

CS 33

Machine Programming (3)

Processor State (x86-64, Partial)

	%rax	%eax		
	%rbx	%ebx		
a4	%rcx	%ecx		
a3	%rdx	%edx		
a2	%rsi	%esi		
a1	%rdi	%edi		
	%rsp	%esp		
	%rbp	%ebp		
	%rip			
	%r8	%r8d		a5
	%r9	%r9d		a6
	%r10	%r10d		
	%r11	%r11d		
	%r12	%r12d		
	%r13	%r13d		
	%r14	%r14d		
	%r15	%r15d		
	CF	ZF	SF	OF
	condition codes			

Condition Codes (Implicit Setting)

- **Single-bit registers**

CF carry flag (for unsigned)

SF sign flag (for signed)

ZF zero flag

OF overflow flag (for signed)

- **Implicitly set (think of it as side effect) by arithmetic operations**

example: *addl/addq Src,Dest* \leftrightarrow $t = a+b$

CF set if carry out from most significant bit or borrow (unsigned overflow)

ZF set if $t == 0$

SF set if $t < 0$ (as signed)

OF set if two's-complement (signed) overflow

$(a>0 \ \&\& \ b>0 \ \&\& \ t<0) \ || \ (a<0 \ \&\& \ b<0 \ \&\& \ t>=0)$

- **Not set by lea instruction**

Condition Codes (Explicit Setting: Compare)

- **Explicit setting by compare instruction**

`cmpl/cmpq src2, src1`

compares `src1:src2`

`cmpl b, a` like computing $a - b$ without setting destination

CF set if carry out from most significant bit or borrow (used for unsigned comparisons)

ZF set if $a == b$

SF set if $(a - b) < 0$ (as signed)

OF set if two's-complement (signed) overflow

$(a > 0 \ \&\& \ b < 0 \ \&\& \ (a - b) < 0) \ \|\ (a < 0 \ \&\& \ b > 0 \ \&\& \ (a - b) > 0)$

Condition Codes (Explicit Setting: Test)

- **Explicit setting by test instruction**

`testl/testq src2, src1`

`testl b,a` like computing `a&b` without setting destination

- sets condition codes based on value of Src1 & Src2
- useful to have one of the operands be a mask

ZF set when $a \& b == 0$

SF set when $a \& b < 0$

Reading Condition Codes

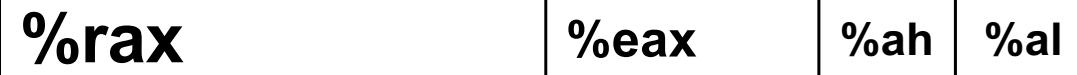
- **SetX instructions**
 - set single byte based on combinations of condition codes

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)

Reading Condition Codes (Cont.)

- **SetX instructions:**
 - set single byte based on combination of condition codes
- **Uses byte registers**
 - does not alter remaining 7 bytes
 - typically use `movzbl` to finish job

```
int gt(int x, int y)
{
    return x > y;
}
```



Body

```
cmpl %esi, %edi      # compare x : y
setg %al              # %al = x > y
movzbl %al, %eax     # zero rest of %eax/%rax
```

Jumping

- **jX instructions**
 - Jump to different part of code depending on condition codes

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	$\sim \text{ZF}$	Not Equal / Not Zero
js	SF	Negative
jns	$\sim \text{SF}$	Nonnegative
jg	$\sim (\text{SF} \wedge \text{OF}) \ \& \ \sim \text{ZF}$	Greater (Signed)
jge	$\sim (\text{SF} \wedge \text{OF})$	Greater or Equal (Signed)
jl	$(\text{SF} \wedge \text{OF})$	Less (Signed)
jle	$(\text{SF} \wedge \text{OF}) \mid \text{ZF}$	Less or Equal (Signed)
ja	$\sim \text{CF} \ \& \ \sim \text{ZF}$	Above (unsigned)
jb	CF	Below (unsigned)

Conditional-Branch Example

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

absdiff:

