Machine Programming: Basics and Control

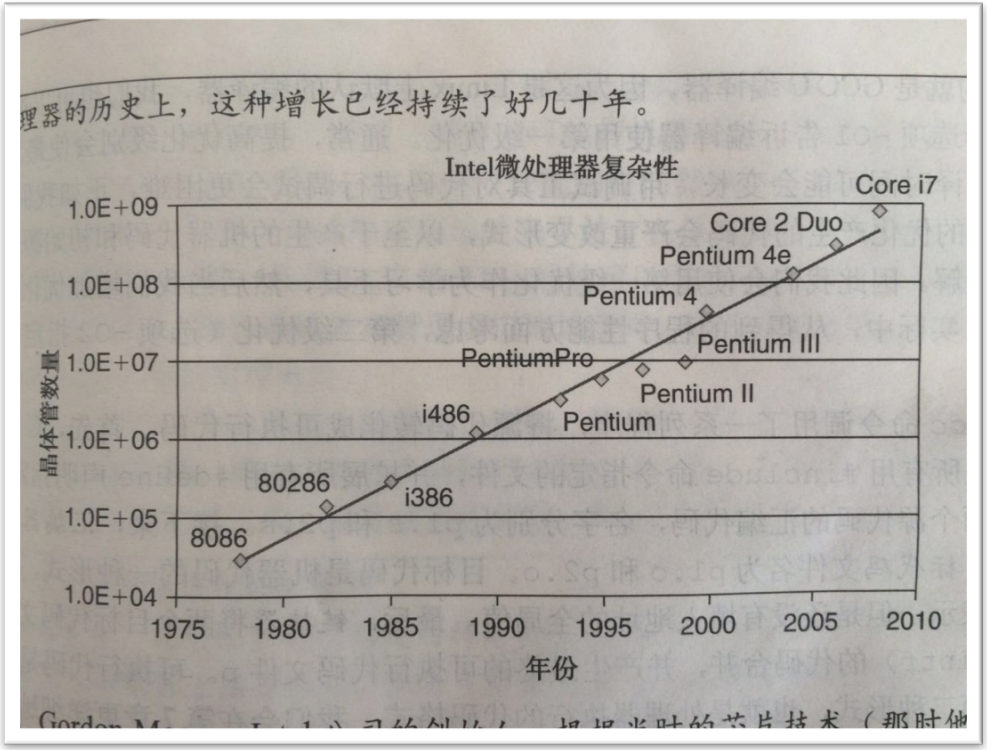
Sun Zheng

Basics

- History
- C, assembly, machine code
- Register, operands, move
- Arithmetic & logical operations

About history

Name	Date	Transistors
8086	1978	29K
80286	1982	134K
I386	1985	275K
I486	1989	1.2M
Pentium 4	2000	42M
Pentium 4E	2004	125M
Core 2	2006	291M
Core i7	2008	781M



x86 Clones: Advanced Micro Devices (AMD)

- Historically
  - AMD has followed just behind Intel.
  - A little bit slower, a lot cheaper.
- Then
  - Recruited top circuit designers from Digital Equipment Corp. and other downward trending companies.
  - Developed x86-64, their own extension to 64 bits.
- Recent
  - Intel got its act together
    - Leads the world in semiconductor technology
  - AMD has fallen behind
    - Relies on external semiconductor manufacturer

C, assembly, machine code

Definitions

Architecture: (also ISA: instruction set architecture) The parts of a processor design that one needs to understand or write assembly/machine code.

