x86 Flow Control

Computer Systems Sections 3.3 - 3.6

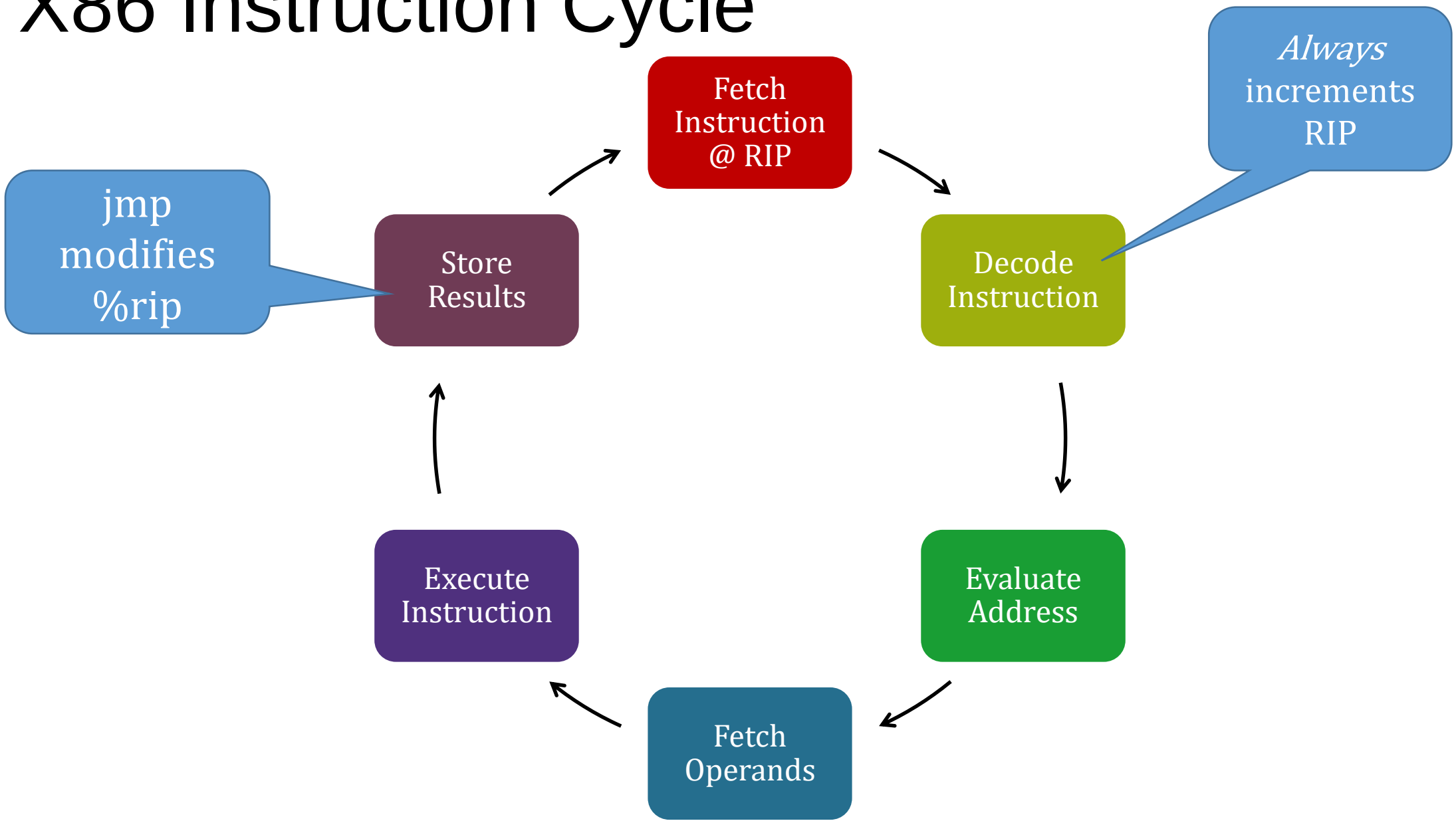
Unconditional Jumps

jmp *target*

- If we were allowed to modify %rip, this would be like:
  mov *target*,%rip

- Updates %rip register with the value of the *target* operand
- Next instruction fetched will be at *target*
- Alters “execute next sequential instruction”