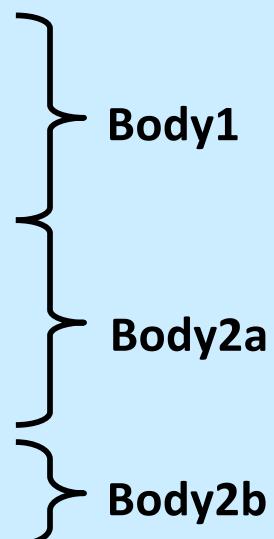
`movl %esi, %eax`
 `cmpl %esi, %edi`
 `jle .L6`
 `subl %eax, %edi`
 `movl %edi, %eax`
 `jmp .L7`

.L6:

`subl %edi, %eax`

.L7:

`ret`



x in %edi

y in %esi

Conditional-Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
    int result;
    if (x <= y) goto Else;
    result = x-y;
    goto Exit;
Else:
    result = y-x;
Exit:
    return result;
}
```

- C allows “goto” as means of transferring control
 - closer to machine-level programming style
- Generally considered bad coding style

```
absdiff:
    movl    %esi, %eax
    cmpl    %esi, %edi
    jle     .L6
    subl    %eax, %edi
    movl    %edi, %eax
    jmp    .L7
.L6:
    subl    %edi, %eax
.L7:
    ret
```

The assembly code is annotated with curly braces on the right side to group specific sections:

- Body1** covers the first five lines of assembly, corresponding to the conditional branch logic in the C code.
- Body2a** covers the sixth line of assembly, corresponding to the "Else" block in the C code.
- Body2b** covers the seventh line of assembly, corresponding to the "Exit" label in the C code.

General Conditional-Expression Translation

C Code

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

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

- Test is expression returning integer
 - == 0 interpreted as false
 - ≠ 0 interpreted as true
- Create separate code regions for then & else expressions
- Execute appropriate one

Goto Version

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

“Do-While” Loop Example

C Code

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

Goto Version

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

- Count number of 1's in argument x (“popcount”)
- Use conditional branch either to continue looping or to exit loop

“Do-While” Loop Compilation

Goto Version

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

Registers:

%edi	x
%eax	result

```
        movl $0, %eax      # result = 0  
.L2:    # loop:  
        movl %edi, %ecx  
        andl $1, %ecx      # t = x & 1  
        addl %ecx, %eax      # result += t  
        shr l %edi         # x >>= 1  
        jne .L2             # if !0, goto loop
```

General “Do-While” Translation

C Code

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

- **Body:** {
 Statement₁;
 Statement₂;
 ...
 Statement_n;
}
- **Test returns integer**
 = 0 interpreted as false
 ≠ 0 interpreted as true

Goto Version

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

“While” Loop Example

C Code

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

Goto Version

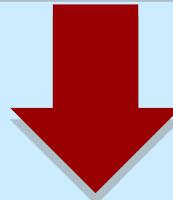
```
int pcount_do(unsigned x) {  
    int result = 0;  
    if (!x) goto done;  
loop:  
    result += x & 0x1;  
    x >>= 1;  
    if (x)  
        goto loop;  
done:  
    return result;  
}
```

- Is this code equivalent to the do-while version?
 - must jump out of loop if test fails

General “While” Translation

While version

```
while (Test)
  Body
```



Do-While Version

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



Goto Version

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

“For” Loop Example

C Code

```
#define WSIZE 8*sizeof(int)
int pcount_for(unsigned x) {
    int i;
    int result = 0;
    for (i = 0; i < WSIZE; i++) {
        unsigned mask = 1 << i;
        result += (x & mask) != 0;
    }
    return result;
}
```

- Is this code equivalent to other versions?

“For” Loop Form

General Form

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

Body

```
for (i = 0; i < WSIZE; i++) {  
    unsigned mask = 1 << i;  
    result += (x & mask) != 0;  
}
```

Init