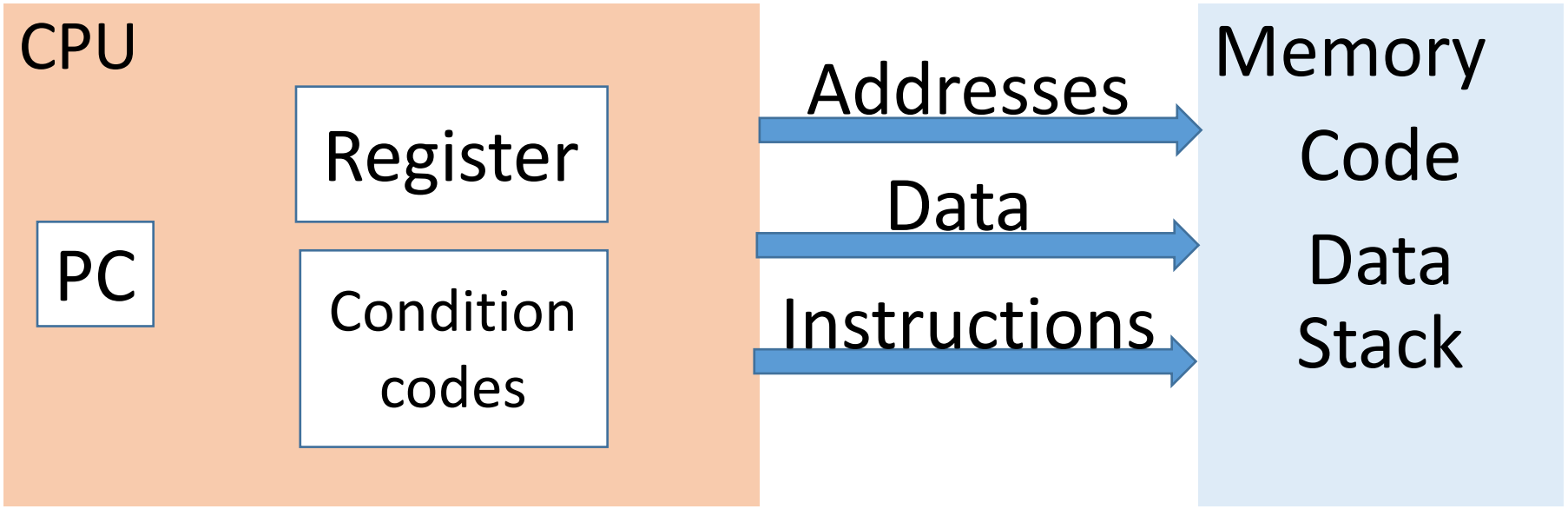
Microarchitecture: Implementation of the architecture.

Code Forms:

Machine Code: The byte-level programs that a processor executes

Assembly Code: A text representation of machine code

Assembly/Machine Code View



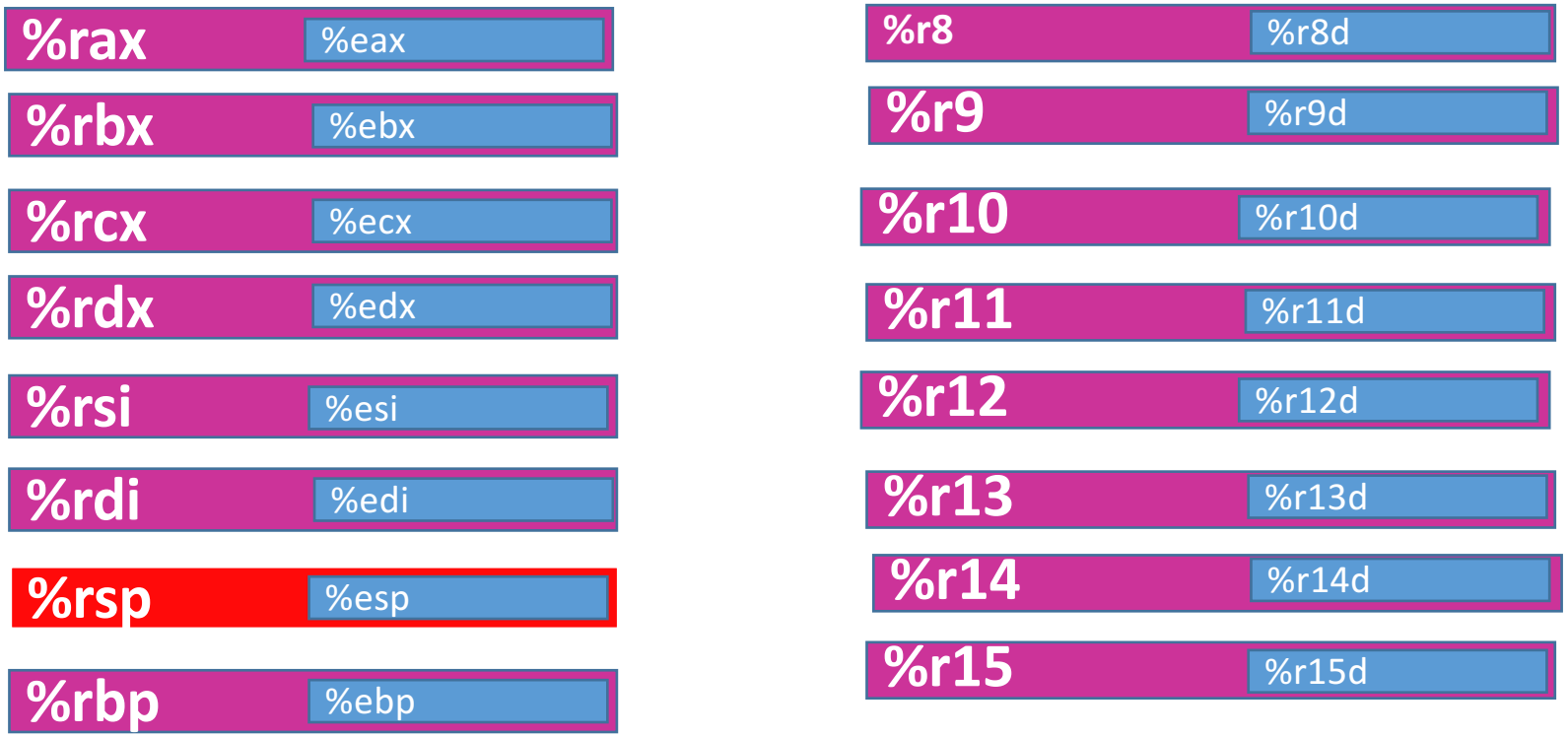
C, assembly, machine code

```
long plus(long x, long y);
void sumstore(long x, long y,
long *dest)
{
long t = plus(x, y);
*dest = t;
}
```

```
sumstore:
pushq %rbx
movq %rdx, %rbx
call plus
movq %rax, (%rbx)
popq %rbx
ret
```

