



Part 6

Control Logic



Intel x86 Jump Instructions

Fly over code

Operations: Program Flow Control

- Unlike high-level languages, processors don't have fancy expressions or blocks
- Programs are controlled by jumping over blocks of code based on status flags



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

3

Operations: Program Flow Control

- The processor moves the program counter *(where your program is running in memory)* to a new address and execution continues



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

4

Types of Jumps: Unconditional

- Unconditional jumps simple transfers the running program to a new address
- Basically, it just "gotos" to a new line
- These are used extensively to recreate the blocks we use in 3GLs (like Java)

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

5

Instruction: Jump

JMP *address*

Usually a label – an constant that holds an address

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

6

Infinite Loop

```
.data
message:
.ascii "I'm getting dizzy!\n\0"

.text
.global _start

_start:
mov $message, %rcx
Loop:
call PrintCString
jmp Loop
```

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

7

Infinite Loop

```
_start:
mov $message, %rcx
Loop:
call PrintCString
jmp Loop
```

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

8

Instruction: Compare

- Performs a comparison operation between two arguments
- The result of the comparison is used for conditional jumps
- Necessary to construct all conditional statements – if, while, ...

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

9

Behind the scenes...



- The first argument is subtracted from the second
- The result of this computation is used to determine how the operands compare
- This subtraction result is discarded

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

10

But... why subtract?

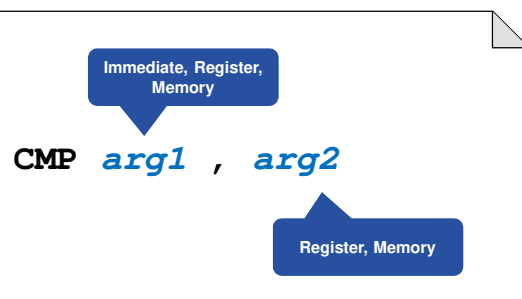
- Why subtract the operands?
- The result can tell you which is larger
- For example: **A** and **B** are both positive...
 - $A - B \rightarrow$ positive number \rightarrow **A** was larger
 - $A - B \rightarrow$ negative number \rightarrow **B** was larger
 - $A - B \rightarrow$ zero \rightarrow both numbers are equal

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

11

Instruction: Compare



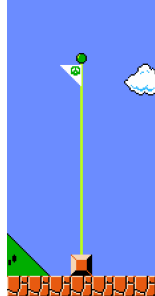
6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

12

Flags

- A *flag* is a Boolean value that indicates the result of an action
- These are set by various actions such as calculations, comparisons, etc...



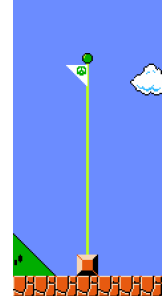
6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

13

Flags

- Flags are typically stored as individual bits in the *Status Register*
- You can't change the register directly, but numerous instructions use it for control and logic



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

14

Zero Flag (ZF)

- True if the last computation resulted in zero (all bits are 0)
- For compare, the zero flag indicates the two operands are equal
- Used by quite a few conditional jump statements

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

15

Sign Flag (SF)

- True if the *most significant bit* of the result is 1
- This would indicate a negative 2's complement number
- Meaningless if the operands are interpreted as unsigned

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

16

Carry Flag (CF)

- True if a 1 is "borrowed" when subtraction is performed
- ...or a 1 is "carried" from addition
- For unsigned numbers, it indicates:
 - exceeded the size of the register on addition
 - or an underflow (too small value) on subtraction

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

17

Overflow Flag (OF)

- Also known as "signed carry flag"
- True if the sign bit changed *when it shouldn't*
- For example:
 - (negative – positive number) should be negative
 - a positive result will set the flag
- For signed numbers, it indicates:
 - exceeded the size of the register on addition
 - or an underflow (too small value) on subtraction

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

18

x86 Flags Used by Compare

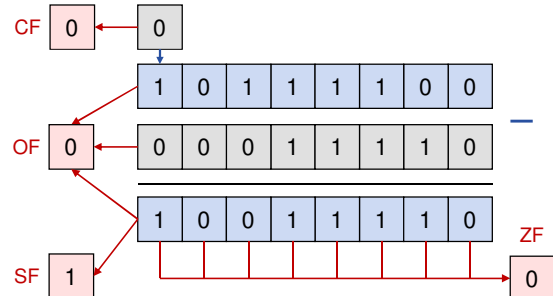
Name	Description	When True
CF	Carry Flag	If an extra bit was "carried" or "borrowed" during math.
ZF	Zero Flag	All the bits in the result are zero.
SF	Sign Flag	If the most significant bit is 1.
OF	Overflow Flag	If the sign-bit changed when it shouldn't have.