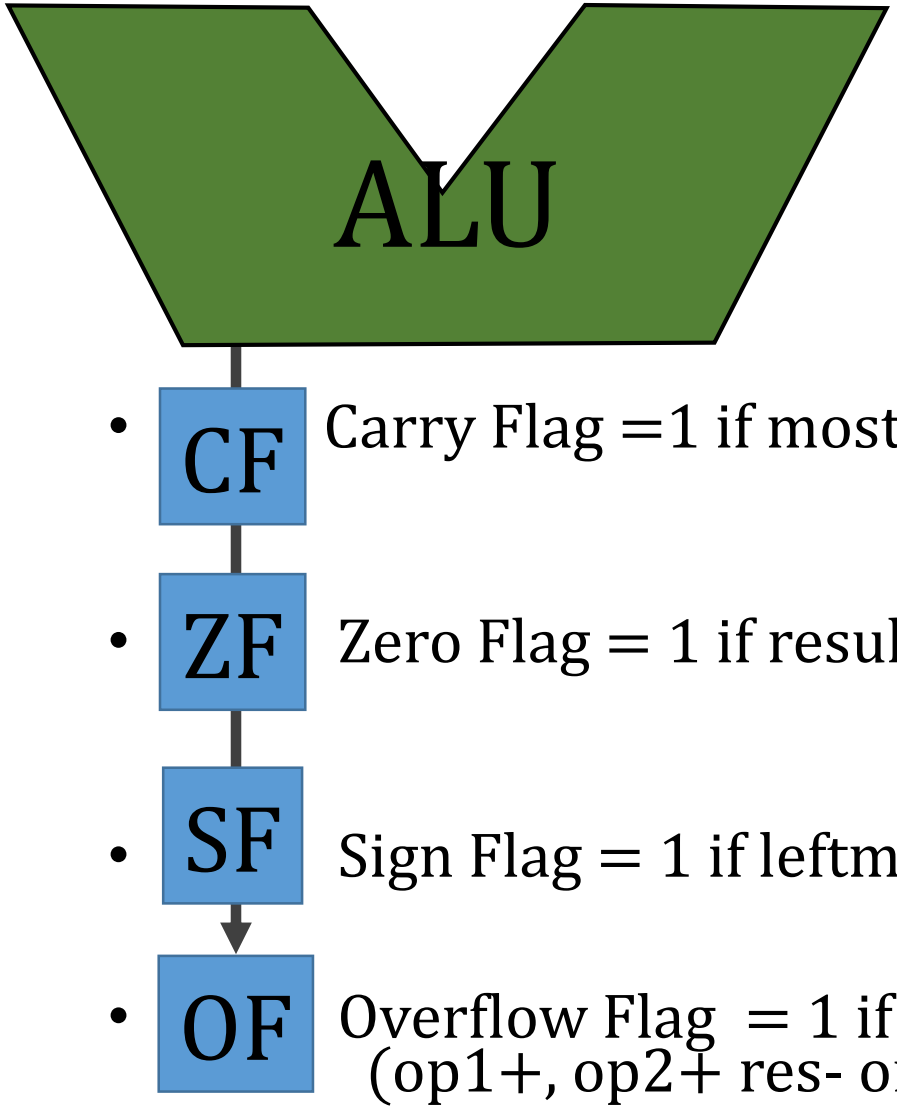
X86 Instruction Cycle



Conditional Jumps

- If you always jumped, you would always execute the “then” block or would always jump back to the start of a loop
- We need some way to tell x86 to jump under some conditions, but not in others
- Conditional jump
  - Sometimes updates %rip
  - Sometimes leaves %rip untouched

Condition Code Registers



- CF Carry Flag =1 if most significant bit overflows (unsigned)
- ZF Zero Flag = 1 if result bits are all zero
- SF Sign Flag = 1 if leftmost result bit is 1 (signed negative)
- OF Overflow Flag = 1 if result sign bit is incorrect
  (op1+, op2+ res- or op1-, op2-, res+)

Unsigned Conditional Jump Mnemonics

cmp *b*,*a* ; *a*'=*a*–*b* without setting destination

OpCode	Description	Interpretation
je	jump if ==	True if ZF=1
jne	jump if !=	True if ZF=0
ja	jump if a>	True if ZF=0 and CF=0
jae	jump if a >=	True if ZF=1 or CF=0
jb	jump if b>	True if ZF=0 and CF=1
jbe	jump if b>=	True if ZF=1 or CF=1

