditional move instructions

- The code using conditional data transfers can outperform the code based on conditional control transfers
  - We should understand modern processor architectures
    - ✓ Piplining
    - ✓ Branch prediction
    - ✓ Out-of-order execution
    - ✓ Etc…
- But, using conditional moves does not always improve the code efficiency

Control Instructions

Types of loops

- do-while
- while
- for

- **gcc** and other compilers generate loop code based on the two basic loop patterns

Control Instructions

Translating do-while loops

- Uses combinations of conditional tests and jumps
- Used as a base form in generating loop code (by most compilers)

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

goto version

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

Control Instructions

Translating do-while loops

- Example) Factorial (n! for n > 0)

C code

```
long fact_do (long n)
{
   long result = 1;
   do {
      result *= n;
      n = n-1;
   } while (n > 1);
   return result;
}
```

goto version

```
long fact_do_goto (long n)
{
   long result = 1;
loop:
   result *= n;
   n = n-1;
   if (n > 1)
      goto loop;
   return result;
}
```

Control Instructions

Translating do-while loops

- Example)

Goto version

```
long fact_do_goto (long n)
{
   long result = 1;
loop:
   result *= n;
   n = n-1;
   if (n > 1)
      goto loop;
   return result;
}
```

Assembly code

```
   long fact_do(long n)
   n in %rdi
fact_do:
   movl $1,%eax       Set result = 1
.L2:                loop:
   imulq %rdi,%rax    Compute result *= n
   subq $1,%rdi       Decrement n
   cmpq $1,%rdi       Compare n:1
   jg .L2             If >, goto loop
   rep; ret           Return
```

[Registers]

```
   %rdi    n
   %rax    result
```

Control Instructions

- Translating while loops
  - **Jump-to-middle** method (**gcc –Og**)

C code

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

goto version

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

Control Instructions

Translating while loops

- Example) Factorial (n! for n >= 0)

C code

```
long fact_while (long n)
{
   long result = 1;
   while (n > 1) {
      result *= n;
      n = n-1;
   }
   return result;
}
```

goto version

```
long fact_while_jm (long n)
{
   long result = 1;
   goto test;
loop:
   result *= n;
   n = n-1;
test:
   if (n > 1)
      goto loop;
   return result;
}
```

Control Instructions

Translating while loops

- Example) Factorial (n! for n >= 0)

Goto version

```
long fact_while_jm (long n)
{
   long result = 1;
   goto test;
loop:
   result *= n;
   n = n-1;
test:
   if (n > 1)
      goto loop;
   return result;
}
```

[Registers]

%rdi	n
%rax	result

Assembly code

```
   long fact_while(long n)
   n in %rdi
fact_while:
   movl $1,%eax       Set result = 1
   jmp .L5            Goto test
.L6:               loop:
   imulq %rdi,%rax    Compute result *= n
   subq $1,%rdi       Decrement n
.L5:               test:
   cmpq $1,%rdi       Compare n:1
   jg .L6             If >, goto loop
   rep; ret           Return
```

Control Instructions

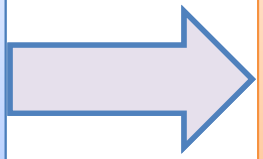
- Translating while loops
  - **Guarded-do** method (**gcc –O1**)

C code

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

do-while version

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



goto version

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

Control Instructions

Translating while loops

- Example) Factorial (n! for n >= 0)

C code

```
long fact_while (long n)
{
   long result = 1;
   while (n > 1) {
      result *= n;
      n = n-1;
   }
   return result;
}
```

goto version

```
long fact_while_gd (long n)
{
   long result = 1;
   if (n <= 1)
      goto done;
loop:
   result *= n;
   n = n-1;
   if (n != 1)
      goto loop;
done:
   return result;
}
```

Control Instructions

Translating while loops

- Example) Factorial (n! for n >= 0)

Goto version

```
long fact_while_gd (long n)
{
   long result = 1;
   if (n <= 1)
      goto done;
loop:
   result *= n;
   n = n-1;
   if (n != 1)
      goto loop;
done:
   return result;
}
```

Assembly code

```
   long fact_while(long n)
   n in %rdi
fact_while:
   movl $1,%eax       Set result = 1
   cmpq $1,%rdi       Compare n:1
   jle .L7            If <=, goto done
.L6:               loop:
   imulq %rdi,%rax    Compute result *= n
   subq $1,%rdi       Decrement n
   cmpq $1,%rdi       Compare n:1
   jne .L6            If !=, goto loop
   rep; ret           Return
.L7:               done:
   ret                Return
```

Control Instructions

- Translating for loops
  - **Jump-to-middle** method

for version

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

while version

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

goto version

```
    init;
    goto testl;
loop:
    body-statement
    update;
testl:
    if (test)
        goto loop;
```