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

19

-68 vs. 30 (if interpreted as signed)
188 vs. 30 (if interpreted as unsigned)



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

20

Conditional Jumps

- Conditional jumps (aka *branching*) will only jump if a certain condition is met
- What happens
 - processor jumps if and only if a specific status flag is set
 - otherwise, it simply continues with the next instruction

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

21

Conditional Jumps

- x86 contains a large number of conditional jump statements
- Each takes advantage of status flags (such as the ones set with compare)
- x86 assembly has several names for the same instruction – which adds readability

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

22

Jump on Equality

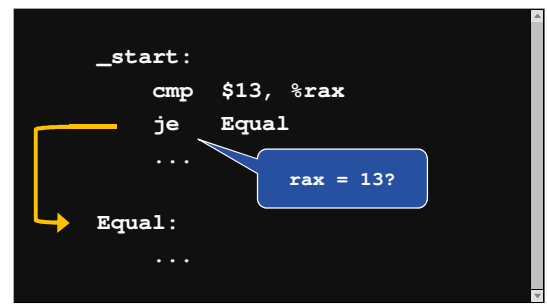
Jump	Description	When True
JE	Equal	ZF = 1
JNE	Not equal	ZF = 0

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

23

Conditional Jump Example



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

24

Signed Jump Instructions

Jump	Description	When True
JG	Jump Greater than	SF = OF, ZF = 0
JGE	Jump Greater than or Equal	SF = OF
JL	Jump Less than	SF ≠ OF, ZF = 0
JLE	Jump Less than or Equal	SF ≠ OF

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

25

Unsigned Jumps

Jump	Description	When True
JA	Jump Above	CF = 0, ZF = 0
JAЕ	Jump Above or Equal	CF = 0
JB	Jump Below	CF = 1, ZF = 0
JBE	Jump Below or Equal	CF = 1

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

26

Conditional Jump Example

```

_start:
    mov $42, %rax
    cmp $13, %rax
    jge Bigger
    ...
Bigger:
    add $5, %rax
    
```

rax >= 13?
(yes, its backwards!)

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

27



If Statements
on the x86

How to we conditionally execute code?

If Statements in assembly

- High-level programming language have easy to use If-Statements
- However, processors handle all branching logic using jumps
- You basically jump over true and else blocks



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

29

If Statements in assembly

- Converting from an If Statement to assembly is easy
- Let's look at If Statements...
 - the block only executes if the expression is true
 - so, if the expression is false your program will skip over the block
 - this is a jump...


6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

30

If Statement jumps over code

```
rax = 18;  
if (rax >= 21) False  
{  
    //true part  
}  
rbx = 12;
```



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

31

Converting an If Statement

- Compare the two values
- If the result is *false* ...
 - then jump over the true block
 - you will need label to jump to
- To jump on false, reverse your logic
 - $a < b \rightarrow \text{not } (a \geq b)$
 - $a \geq b \rightarrow \text{not } (a < b)$

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

32

Please Note...

- Following examples use very generic label names
- In your program, each label you create must be unique
- So, please don't think that each label (as it is typed) is "the" label you need to use



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

33

Converting an If Statement

```
if (rax >= 21) Greater-Than or Equal  
So, jump on Less-Than  
{  
    //true block  
}  
//end
```

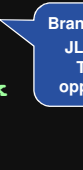
6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

34

Jump over true part

```
cmp $21, %rax  
jl End Branch when false.  
JL (Jump Less Than)  
is the opposite of JGE  
  
#true block  
  
End:
```




6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

35

Jump over true part

```
cmp $21, %rax  
jl End Jumps over  
true part  
  
#true block  
  
End:
```



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

36

Else Clause

- The Else Clause is a tad more complex
- You need to have a true block and a false block
- Like before...
 - you must jump over instructions
 - just remember: *the program will continue with the next instruction unless you jump!*

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

37

Else Clause

```
if (rax >= 21)
{
    //true block
}
else
{
    //false block
}
//end
```

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

38

Jump over true part

```
cmp $21, %rax
jl Else
#true block
jmp End
Else:
#false block
End:
```

Jump to false block

False block flows down to End

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

39

Jump over true part

```
cmp $21, %rax
jl Else
#true block
jmp End
Else:
#false block
End:
```

If we run the true block, we have to jump over the false block

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

40

Alternative Approach

- In the examples before, I put the False Block first and used inverted logic for the jump
- You can construct If Statements without inverting the conditional jump, but the format is layout is different