```
i = 0
```

Test

```
i < WSIZE
```

Update

```
i++
```

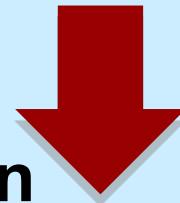
Body

```
{  
    unsigned mask = 1 << i;  
    result += (x & mask) != 0;  
}
```

“For” Loop → While Loop

For Version

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



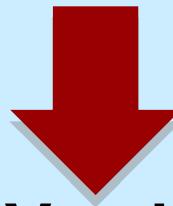
While Version

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

“For” Loop → ... → Goto

For Version

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

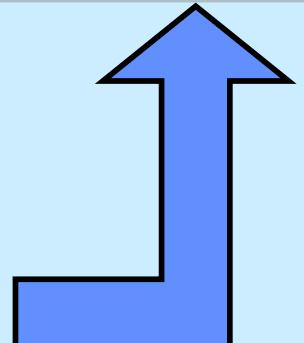


While Version

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



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



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

“For” Loop Conversion Example

C Code

```
#define WSIZE 8*sizeof(int)
int pcount_for(unsigned x) {
    int i;
    int result = 0;
    for (i = 0; i < WSIZE; i++) {
        unsigned mask = 1 << i;
        result += (x & mask) != 0;
    }
    return result;
}
```

Initial test can be optimized away

Goto Version

```
int pcount_for_gt(unsigned x) {
    int i;
    int result = 0; Init
    i = 0;
    if (!(i < WSIZE)) !Test
        goto done;
loop:
{
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
}
Update
if (i < WSIZE) Test
    goto loop;
done:
return result;
}
```

```
long switch_eg
(long x, long y, long z) {
    long w = 1;
    switch(x) {
        case 1:
            w = y*z;
            break;
        case 2:
            w = y+z;
            /* Fall Through */
        case 3:
            w += z;
            break;
        case 5:
        case 6:
            w -= z;
            break;
        default:
            w = 2;
    }
    return w;
}
```

Switch-Statement Example

- **Multiple case labels**
 - here: 5 & 6
- **Fall-through cases**
 - here: 2
- **Missing cases**
 - here: 4

Offset Structure

Switch Form

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

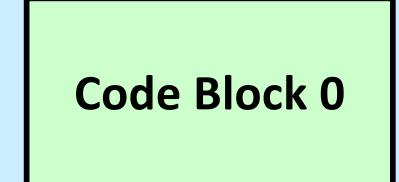
Jump Offset Table

Otab:

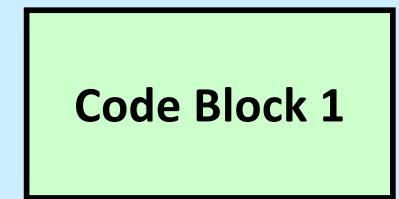
Targ0 Offset
Targ1 Offset
Targ2 Offset
•
•
•
Targn-1 Offset

Jump Targets

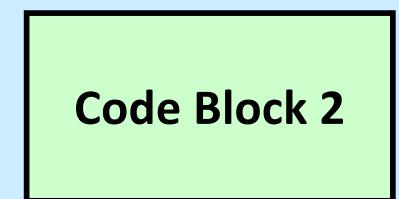
Targ0:



Targ1:



Targ2:



•

•

•

Targn-1:



Approximate Translation

```
target = Otab + OTab[x];  
goto *target;
```

Assembler Code (1)

```
switch_eg:  
    cmpq    $6, %rdi  
    ja     .L8  
    leaq    .L4(%rip), %r8  
    movslq  (%r8,%rdi,4), %rcx  
    addq    %r8, %rcx  
    jmp    *%rcx  
  
    .section .rodata  
    .align 4  
    .L4:  
    .long   .L8-.L4  
    .long   .L7-.L4  
    .long   .L6-.L4  
    .long   .L9-.L4  
    .long   .L8-.L4  
    .long   .L3-.L4  
    .long   .L3-.L4  
    .text  
    .L7:  
    movq    %rsi, %rax  
    imulq  %rdx, %rax  
    ret
```

Assembler Code (2)

.L6:

```
    leaq    (%rsi,%rdx), %rax  
    jmp     .L5
```

.L9:

```
    movl    $1, %eax
```

.L5:

```
    addq    %rdx, %rax  
    ret
```

.L3:

```
    movl    $1, %eax  
    subq    %rdx, %rax  
    ret
```

.L8:

```
    movl    $2, %eax  
    ret
```

Switch-Statement Example (x86-64)

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
        . . .
    }
    return w;
}
```

What range of values is covered by the default case?