Register, operands, move

- x86-64 Integer Registers



Operands Types

- Immediate: $0x11, $123
- Register: %rax, %r13
- Memory: (%rax), 2(%rax), (rax,rbx), 2(%rax, %rbx, 2)

Memory Addressing Modes

- General Form: D(Rb, Ri, S)
  Mem[Reg[Rb]+S*Reg[Ri]+ D] //S=1,2,4,8
- Special Cases: D(Rb, Ri), (Rb, Ri, s), D(, Ri, s)…

movq Operand Combinations

movq S,D
Source: Imm, Reg, Mem
Dest: Reg, Mem
False:movq (%rax), (%rbx)

movq % rax, %rbx

%rax	0x11	%rax	0x11
%rbx	0x10	%rbx	0x11

Example#Memory Addressing Modes

地址	值
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

寄存器	值
%rax	0x100
%rcx	0x1
%rdx	0x3

操作数	值
0x100	0xFF
$0x108	0x108
(%rax)	0xFF
9(%rax, %rdx)	0x11
(%rax, %rdx, 4)	0x11

Example#move

- void swap
- (long *xp, long *yp)
- {
- long t0 = *xp;
- long t1 = *yp;
- *xp = t1;
- *yp = t0;
- }

- swap:
- movq (%rdi), %rax
- movq (%rsi), %rdx
- movq %rdx, (%rdi)
- movq %rax, (%rsi)
- ret

Arithmetic & logical operations

Format	Computation	Note
leaq S, D	D=&S	D: register
incq D	D=D+1	
decq D	D=D-1	
negq D	D=-D	
notq D	D=~D	
addq S, D	D=D+S	
subq S, D	D=D-S	
imulq S, D	D=D*S	
xorq S, D	D=D^S	
orq S, D	D=D|S	
andq S, D	D=D&S	
salq k,D	D=D<<k	
sarq k,D	D=D>>k	算数右移
shrq k,D	D=D>>k	逻辑右移

Example

```
long arith(long x, long y, long z)
{
  long t1 = x+y;
  long t2 = z+t1;
  long t3 = x+4;
  long t4 = y * 48;
  long t5 = t3 + t4;
  long rval = t2 * t5;
  return rval;
}
```

```
arith:
  leaq (%rdi,%rsi), %rax
  addq %rdx, %rax
  leaq (%rsi,%rsi,2), %rdx
  salq $4, %rdx
  leaq 4(%rdi,%rdx), %rcx
  imulq %rcx, %rax
  ret
```