Warning: mnemonics assume cmp b,a, but other instructions may occur

Signed Conditional Jump Mnemonics

cmp *b*,*a* ; *a*'=*a*–*b* without setting destination

OpCode	Description	Interpretation
je	jump a==b	True if ZF=1
jne	jump a!=b	True if ZF=0
jg	jump a>b	True if ZF=0 and SF=0 and OF=0 or if ZF=0 and SF=1 and OF=1
jge	jump a>=b	True if SF=0 and OF=0 or if SF=1 and OF=1
jl	jump a<b	True if ZF=0 and SF=1 and OF=0 or if ZF=0 and SF=0 and OF=1
jle	jump a<=b	True if SF=1 and OF=0 or if SF=0 and OF=1

If/Then/Else

```
int xmpif(int a,int b) {
        int c;
        if (a>b)
                c = a;
        else
                c = b;
        return c;
}
```

```
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -20(%rbp)
        movl    %esi, -24(%rbp)
        movl    -20(%rbp), %eax
        cmpl    -24(%rbp), %eax
        jle     .L2
        movl    -20(%rbp), %eax
        movl    %eax, -4(%rbp)
        jmp     .L3
.L2:    movl    -24(%rbp), %eax
        movl    %eax, -4(%rbp)
.L3:    movl    -4(%rbp), %eax
        popq    %rbp
        ret
```

See progIf.c in xmp_x86

If/Then/Else

```
int xmpif(int a,int b) {
        int c;
        if (a>b)
                c = a;
        else
                c = b;
        return c;
}
```

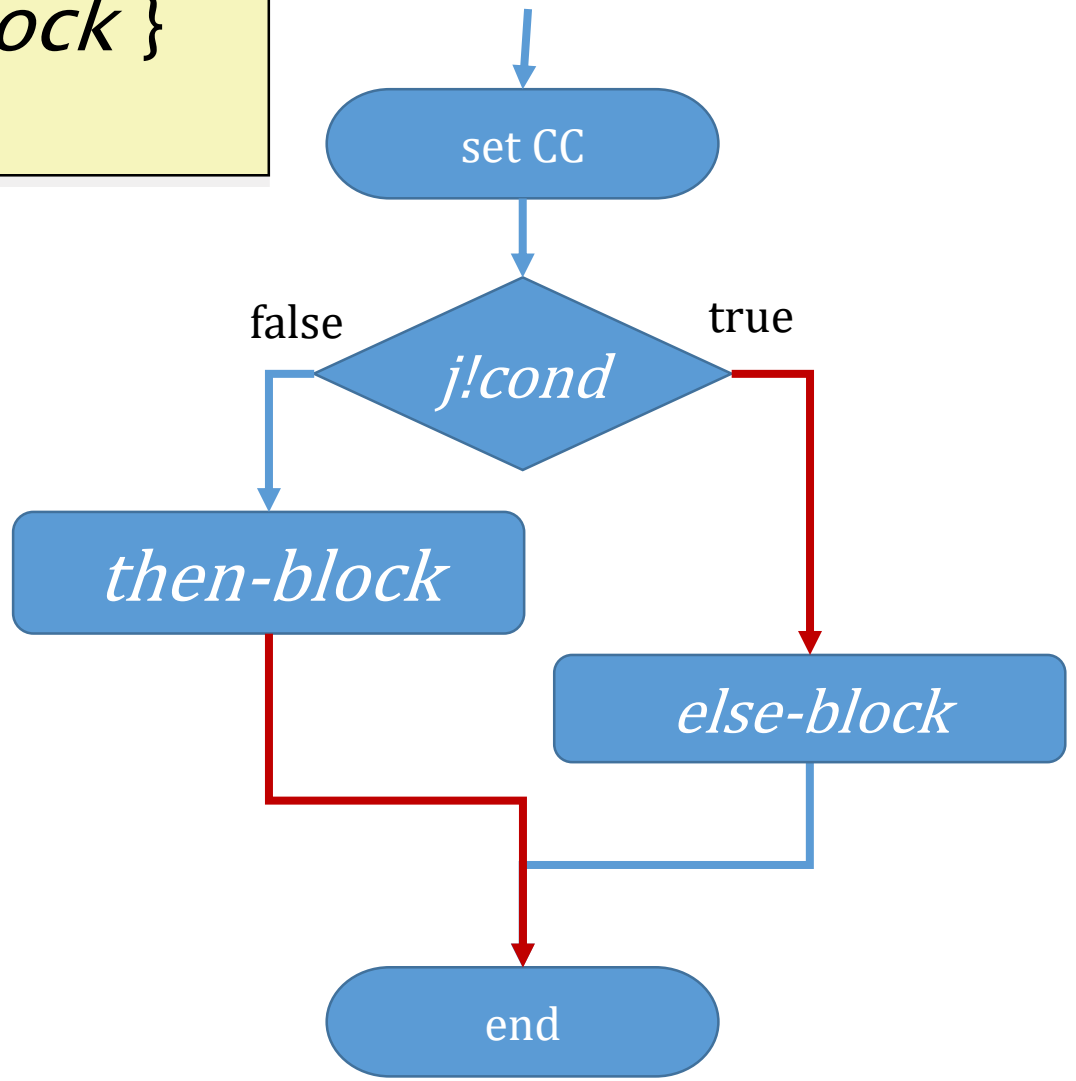
```
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -20(%rbp)
        movl    %esi, -24(%rbp)
        movl    -20(%rbp), %eax
        cmpl    -24(%rbp), %eax
        jle     .L2
        movl    -20(%rbp), %eax
        movl    %eax, -4(%rbp)
        jmp     .L3
.L2:
        movl    -24(%rbp), %eax
        movl    %eax, -4(%rbp)
.L3:
        movl    -4(%rbp), %eax
        popq    %rbp
        ret
```