Control Instructions

- Translating for loops
  - **Guarded-do** method

for version

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

while version

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

do-while version

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

goto version

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

Control Instructions

Translating for loops

- Example) Factorial (n! for n >= 0)

C code

```
long fact_for(long n)
{
    long i;
    long result = 1;
    for (i=2; i<=n; i++)
        result *= i;
    return result;
}
```

while version

```
long fact_for_while (long n)
{
    long result = 1;
    long i = 2;
    while (i <= n) {
        result *= i;
        i++;
    }
    return result;
}
```

Control Instructions

Translating for loops

- Example) Factorial (n! for n >= 0)

while version

```
long fact_for_while (long n)
{
   long result = 1;
   long i = 2;
   while (i <= n) {
      result *= i;
      i++;
   }
   return result;
}
```

goto version (jm)

```
long fact_for_jm (long n)
{
   long result = 1;
   long i = 2;
   goto test;
loop:
   result *= i;
   i++;
test:
   if (i <= n)
      goto loop;
   return result;
}
```

Control Instructions

Translating for loops

- Example) Factorial (n! for n >= 0)

Goto version

```
long fact_for_jm (long n)
{
   long result = 1;
   long i = 2;
   goto test;
loop:
   result *= i;
   i++;
test:
   if (i <= n)
      goto loop;
   return result;
}
```

Assembly code

```
   long fact_for(long n)
   n in %rdi

fact_for:
   movl $1,%eax       Set result = 1
   movl $2,%edx       Set i = 2
   jmp .L8            Goto test
.L9:                loop:
   imulq %rdx,%rax    Compute result *= i
   addq $1,%rdx       Increment i
.L8:                test:
   cmpq %rdi,%rdx     Compare i:n
   jle .L9            If <=, goto loop
   rep; ret           Return
```

Control Instructions

Translating switches

- Multi-way branching capability
- Efficient implementation is possible using a data structure called a **jump table**
  - **Jump table**
    - ✓ An array where entry **i** is the address of a code segment implementing the action the program should take when the switch index equals **i**
  - More efficient than a long sequence **if-else** in that the time taken to perform the switch is independent of the number of switch cases

Control Instructions

Translating switches

- **gcc** selects the method of translating a switch statement based on the number of cases and the sparsity of the case values

Control Instructions

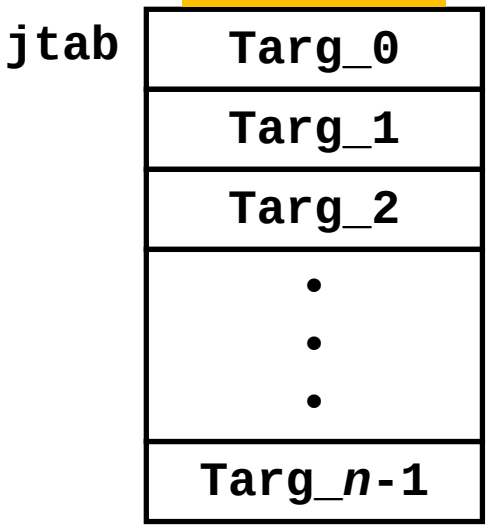
Translating switches

- Jump table structure

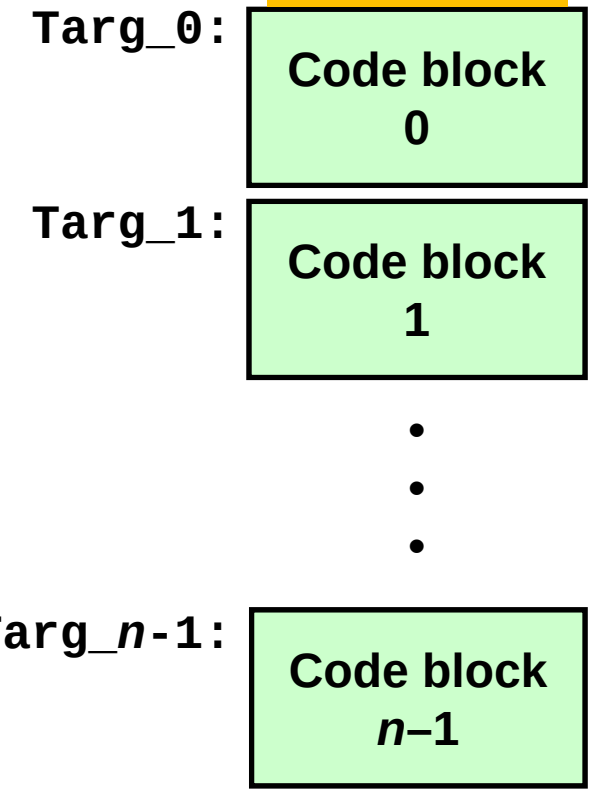
Switch Form

```
switch (op) {
  case val_0:
    Block 0
  case val_1:
    Block 1
    . . .
  case val_n-1:
    Block n-1
}
```