Control

- Condition codes
- Conditional branches
- Loops
- Switch Statements

Condition Codes

- **CF** Carry Flag (for unsigned)
- **ZF** Zero Flag
- **SF** Sign Flag (for signed)
- **OF** Overflow Flag (for signed)

cmpq(compare)

- Explicit Setting by Compare Instruction
  - **`cmpq`** *Src2*, *Src1*
  - **`cmpq b,a`** like computing **`a-b`** without setting destination

testq(test)

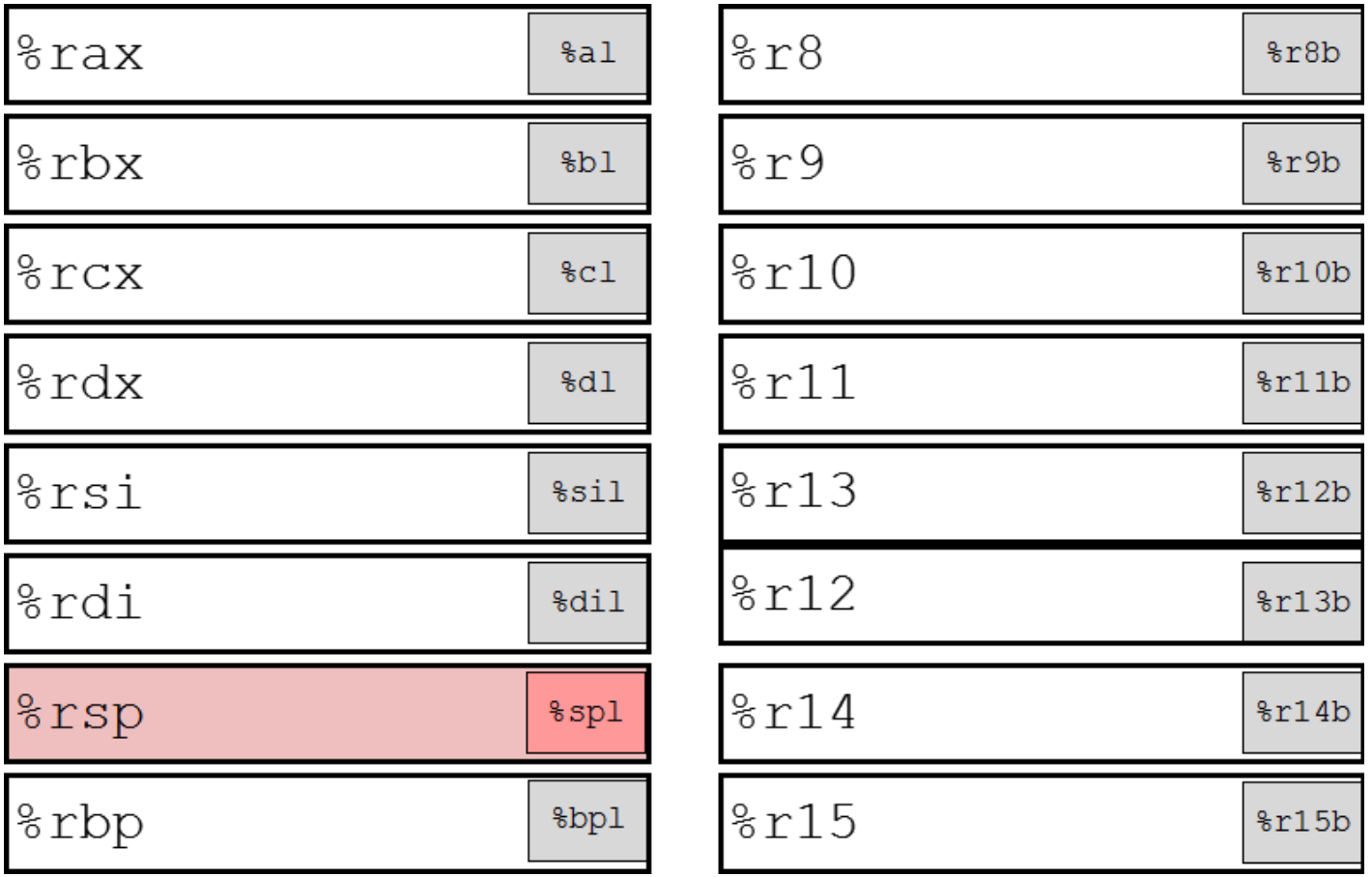
- Explicit Setting by Test instruction
  - **`testq`** *Src2*, *Src1*
    - **`testq b,a`** like computing **`a&b`** without setting destination
    - testq %rax %rax