Setup:

```
switch_eg:
...      # Setup
cmpq    $6, %rdi          # Compare x:6
ja      .L8 <              # If unsigned > goto default
leaq    .L4(%rip), %r8    # Get address of offset table
movslq  (%r8,%rdi,4),%rcx # Get offset from table
addq    %r8, %rcx          # Add offset to address of table
jmp     *%rcx              # Goto * (OTab + OTab[x])
```

Note that w not initialized here

Switch-Statement Example

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
        . . .
    }
    return w;
}
```

Setup:

```
switch_eg:
    ...      # Setup
    cmpq    $6, %rdi          # Compare x:6
    ja     .L8                  # If unsigned > goto default
    leaq    .L4(%rip), %r8      # Get address of offset table
    movslq (%r8,%rdi,4),%rcx  # Get offset from table
    addq    %r8, %rcx          # Add offset to address of table
    jmp    *%rcx               # Goto *(OTab + OTab[x])
```

Indirect
jump 

Jump Offset Table

```
.section  .rodata
.align 4
.L4:
.long   .L8-.L4  # x = 0
.long   .L7-.L4  # x = 1
.long   .L6-.L4  # x = 2
.long   .L9-.L4  # x = 3
.long   .L8-.L4  # x = 4
.long   .L3-.L4  # x = 5
.long   .L3-.L4  # x = 6
.text
.L7:
```

Assembly-Setup Explanation

- **Table structure**

- each offset is 4 bytes
- base address at .L4

- **Loading the offset**

```
leaq .L4(%rip), %r8
```

- gcc knows .L4's offset relative to %rip

```
movslq (%r8, %rdi, 4), %rcx
```

- get the offset from the table, convert to quad

```
addq %r8, %rcx
```

- add table address to offset

Jump Offset Table

```
.section .rodata
.align 4
.L4:
.long .L8-.L4 # x = 0
.long .L7-.L4 # x = 1
.long .L6-.L4 # x = 2
.long .L9-.L4 # x = 3
.long .L8-.L4 # x = 4
.long .L3-.L4 # x = 5
.long .L3-.L4 # x = 6
.text
.L7:
```

Assembly-Setup Explanation

- **Jumping**

direct: `jmp .L4`

- jump target is denoted by label `.L4`

indirect: `jmp *%rcx`

- jump to address contained in `%rcx`

Offset table

```
.section  .rodata
.align 4
.L4:
.long   .L8-.L4  # x = 0
.long   .L7-.L4  # x = 1
.long   .L6-.L4  # x = 2
.long   .L9-.L4  # x = 3
.long   .L8-.L4  # x = 4
.long   .L3-.L4  # x = 5
.long   .L3-.L4  # x = 6
.text
.L7:
```

Offset Table

Offset table

```
.section    .rodata
.align 4
.L4:
.long .L8-.L4 # x = 0
.long .L7-.L4 # x = 1
.long .L6-.L4 # x = 2
.long .L9-.L4 # x = 3
.long .L8-.L4 # x = 4
.long .L3-.L4 # x = 5
.long .L3-.L4 # x = 6
```

```
switch(x) {
    case 1:          // .L7
        w = y*z;
        break;
    case 2:          // .L6
        w = y/z;
        /* Fall Through */
    case 3:          // .L9
        w += z;
        break;
    case 5:
    case 6:          // .L3
        w -= z;
        break;
    default:         // .L8
        w = 2;
}
```

Code Blocks (Partial)

```
switch(x) {  
    case 1:          // .L7  
        w = y*z;  
        break;  
    . . .  
    case 5:          // .L3  
    case 6:          // .L3  
        w -= z;  
        break;  
    default:         // .L8  
        w = 2;  
}
```

```
.L7:                      # x == 1  
    movq  %rsi, %rax      # y  
    imulq %rdx, %rax     # w = y*z  
    ret  
.L6:                      # x == 5, x == 6  
    movl  $1, %eax        # w = 1  
    subq  %rdx, %rax     # w -= z  
    ret  
.L8:                      # Default  
    movl  $2, %eax        # w = 2  
    ret
```

Handling Fall-Through