Memory		
%rbp	xFFFF E4C0	
c	xFFFF E4BC	x0000 0004
	xFFFF E4B8	
	xFFFF E4B4	
	xFFFF E4B0	
a	xFFFF E4AC	x0000 0003
b	xFFFF E4A8	x0000 0004
	...	

Reg	Value
rbp	x7FFF FFFF FFFF E4C0
rdi	x???? ???? 0000 0003
rsi	x???? ???? 0000 0004
rax	x???? ???? 0000 0003 -> 4

Translating C to x86: If/then/else

if (*cond*) { *then-block* }
else { *else-block* }




```
    …
    cmpl    …

    jxx     .L4

    …       ; then block
    jmp  .L5
.L4:
    …       ; else block

.L5:
```

Condition Codes (Explicit Set: Test)

Condition code can be set explicitly by the "test" instruction…

- `test` *a*,*b* like computing ***a'=a&b*** without setting destination

Flag	Set to 1 if…	Interpretation
CF	Carry out or borrow from high order bit	Always 0 after test
ZF	a' is all zeroes	every 1 bit in a is 0 in b
SF	The sign bit is on in a'	a<0 and b<0
OF	The sign bit is incorrect	Always 0 after test

x86 idiom – "jump if false"

```
test a,a
je falseLabel
; a must have been true to get here
```

- Zero is "false", non-zero is "true"
- test a,a ; compute a&a
  - If a==0, then a&a==0 and ZF is 1
  - If a!= 0, then a&a!=0 and ZF is 0
  - je branches only if ZF is 1

While loop

```
int xmpWhile(int a,int b) {
        int c=0;
        while(a<10) {
                c+=b;
                a++;
        }
        return c;
}
```

```
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -20(%rbp)
        movl    %esi, -24(%rbp)
        movl    $0, -4(%rbp)
        jmp     .L4
.L5:
        movl    -24(%rbp), %eax
        addl    %eax, -4(%rbp)
        addl    $1, -20(%rbp)
.L4:
        cmpl    $9, -20(%rbp)
        jle     .L5
        popq    %rbp
        ret
```