Reading Condition Codes

- SetX Instructions
  - Set low-order byte of destination to 0 or 1 based on combinations of condition codes
  - 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	Below (unsigned)

x86-64 Integer Registers



Example

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

```
    cmpq    %rsi, %rdi    # Compare x:y
    setg    %al           # Set when >
    movzbl  %al, %eax     # Zero rest of %rax
    ret
```

Conditional branches

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

Jumping

- 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`	Below (unsigned)

- jmp .L ||jmp*operand

  jmp *%rax

  jmp *%(rax)

Example

```
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
    jle     .L4
    movq    %rdi, %rax
    subq    %rsi, %rax
    ret
.L4:                     # x <= y
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

```
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;
}
```

Loops

- Do-While
- While
- For

Do-While

- do

  body

  while(test);

- loop:

  body

  t=test;

  if(t)

  goto loop;

Do–While example

```
int fact-do(int n)
  {
        int result=1;
        do{
            result +=n;
            n=n-1;
         }while(n>1);
         return result;
  }
```

```
int  fact-do(int n)
  {
        int result=1;
        loop:
            result+=n;
            n=n-1;
             if(n>1)
                  go-to loop;
  }
```

```
    movq $0, %rax
.L2:
    addq %rdx , %rax
    subq &1,  %rdx
     cmpq $1, %rdx
      jg  .L2
```

While

```
while (Test)
  Body
```

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

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

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

Example

```
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;
}
```

```
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

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

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

Example

```
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;
}
```

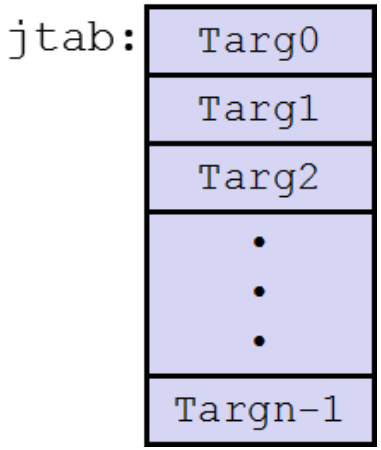
```
long pcount_for_while
  (unsigned long x)
{
  size_t i;
  long result = 0;
  i = 0;
  while (i < WSIZE)
  {
    unsigned bit =
      (x >> i) & 0x1;
    result += bit;
    i++;
  }
  return result;
}
```

Switch

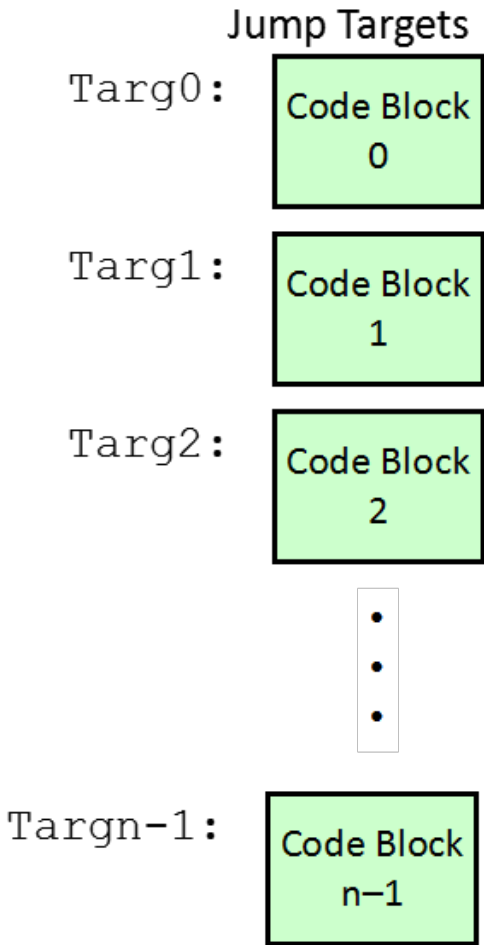
Switch Form

```
switch(x) {
  case val_0:
    Block 0
  case val_1:
    Block 1
    • • •
  case val_n-1:
    Block n–1
}
```