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

41

If Statement – No Else

```
cmp $21, %rax
jge Then
jmp End
Then:
#true block
End:
```

Jumps to true block

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

42

If Statement – No Else

```

cmp $21, %rax
jge Then
jmp End
Then:
#true block
End:

```

Jump to end if false (it didn't jump with JGE)

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

43

If Statement with Else

```

cmp $21, %rax
jge Then
#false block
jmp End
Then:
#true block
End:

```

Notice that this is identical to the last slide – the false block is just empty

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

44

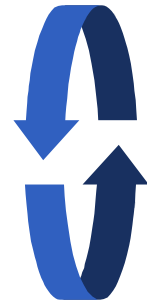


While Loops

Doing the same thing again and again
... and again

While Statement

- Processors do not have While Statements – just like If Statements
- Looping is performed much like an implementing an If Statement
- A While Statement is, in fact, the same thing as an If Statement



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

45

If Statement vs. While Statement

If Statement	While Statement
Uses a conditional expression	Uses a conditional expression
Executes a block of statements	Executes a block of statements
Executes only once	Executes multiple times

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

47

Converting a While Statement

- To create a While Statement
 - start with an If Statement and...
 - add an unconditional jump at the end of the block that jumps to the beginning
- You will "branch out" of an infinite loop
- Structurally, this is almost identical to what you did before
- However, you do need another label :(

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

48

Converting an While Statement

```
while (rax < 21)
{
    //true block
}
//end
```

Less-Than.
So, jump on
Greater-Than or Equal

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

49

Converting an While Statement

```
While:
    cmp $21, %rax
    jge End
    #true block
    jmp While
End:
```

Branch when false. JL
(Jump Less Than) is
the opposite of >=

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

50

Converting an While Statement

```
While:
    cmp $21, %rax
    jge End
    #true block
    jmp While
End:
```

Loop after block
executes

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

51

Converting an While Statement

```
While:
    cmp $21, %rax
    jge End
    #true block
    jmp While
End:
```

Escape infinite
loop

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

52

Alternative Approach

- Before, we created an If Statement by inverting the branch logic (jump on false)
- You can, alternatively, also implement a While Statement without inverting the logic
- Either approach is valid – use what you think is best

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

53

Alternative Approach

```
while (rax < 21)
{
    //true block
}
//end
```

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

54

Alternative Approach

```

While:
    cmp    $21, %rax
    jl     Do
    jmp     End
Do:
    #true block
    jmp     While
End:
    
```

Jumps to Do Block

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

55

Alternative Approach

```

While:
    cmp    $21, %rax
    jl     Do
    jmp     End
Do:
    #true block
    jmp     While
End:
    
```

jl didn't jump, so jump out of the loop

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

56

Alternative Approach

```

While:
    cmp    $21, %rax
    jl     Do
    jmp     End
Do:
    #true block
    jmp     While
End:
    
```

Repeat the loop

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

57

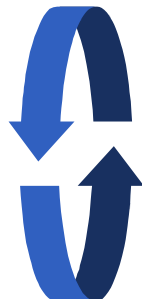


Do Loops

Test Last While Loops

Do Loops

- Programming languages also support test-last loop statements
- Many programming languages use the keyword "repeat" or "do"
- Easier than While Statements



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

59

Converting Do Loops

```

do
{
    //true block
}
while (rax < 10);
//end
    
```

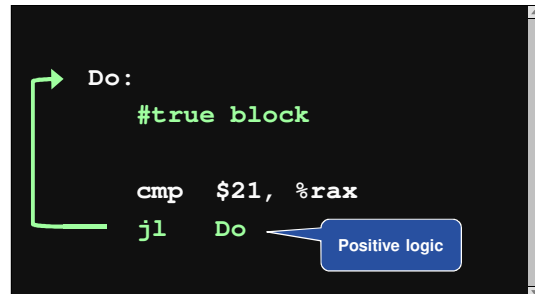
We jump UP when TRUE

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

60

Converting Do Loops



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

61

Alternative Approach

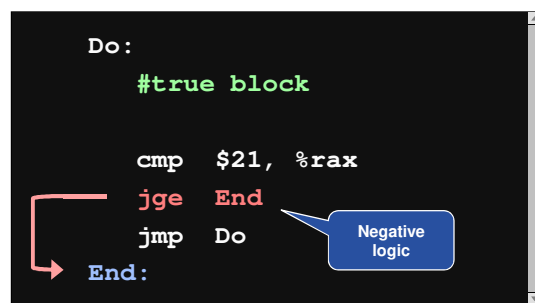
- You can also implement Do Loops using negative logic
- But it requires a few an extra label and jump statement