```
long w = 1;  
. . .  
switch(x) {  
    . . .  
case 2:  
    w = y+z;  
    /* Fall Through */  
case 3:  
    w += z;  
break;  
    . . .  
}
```

```
case 2:  
    w = y/z;  
goto merge;
```

```
case 3:  
    w = 1;  
merge:  
    w += z;
```

Code Blocks (Rest)

```
switch(x) {  
    . . .  
    case 2: // .L6  
        w = y+z;  
        /* Fall Through */  
    case 3: // .L9  
        w += z;  
        break;  
    . . .  
}
```

```
.L6:    # x == 2  
    leaq   (%rsi,%rdx), %rax  
    jmp    .L5  
.L9:    # x == 3  
    movl   $1, %eax # w = 1  
.L5:    # merge:  
    addq   %rdx, %rax # w += z  
    ret
```

Gdb and Switch

```
B+ 0x555555555145 <switch_eg>
    0x555555555149 <switch_eg+4>
    0x55555555514b <switch_eg+6>
> 0x555555555152 <switch_eg+13>
    0x555555555156 <switch_eg+17>
    0x555555555159 <switch_eg+20>
    0x55555555515b <switch_eg+22>
    0x55555555515e <switch_eg+25>
    0x555555555162 <switch_eg+29>
    0x555555555163 <switch_eg+30>
    0x555555555167 <switch_eg+34>
    0x555555555169 <switch_eg+36>
    0x55555555516e <switch_eg+41>
    0x555555555171 <switch_eg+44>
    0x555555555172 <switch_eg+45>
    0x555555555177 <switch_eg+50>
    0x55555555517a <switch_eg+53>
    0x55555555517b <switch_eg+54>
    0x555555555180 <switch_eg+59>

    cmp    $0x6,%rdi
    ja     0x55555555517b <switch_eg+54>
    lea    0xeb2(%rip),%r8          # 0x5
    movslq (%r8,%rdi,4),%rcx
    add    %r8,%rcx
    jmpq   *%rcx
    mov    %rsi,%rax
    imul   %rdx,%rax
    retq
    lea    (%rsi,%rdx,1),%rax
    jmp    0x55555555516e <switch_eg+41>
    mov    $0x1,%eax
    add    %rdx,%rax
    retq
    mov    $0x1,%eax
    sub    %rdx,%rax
    retq
    mov    $0x2,%eax
    retq
```

```
(gdb) x/10dw $r8
0x555555556004: -3721    -3753    -3745    -3739
0x555555556014: -3721    -3730    -3730    680997
0x555555556024: 990059265      64
```

Gdb and Switch

```
B+ 0x555555555145 <switch_eg>
    0x555555555149 <switch_eg+4>
    0x55555555514b <switch_eg+6>
    0x555555555152 <switch_eg+13>
> 0x555555555156 <switch_eg+17>
    0x555555555159 <switch_eg+20>
0x55555555515b <switch_eg+22>
    0x55555555515e <switch_eg+25>
    0x555555555162 <switch_eg+29>
    0x555555555163 <switch_eg+30>
    0x555555555167 <switch_eg+34>
    0x555555555169 <switch_eg+36>
    0x55555555516e <switch_eg+41>
    0x555555555171 <switch_eg+44>
    0x555555555172 <switch_eg+45>
    0x555555555177 <switch_eg+50>
    0x55555555517a <switch_eg+53>
    0x55555555517b <switch_eg+54>
    0x555555555180 <switch_eg+59>

        cmp    $0x6,%rdi
        ja     0x55555555517b <switch_eg+54
        lea    0xeb2(%rip),%r8          # 0x5
        movslq (%r8,%rdi,4),%rcx
        add    %r8,%rcx
        jmpq   *%rcx
mov    %rsi,%rax
        imul   %rdx,%rax
        retq
        lea    (%rsi,%rdx,1),%rax
        jmp    0x55555555516e <switch_eg+41
        mov    $0x1,%eax
        add    %rdx,%rax
        retq
        mov    $0x1,%eax
        sub    %rdx,%rax
        retq
        mov    $0x2,%eax
        retq
```

(gdb) x/10dw \$r8

0x555555556004:	-3721	-3753	-3745	-3739
0x555555556014:	-3721	-3730	-3730	680997
0x555555556024:	990059265		64	

Gdb and Switch