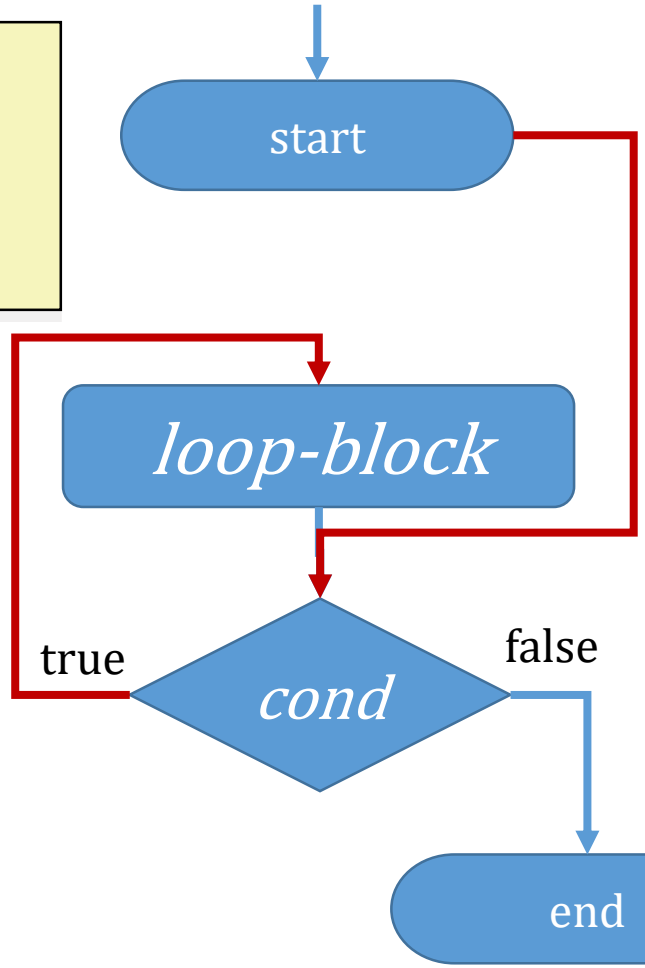
Memory		
%rbp	xFFFF E840	
c	xFFFF E83C	x0000 0000
	xFFFF E838	
	xFFFF E834	
	xFFFF E830	
a	xFFFF E82C	x0000 0003
b	xFFFF E828	x0000 0004
	...	

Reg	Value
rbp	x7FFF FFFF FFFF E840
rdi	x???? ???? 0000 0003
rsi	x???? ???? 0000 0004
rax	x???? ???? temp

See progWhile.c in xmp_x86

Translating C to x86: while

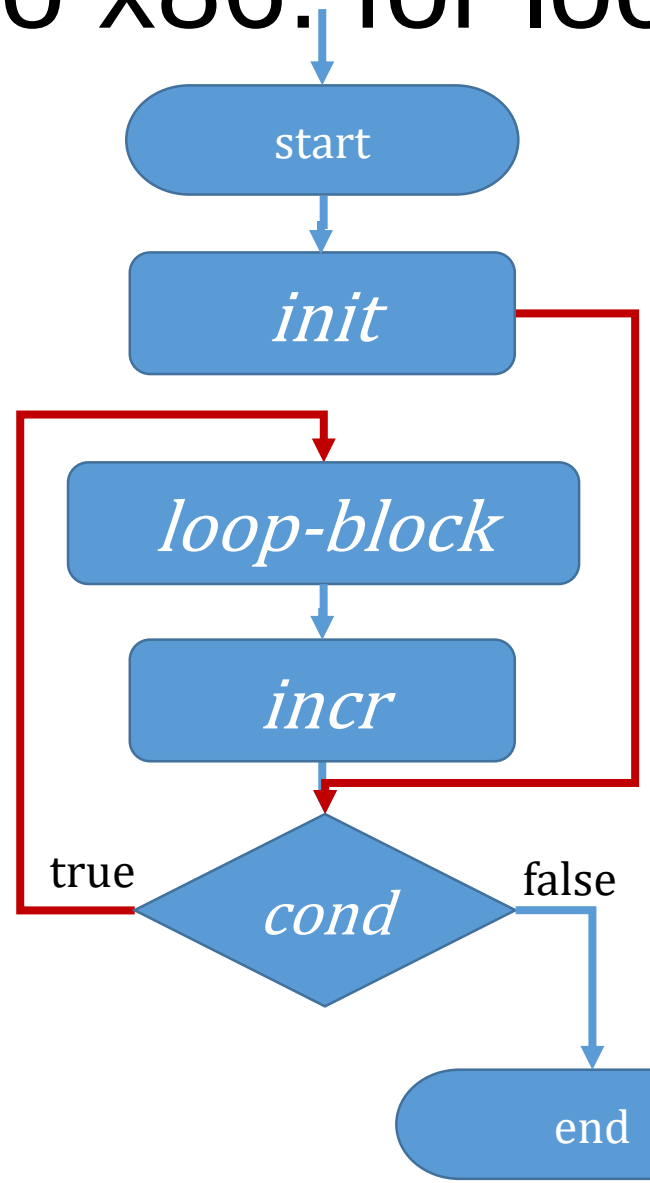
```
while ( cond ) {
        loop-block
}
```




```
        …
        jmp L4

.L5:
        …           ; loop block


.L4:
        cmpl    …; condition
        jxx     .L5

        …
```