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

62

Alternative Approach

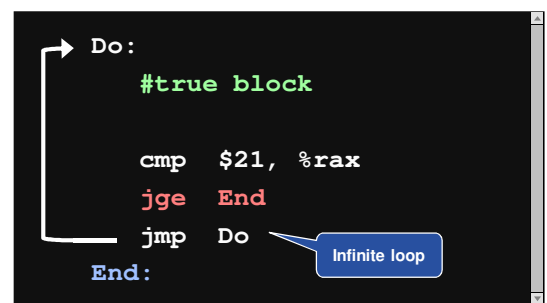


6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

63

Alternative Approach



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

64



Switch Statements on the x86

Reason for the C, Java, and C# design

Switch Statements on the x86

- You might have noticed the strange behavior of Switch statements in C, Java, and C#
- Java and C# inherited their behavior from C



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

66

Switch Statements on the x86

- C, in turn, was designed for embedded systems
- Language creates very efficient assembly code
- The Switch Statement converts easily to efficient code



6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

67

Switch Statement

- It is very efficient because...
 - it is restricted to integer constants
 - once a case is matched, no others are checked
 - they can fall through to match multiple values
- So, how?
 - start of the statement sets up just 1 register
 - compared to each "case" constant
 - jumps to a label created for each

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

68

Switch Statement Syntax

```
switch (integer)
{
    case value :
        Statements

    default:
        Statements
}
```

integer expression

You can have as many of these as needed

Executed if nothing matched

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

69

C/Java Code

```
switch (party)
{
    case 1:
        Democrat();
    case 2:
        Republican();
    default:
        ThirdParty();
}
```

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

70

Assembly Code

```
mov party, %rax
cmp $1, %rax
je case_1
cmp $2, %rax
je case_2
jmp default

case_1:
    call Democrat
case_2:
    call Republican
default:
    call ThirdParty
```

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

71

Assembly Code

```
mov party, %rax
cmp $1, %rax
je case_1
cmp $2, %rax
je case_2
jmp default

case_1:
    call Democrat
case_2:
    call Republican
default:
    call ThirdParty
```

Jump header

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

72

Assembly Code: Jump Header

```
mov party, %rax
cmp $1, %rax
je case_1
cmp $2, %rax
je case_2
jmp default
```

case 1:
case 2:
default:

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

73

Assembly Code

```
mov Party, %rax
cmp $1, %rax
je case_1
cmp $2, %rax
je case_2
jmp default
```

```
case_1:
call Democrat
case_2:
call Republican
default:
call ThirdParty
```

Case Body

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

74

Assembly Code: The Case Body

```
case_1:
call Democrat
case_2:
call Republican
default:
call ThirdParty
```

Each "falls through". They are just labels!

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

75

Fall-Through Labels

```
1
Democrat
Republican
Third Party
```

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

76

Break Statement

- Even in the last example, we still fall-through to the default
- The "Break" Statement is used exit a case
- Semantics
 - simply jumps to a label after the last case
 - so, break converts directly to a single jump

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

77

Java Code

```
switch (party)
{
case 1:
Democrat();
break;
case 2:
Republican();
break;
default:
ThirdParty();
}
```

Let's jump to the end

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

78

Assembly Code: The Cases

```
case_1:
    call Democrat
    jmp End
case_2:
    call Republican
    jmp End
default:
    call ThirdParty
End:
```

Break jumps to the end

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

79

When Fallthrough Works

- The fallthrough behavior of C was designed for a reason
- It makes it easy to combine "cases" – make a Switch Statement match multiple values
- ... and keeps the same efficient assembly code

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

80

Java Code: Primes from 1 to 10

```
switch (number)
{
    case 2:
    case 3:
    case 5:
    case 7:
        result = True;
        break;
    default:
        result = False;
}
```

Match Multiple

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

81

Primes: Jump Header

```
mov Number, %rax
cmp $2, %rax
je case_2
cmp $3, %rax
je case_3
cmp $5, %rax
je case_5
cmp $7, %rax
je case_7
jmp default
```

These are our primes

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

82

Assembly Code: The Cases

```
case_2:
case_3:
case_7:
case_9:
    movq $1, Result
    jmp End
default:
    movq $0, Result
```

All these labels will be at the same address. You, of course, would write prettier code.

6/21/2019

Sacramento State - Cook - CSc 35 - Summer 2019

83