Logic Structures in Assembly Language

XD

Peking University

2021

Outline

Basic Assembly Commands

In Lesson 5, we learnt a bunch of new features, syntax and commands. Now let's take a little time to go over them once again.

Program Counter

- Definition: A register storing the address of the next command.

Program Counter

- Definition: A register storing the address of the next command.
- Only an abstract concept; Has different implementations, like %rip(RIP) in x86.

Program Counter

- Definition: A register storing the address of the next command.
- Only an abstract concept; Has different implementations, like %rip(RIP) in x86.
- Cannot be directly written (read-only).

Program Counter

- Definition: A register storing the address of the next command.
- Only an abstract concept; Has different implementations, like %rip(RIP) in x86.
- Cannot be directly written (read-only).

Remark

Can be used to access data stored in memory with relative address difference.

Example

movq 8(%rip) %rax

Condition Codes

- Synonym: State Flag

Condition Codes

- Synonym: State Flag
- Four Condition Codes:

	Full Name	Meaning
CF	Carry Flag	unsigned overflow
ZF	Zero Flag	zero
SF	Sign Flag	negative result
OF	Overflow Flag	signed overflow

Condition Codes

- Synonym: State Flag
- Four Condition Codes:

	Full Name	Meaning
CF	Carry Flag	unsigned overflow
ZF	Zero Flag	zero
SF	Sign Flag	negative result
OF	Overflow Flag	signed overflow

- Programmers should know which sort of (signed, unsigned) data they are dealing with. No checker inside CPU.

How condition codes are set?

The state of condition code depends on the result of the last arithmetic command.

How condition codes are set?

The state of condition code depends on the result of the last arithmetic command.

Example

Suppose all the flags are 0 at first, doing
sub %rax %rax
mov %rbx %rax
only changes ZF to be 1.

How condition codes are set?

The state of condition code depends on the result of the last arithmetic command.

Example

Suppose all the flags are 0 at first, doing
sub %rax %rax
mov %rbx %rax
only changes ZF to be 1.

Remark

leaq *doesn't change the condition code.*

How condition codes are set?

The state of condition code depends on the result of the last arithmetic command.

Example

Suppose all the flags are 0 at first, doing
sub %rax %rax
mov %rbx %rax
only changes ZF to be 1.

Remark

leaq *doesn't change the condition code.*

Remark

inc *and* dec *set OF and ZF flags, but leave* CF *untouched.*

Compare

There are two commands designed explicitly for setting the condition code: cmp (compare) and test.

definition

cmpX S2 S1 ($X \in \{b, w, l, q\}$): Do S1 - S2 without storing the result.

Compare

There are two commands designed explicitly for setting the condition code: cmp (compare) and test.

definition

cmpX S2 S1 ($X \in \{b, w, l, q\}$): Do S1 - S2 without storing the result.

Remark

Used to compare the order of two (un)signed numbers.

Test

definition

testX S2 S1 ($X \in \{b, w, l, q\}$): Do S1 & S2 without storing the result.

Test

definition

testX S2 S1 ($X \in \{b, w, l, q\}$): Do S1 & S2 without storing the result.

Remark

Useful with a mask *to check if some given digits are all 0.*

Test

definition

testX S2 S1 ($X \in \{b, w, l, q\}$): Do S1 & S2 without storing the result.

Remark

Useful with a mask *to check if some given digits are all 0.*

Remark

Frequently used as testq %rax %rax to check if a number if zero. (Shorter than cmpq %rax 0!)

Conditions

- Canonical
  - n: not
  - e: equal ZF
  - z: zero ZF

Conditions

- Canonical
  - n: not
  - e: equal ZF
  - z: zero ZF
- Signed
  - g: greater $(SF \oplus OF)\&ZF$
  - l: less $SF \oplus OF$

Conditions

- Canonical
  - n: not
  - e: equal ZF
  - z: zero ZF
- Signed
  - g: greater $(SF \oplus OF)\&ZF$
  - l: less $SF \oplus OF$
- Unsigned
  - a: above $CF\& ZF$
  - b: below CF

Conditions

- Canonical
  - n: not
  - e: equal ZF
  - z: zero ZF
- Signed
  - g: greater $(SF \oplus OF)\&ZF$
  - l: less $SF \oplus OF$
- Unsigned
  - a: above $CF\& ZF$
  - b: below CF

Remark

n *is used as a prefix to negate the conditions that follow.*

Remark

e *can be used as a suffix to a,b,g,l representing 'or equal'.*

Set

We need a way to retrieve and manipulate the condition codes.

definition

setX D (X is a condition): moving the corresponding combination of condition codes into the lower-order byte of D.

Remark

Set doesn't clear the high-order bytes.

Jumps

To implement more complicated structures while minimizing the size of the assembly program, jumps are introduced.

definition

jmp X: jump to an memory address specified by X (can be a label)
jmp *R: jump to the memory address specified in register R
jmp *(R): jump to the memory address specified in the memory specified by register R

Jumps

To implement more complicated structures while minimizing the size of the assembly program, jumps are introduced.

definition

jmp X: jump to an memory address specified by X (can be a label)
jmp *R: jump to the memory address specified in register R
jmp *(R): jump to the memory address specified in the memory specified by register R

Remark

PC-relative encoding have two advantages:

- *more compactly encoded instruction*
- *more portable code*

Jump Tables

We can also use a constant table to refer to what's the jump destination.

definition

An anchor label and an alignment specification.