Translating C to x86: for loops

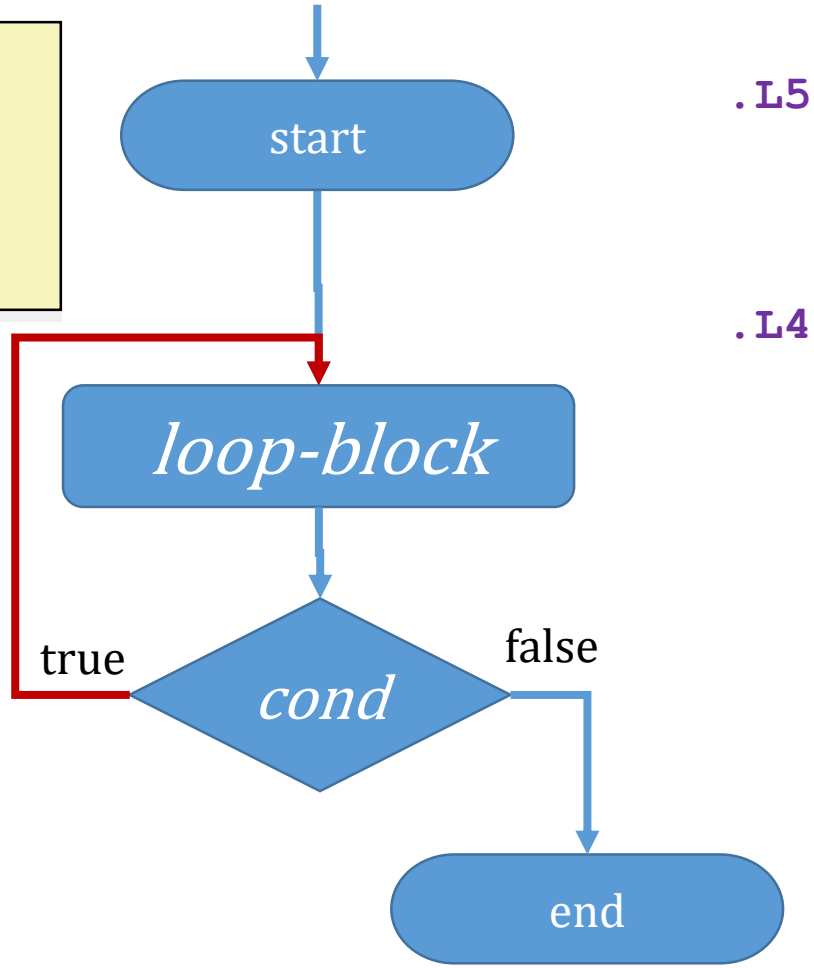
```
for(init;cond;incr) {
        loop-block
}
```




```
        ...
        ...     ; init
        jmp L4
.L5:
        ...     ; loop block

        ...     ; incr

.L4:
        cmpl    ...; condition
        jxx     .L5

        ...
```

Translating C to x86: do-while

```
do {
        loop-block
} while(condition);
```




```
.L5:
    …           ; loop block

.L4:
    cmpl    …; condition
    jxx     .L5

    …
```

Example Switch Statement….

```
switch(a) {
        case 10 : c=a; break;
        case 11: b++;
        case 12:
        case 13: c=a+b; break;
        case 14: c=a*b; break;
        case 15: c=a+2; break;
        default: c=0;
}
```