Jump Table



Jump Targets



Example

```
int switch_eg(int x, int n) {
    int result = x;

    switch (n) {

    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;
}
```

```
int switch_eg_impl(int x, int n) {
    /* Table of code pointers */
    static void *jt[7] = {
        &&loc_A, &&loc_def, &&loc_B,
        &&loc_C, &&loc_D, &&loc_def,
        &&loc_D
    };

    unsigned index = n - 100;
    int result;

    if (index > 6)
        goto loc_def;

    /* Multiway branch */
    goto *jt[index];

loc_def:  /* Default case*/
    result = 0;
    goto done;

loc_C:    /* Case 103 */
    result = x;
    goto rest;

loc_A:    /* Case 100 */
    result = x * 13;
    goto done;

loc_B:    /* Case 102 */
    result = x + 10;
    /* Fall through */

rest:     /* Finish case 103 */
    result += 11;
    goto done;

loc_D:    /* Cases 104, 106 */
    result = x * x;
    /* Fall through */

done:
    return result;
}
```

```
x at %ebp+8, n at %ebp+12
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
Set up jump table access
    subl    $100, %eax
    cmpl    $6, %eax
    ja      .L2
    jmp     *.L7(,%eax,4)
Default case
.L2:
    movl    $0, %eax
    jmp     .L8
Case 103
.L5:
    movl    %edx, %eax
    jmp     .L9
Case 100
.L3:
    leal    (%edx,%edx,2), %eax
    leal    (%edx,%eax,4), %eax
    jmp     .L8
Case 102
.L4:
    leal    10(%edx), %eax
Fall through
.L9:
    addl    $11, %eax
    jmp     .L8
Cases 104, 106
.L6:
    movl    %edx, %eax
    imull   %edx, %eax
Fall through
.L8:
Return result
```

```
Get x
Get n

Compute index = n-100
Compare index:6
If >, goto loc_def
Goto *jt[index]

loc_def:
  result = 0;
  Goto done

loc_C:
  result = x;
  Goto rest

loc_A:
  result = x*3;
  result = x+4*result
  Goto done

loc_B:
  result = x+10

rest:
  result += 11;
  Goto done

loc_D
  result = x
  result *= x

done:
```

```
  .section      .rodata
  .align 4      Align address to multiple of 4
.L7:
  .long   .L3   Case 100: loc_A
  .long   .L2   Case 101: loc_def
  .long   .L4   Case 102: loc_B
  .long   .L5   Case 103: loc_C
  .long   .L6   Case 104: loc_D
  .long   .L2   Case 105: loc_def
  .long   .L6   Case 106: loc_D
```

Question

```
int switch2(int x){
    int result=0;
    switch (x){
  /*body*/
 }
}
```

```
x at %rbx+8
    movq   8(%rbx),  %rax
  set up jump table access
    addq   $2,   %rax
    cmpq   $6,   %rax
    ja       .L2
    jmp      *.L8(,%eax,4)
```

```
.L8
    .long    .L3
    .long    .L2
    .lpng    .L4
    .long    .L5
    .long    .L6
    .long    .L6
    .long    .L7
```

Q1:情况标号值？

Q2:哪些情况有多个标号？

Extention1#move of IA32

指令	效果
MOV S,D	D←S
movb	传送字节
Movw	传送字
Movl	传送双字
MOVS S,D	D←符号扩展(S)
Movsbw	将做了符号扩展的字节传送到字
Movsbl	将做了符号扩展的字节传送到双字
Movswl	将做了符号扩展的字传送到双字
MOVZ S,D	D←零扩展(S)
Movzbw	将做了零扩展的字节传送到字
Movzbl	将做了零扩展的字节传送到双字
movzwl	将做了零扩展的字传送到双字

Eaxample

Assume initially that %dl=CD, %eax=98765432

movb %dl, %al	%eax=987654CD
movsbl %dl, %eax	%eax=FFFFFFCD
movzbl %dl, %eax	%eax=000000CD

思考题

```
void cond(int a, int *p){
    if(p&&a>0)
        *p+=a;
}
```

```
%rdx  a   %rax p
  testq   %rax, %rax
  je      .L3
  testq   %rdx, %rdx
  jle     .L3
  addq    %rdx,  (%rax)
.L3:
```