Jump Table



Jump Targets



Approx. Translation

```
target = jtab[op];
goto *target;
```

Control Instructions

Translating switches: Example)

```
void switch_eg (long x, long n, long *d)
{
    long v = x;
    switch(n) {
    case 100:
        v *= 13;
        break;
    case 102:
        v += 10;
        /* Fall through */
    case 103:
        v += 11;
        break;
    case 104:
    case 106:
        v *= v;
        break;
    default:
        v = 0;
    }
    *d = v;
}
```

```
void switch_eg_impl(long x, long n, long *d)
{
   /* Table of code pointers */
   static void *jt[7] = {
      &&l_A, &&l_def, &&l_B, &&l_C,
      &&l_D, &&l_def, &&l_D};
   unsigned long index = n - 100;
   long v;

   if (index > 6)
      goto l_def;
   goto *jt[index]; /* Multiway branch */

l_A:   /* case 100 */
   v = x * 13;
   goto done;
l_B:   /*case 102 */
   v = x + 10;
   /* Fall through */
l_C:   /* case 103 */
   v = x + 11;
   goto done;
l_D:   /* case 104, 106 */
   v = x * x;
   goto done;
l_def:
   v = 0;
done:
   *d = v;
```

Control Instructions

Translating switches: Example)

```
void switch_eg_impl(long x, long n, long *d)
{
   /* Table of code pointers */
   static void *jt[7] = {
      &&l_A, &&l_def, &&l_B, &&l_C,
      &&l_D, &&l_def, &&l_D};
   unsigned long index = n - 100;
   long v;

   if (index > 6)
      goto l_def;
   goto *jt[index]; /* Multiway branch */

l_A:   /* case 100 */
   v = x * 13;
   goto done;
l_B:   /*case 102 */
   v = x + 10;
   /* Fall through */
l_C:   /* case 103 */
   v = x + 11;
   goto done;
l_D:   /* case 104, 106 */
   v = x * x;
   goto done;
l_def:
   v = 0;
done:
   *d = v;
```

Assembly code

```
   void switch_eg(long x, long n, long *d)
   x in %rdi, n in %rsi, d in %rdx

switch_eg:
   subq $100,$rsi              Compute index = n-100
   cmpq $6,%rsi                Compare index:6
   ja .L8                      If >, goto loc_def
   jmp *.L4(,%rsi,8)           Goto *jt[index]
.L3:                        loc_A:
   leaq (%rdi,%rdi,2),%rax      3*x
   leaq (%rdi,%rax,4),%rdi      v = 13*x
   jmp .L2                      Goto done
.L5:                        loc_B:
   addq $10,%rdi                v = x + 10
.L6:                        loc_C:
   addq $11,%rdi                v = x + 11
   jmp .L2                      Goto done
.L7:                        loc_D:
   imulq %rdi,%rdi              v = x * x
   jmp .L2                      Goto done
.L8:                        loc_def:
   movl $0,%edi                 v = 0
.L2:                        done:
   movq %rdi,(%rdx)             *dest = v
   ret                          Return
```

jump table 시작점

Control Instructions

Translating switches: Example)

- Jumping
  - **jmp *.L4(,%rsi,8)**
    - ✓ Start of jump table denoted by label **.L4**
    - ✓ Register **%rsi** holds **index**
      - Must scale by factor of 8 to get offset into the table
    - ✓ Fetch target from effective address **.L4 + index * 8**

Control Instructions

- Translating switches: Example) jump table

```
.section .rodata
  .align 8              Align address to multiple of 8
.L4:
  .quad .L3             Case 100: l_A
  .quad .L8             Case 101: l_def
  .quad .L5             Case 102: l_B
  .quad .L6             Case 103: l_C
  .quad .L7             Case 104: l_D
  .quad .L8             Case 105: l_def
  .quad .L7             Case 106: l_D
```

Control Instructions

Translating switches

- Note) **sparse switch**
  - Not practical to use jump table
    - ✓ Would require 1000 entries
  - Translation into if-else would have maximum 9 tests

```
int div(int x) {
  switch (x) {
  case   0: return 0;
  case 111: return 1;
  case 232: return 2;
  case 356: return 3;
  case 389: return 4;
  case 560: return 5;
  case 682: return 6;
  case 877: return 7;
  case 899: return 8;
  case 900: return 9;
  default: return -1;
  }
}
```

규칙성 X → index mapping이 불가

Summary