See progSwitch.c in xmp_x86

C Switch Statement

- Specify a variable in the switch statement – the value to switch on
- Each case statement specifies a literal value
  - switch jumps to first matching case
  - Code is executed from the case statement to a “break”
  - Subsequent case statements before a “break” are ignored (skipped), but statements after the case statement are executed
  - “break” jumps to the end of the switch statement
- If no case statement is satisfied, jump to the “default” statement

Convert cases to labeled x86 code

```
switch(a) {
    case 10 : c=a; break;
    case 11: b++;
    case 12:
    case 13: c=a+b; break;
    case 14: c=a*b; break;
    case 15: c=a+2; break;
    default: c=0;
  }
```

```
.L5:    movl    -20(%rbp), %eax
        movl    %eax, -4(%rbp)
        jmp     .L11
.L7:    addl    $1, -24(%rbp)
.L8:    movl    -20(%rbp), %edx
        movl    -24(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        jmp     .L11
.L9:    movl    -20(%rbp), %eax
        imull   -24(%rbp), %eax
        movl    %eax, -4(%rbp)
        jmp     .L11
.L10:   movl    -20(%rbp), %eax
        addl    $2, %eax
        movl    %eax, -4(%rbp)
        jmp     .L11
.L4:    movl    $0, -4(%rbp)
.L11
```

C	X86
a	-20(%rbp)
b	-24(%rbp)
c	-4(%rbp)

gcc Compilation of Switch statement

- Is there a way to map each case to a table index?
- If so, is the table densely populated?
- If so, make a jump table (after next slide)
- If not, compiler implements…..

```
if (case 10) goto 10 block
else if (case 11) goto case 11 block
else if (case 12) goto case 12 block
…
else if (case 15) goto case 15 block
else default block
```

switch(a) w/o jump table

```
        movl    -20(%rbp), %eax ;  put "a" in %eax register
        cmpl    $10,%eax ;         set cc using %eax-10
        je      .L5 ;              ->L5 if a==10
        cmpl    $11,%eax ;         set cc using %eax-11
        je      .L7 ;              -> L7 if a==11
        cmpl    $12, %eax ;        set cc using %eax-12
        je      .L8 ;              -> L8 if a==12
        cmpl    $13,%eax ;         set cc using %eax-13
        je      .L8 ;              -> L8 if a==13
        cmpl    $14,%eax ;         set cc using %eax-14
        je      .L9 ;              -> L9 if a==14
        cmpl    $15,%eax ;         set cc using %eax-15
        je      .L10 ;             -> L10 if a==15
.L4     movl    $0, -4(%rbp) ;     default, set c=0
```

```
switch(a) {
        case 10 : c=a; break;
        case 11: b++;
        case 12:
        case 13: c=a+b; break;
        case 14: c=a*b; break;
        case 15: c=a+2; break;
        default: c=0;
}
```

Indirect Jump

jmp **reference*

- *reference* : A register that contains an address
  - The value is the branch target
- May also see **memory_reference*
  - For example, table addressing notation: jmp *.L4(,%rax,8)
  - Use %rax as index into a table starting at L4
  - Each entry in the table is 8 bytes wide (one address)
  - Use the %raxth entry as the target for the jump

C compiler creates a “Jump Table”

Case	index	Label	Address
10	0	L5	4004fd
11	1	L7	400505
12	2	L8	400509
13	3	L8	400509
14	4	L9	400516
15	5	L10	400522

a-10

Memory	
Address	Value
…	
0000 0000 0040 05f0	0000 0000 0040 0522
0000 0000 0040 05e8	0000 0000 0040 0516
0000 0000 0040 05e0	0000 0000 0040 0509
0000 0000 0040 05d8	0000 0000 0040 0509
0000 0000 0040 05d0	0000 0000 0040 0505
0000 0000 0040 05c8	0000 0000 0040 04fd
…	

Implementation of switch(a)

```
movl  -20(%rbp), %eax ;  put "a" in %eax register
subl   $10, %eax ;            %eax-=10… index in jump table
cmpl  $5, %eax ;              Set condition based on %eax-5
ja       .L4 ;                -> default if %eax>5 OR %eax < 0
movl  %eax, %eax ;            ??? %rax=%eax (zero high bits)
movq .L6(,%rax,8), %rax ; %rax=jumptable[%eax]
jmp    *%rax ;                 Indirect jump to jumptable[%eax]
```