```
B+ 0x555555555145 <switch_eg>
    0x555555555149 <switch_eg+4>
    0x55555555514b <switch_eg+6>
    0x555555555152 <switch_eg+13>
> 0x555555555156 <switch_eg+17>
    0x555555555159 <switch_eg+20>
0x55555555515b <switch_eg+22>
    0x55555555515e <switch_eg+25>
    0x555555555162 <switch_eg+29>
    0x555555555163 <switch_eg+30>
    0x555555555167 <switch_eg+34>
    0x555555555169 <switch_eg+36>
    0x55555555516e <switch_eg+41>
    0x555555555171 <switch_eg+44>
    0x555555555172 <switch_eg+45>
    0x555555555177 <switch_eg+50>
    0x55555555517a <switch_eg+53>
0x55555555517b <switch_eg+54>
    0x555555555180 <switch_eg+59>

        cmp    $0x6,%rdi
        ja     0x55555555517b <switch_eg+54>
        lea    0xeb2(%rip),%r8          # 0x5
        movslq (%r8,%rdi,4),%rcx
        add    %r8,%rcx
        jmpq   *%rcx
mov    %rsi,%rax
        imul   %rdx,%rax
        retq
        lea    (%rsi,%rdx,1),%rax
        jmp    0x55555555516e <switch_eg+41>
        mov    $0x1,%eax
        add    %rdx,%rax
        retq
        mov    $0x1,%eax
        sub    %rdx,%rax
        retq
mov    $0x2,%eax
        retq
```

(gdb) x/10dw \$r8

0x555555556004:	-3721	-3753	-3745	-3739
0x555555556014:	-3721	-3730	-3730	680997
0x555555556024:	990059265		64	

Gdb and Switch

```
B+ 0x555555555145 <switch_eg>
    0x555555555149 <switch_eg+4>
    0x55555555514b <switch_eg+6>
    0x555555555152 <switch_eg+13>
> 0x555555555156 <switch_eg+17>
    0x555555555159 <switch_eg+20>
0x55555555515b <switch_eg+22>
    0x55555555515e <switch_eg+25>
    0x555555555162 <switch_eg+29>
0x555555555163 <switch_eg+30>
    0x555555555167 <switch_eg+34>
    0x555555555169 <switch_eg+36>
    0x55555555516e <switch_eg+41>
    0x555555555171 <switch_eg+44>
    0x555555555172 <switch_eg+45>
    0x555555555177 <switch_eg+50>
    0x55555555517a <switch_eg+53>
0x55555555517b <switch_eg+54>
    0x555555555180 <switch_eg+59>

        cmp    $0x6,%rdi
        ja     0x55555555517b <switch_eg+54>
        lea    0xeb2(%rip),%r8          # 0x5
        movslq (%r8,%rdi,4),%rcx
        add    %r8,%rcx
        jmpq   *%rcx
mov    %rsi,%rax
        imul   %rdx,%rax
        retq
lea    (%rsi,%rdx,1),%rax
        jmp    0x55555555516e <switch_eg+41>
        mov    $0x1,%eax
        add    %rdx,%rax
        retq
        mov    $0x1,%eax
        sub    %rdx,%rax
        retq
mov    $0x2,%eax
        retq
```

(gdb) x/10dw \$r8

0x555555556004:	-3721	-3753	-3745	-3739
0x555555556014:	-3721	-3730	-3730	680997
0x555555556024:	990059265		64	

Quiz 3

What C code would you compile to get the following assembler code?

```
        movq    $0, %rax
.L2:
        movq    %rax, a(,%rax,8)
        addq    $1, %rax
        cmpq    $10, %rax
        jne     .L2
        ret
```

```
long a[10];
void func() {
    long i;
    for (i=0; i<10; i++)
        a[i]= 1;
}
```

a

```
long a[10];
void func() {
    long i=0;
    while (i<10)
        a[i]= i++;
}
```

b

```
long a[10];
void func() {
    long i=0;
    switch (i) {
case 0:
    a[i] = 0;
    break;
default:
    a[i] = 10
    }
}
```

c