Jump Tables

We can also use a constant table to refer to what's the jump destination.

definition

An anchor label and an alignment specification.

structure

```
.section .rodata /* stands for read-only data */
.align 8 /* specifies the alignment used in the table */
.L4: /* the anchor label */
.quad .L8
.quad .L2
...
```

usage

jmp .L2(,%rax,8): jump to the label specified by %rax

Jump Tables

We can also use a constant table to refer to what's the jump destination.

definition

An anchor label and an alignment specification.

structure

.section .rodata /* stands for read-only data */
.align 8 /* specifies the alignment used in the table */
.L4: /* the anchor label */
.quad .L8
.quad .L2
...

usage

jmp .L2(,%rax,8): jump to the label specified by %rax

Remark

Jump tables help perform jumps automatically.

Conditional Jumps

To harness the power of condition codes, conditional jumps are introduced.

definition

jX D (X is a condition): jump to the address specified by D if X is satisfied.

Remark

Conditional jump are the reason why we could implement logic structure.

rep&repz

In the assembly code provided in the CSAPP textbook and the bomb lab, sometimes **ret** is written as **rep ret** or **repz ret**.
So, why are they there in the first place?

rep&repz

In the assembly code provided in the CSAPP textbook and the bomb lab, sometimes **ret** is written as **rep ret** or **repz ret**.
So, why are they there in the first place?

Answer

They are only used for AMD CPUs since these CPUs cannot handle the return address if ret is reached from a conditional jump. In another word, rep and repz are meaningless occupiers that could avoid this situation from happening.

Condition Moves

Before diving into details why we introduce conditional moves, we first review the definition of conditional moves(cmov).

definition

cmovSX R1 R2 (S is the size and X is a condition): if X is satisfied, move data from R1 to R2.
R1, R2 cannot both be memory address.

Example

```
movq %rdi, %rax
subq %rsi, %rax
movq %rsi, %rdx
subq %rdi, %rdx
cmpq %rsi, %rdi
cmovge %rdx, %rax
ret
```

Condition Moves

So, why use cmov?

Condition Moves

So, why use cmov?

- Advantages
  - Save space
  - Enhanced performance on pipelined CPUs (while jmp perform terribly) due to **condition code transfers**.

Remark

Pipelined CPU achieves high performance by overlapping the steps of the successive instructions.

Condition Moves

So, why use cmov?

- Advantages
  - Save space
  - Enhanced performance on pipelined CPUs (while jmp perform terribly) due to **condition code transfers**.
- Disadvantages
  - May break the structure of assembly code
  - Increase the calculation workload
  - Forbidden in some cases

Remark

Pipelined CPU achieves high performance by overlapping the steps of the successive instructions.

Condition Moves

So, why use cmov?

- Advantages
  - Save space
  - Enhanced performance on pipelined CPUs (while jmp perform terribly) due to **condition code transfers**.
- Disadvantages
  - May break the structure of assembly code
  - Increase the calculation workload
  - Forbidden in some cases
- Bad cases
  - Expensive evaluations: function-involved
  - Risky computation: p ? *p : 0
  - Computations with side effects: +=, *=

Remark

Pipelined CPU achieves high performance by overlapping the steps of the successive instructions.

if

Implementation

cmov, conditional jumps

Example

Implement a function that return abs of the first argument.

Answer

mov %rdi %rax test %rax %rax
jge .L1
sub $0 %rax
.L1:
repz ret

for, while

Implementation

conditional jumps

Example

Implement a loop using %rdi and %rsi where the outer i iterates from 0 to 9 and the inner one iterates from i to 9

for,while

Answer

```
mov $0 %rdi
jmp .CHECK1
.LOOP1: mov %rdi %rsi
jmp .CHECK2
.LOOP2:
... //do stuff
.CHECK2:
cmp %rsi $9
jle .LOOP2
.CHECK1:
cmp %rdi $9
jle .LOOP1
```

switch

Implementation

jump tables

Example

Implement a function on a 64x machine to jump to .L1 when %rax is prime and .L0 when it is not. ($\%rax \in \{1, 2, 3, 4\}$)

Answer

```
.section .rodata
.align 8
.ANCHOR:
.quad .L0
.quad .L1
.quad .L1
.quad .L0
...
jmp .ANCHOR(,%rax,$8)
```

Exercises

Implemented on a x86-64 AMD machine.

```
function f(int a, int b, char flag){
return flag?abs(a-b):-abs(a-b);
}
pushq %rbp
__(1)__ %dl %rdx
__(2)__ %rsi %rdi
__(3)__ %rbp
movzbq %bpl %rbp
__(4)__ %rbp %rdx
jnz .L1
__(2)__ %rsi %rdi
__(5)__ %rbp
movzbq %bpl %rbp
__(6)__ %rdx %rbp
__(7)__ .L1
mov %rdi %rax
sub __(8)__ __(9)__
```

Exercises

```
.L1:
mov __(10)__ %rax
sub __(11)__ __(12)__
.RET:
pop %rbp
__(13)__ ret
```