Computer Architecture
Machine Representation, Fundamentals and Control

Cain Susko

Queen's University
School of Computing

February 13, 2022

Complete Memory Addressing Modes

`D` Constant Displacement

`Rb` Base register: any of the 16 registers

`Ri` Index register: any register except `%rsp`

`S` Scale: 1, 2, 4, 8 (only these numbers)

There are also some special cases:

(Rb,Ri)	**Mem[Reg[Rb]+Reg[Ri]]**
D(Rb,Ri)	**Mem[Reg[Rb]+Reg[Ri]+D]**
(Rb,Ri,S)	**Mem[Reg[Rb]+S*Reg[Ri]]**

Here are some examples of address computation.

`%rdx`	`0xf000`
`%rcx`	`0x0100`

Expression	Address Computation	Address
`0x8(%rdx)`	`0xf000 + 0x8`	`0xf008`
`(%rdx,%rcx)`	`0xf000 + 0x100`	`0xf100`
`(%rdx,%rcx,4)`	`0xf000 + 4*0x100`	`0xf400`
`0x80(,%rdx,2)`	`2*0xf000 + 0x80`	`0x1e080`

Arithmetic and Logic Operations

Address Computation Instruction Is denoted by `leaq` *`src dest`*

Where *src* is the address mode expression and sets *dest* to address denoted by expression.

It can be used to compute memory addresses without a memory reference (In C this would be equiv. to `p = &x[i]`

It can also be used for computing arithmetic expressions of the form $x + k * y$ where $k = \{1, 2, 4, 8\}$. For example:

```
long m12(long x)
{
  return x*12;
}
```

Converted to ASM by compiler:

```
leaq (%rdi,%rdi,2), %rax # t <- x+x*2
salq $2, %rax            # return t<<2
```

There are also a host of other more common arithmetic operations: Note

Format	Computation	
addq	Src, Dest	Dest = Dest + Src
subq	Src, Dest	Dest = Dest – Src
imulq	Src, Dest	Dest = Dest * Src
salq	Src, Dest	Dest = Dest << Src
sarq	Src, Dest	Dest = Dest >> Src
shrq	Src, Dest	Dest = Dest >> Src
xorq	Src, Dest	Dest = Dest ^ Src
andq	Src, Dest	Dest = Dest & Src
orq	Src, Dest	Dest = Dest | Src

that numbers are in 2's complement and **Watch Out** for the argument order. Also, there is no distinction between signed and unsigned integers.

Here are some examples of Arithmetic operations with addresses:

```
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;
}
```

```
leaq    (%rdi,%rsi), %rax   # t1
addq    %rdx, %rax          # t2
leaq    (%rsi,%rsi,2), %rdx
salq    $4, %rdx            # t4
leaq    4(%rdi,%rdx), %rcx  # t5
imulq   %rcx, %rax          # rval
ret
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rdx	Argument z
%rax	t1, t2, rval
%rdx	t4
%rcx	t5

Summary - so far

- The History of Intel Processors / x86
- System assembly and machine code
- Assembly basics: registers, operands, etc.
- Arithmetic

Control

Control aka Condition Codes are one of the four following:

$$CF \;\; ZF \;\; SF \;\; OF.$$

They are also known as status flags. These control the flow of computation. They each do the following:

CF Carry Flag (for unsigned)

SF Sign Flag (for signed)

ZF Zero Flag

OF Overflow Flag (for signed)

Implicitly Set Condition Codes

Given the example `addq src dest` $\leftrightarrow$ `t = a+b` These flags are implicitly set in arithmetic operations:

- CF is set if there is an unsigned overflow (carry out most significant bit)
- ZF is set if `t == 0`
- SF is set if `t<0` (as signed)
- OF is set if two's complement overflow (signed overflow)

Note: that these are not set by the `leaq` instruction

Explicitly Set Condition Codes

We can set the flags explicitly by using `cmpq Src1 Src2`, which is like computing $a - b$ without setting a destination, This will cause the following flags:

- CF set if carry out from most significant bit (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

We can also explicitly set the flags by using the `testq Src1 Src2`, which is like computing $a\&\&b$ without setting a destination. This will cause the following:

- 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 We can use `SetX` instructions to set the low order byte destination to 1 or 0 based on the combinations of condition codes. They do note alter the remaining 7 bytes and are comprised of the following:

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)

Note: typically use `movzbl` to finish the job.
Below are some examples.

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

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
```

However, While this technique is useful, it is not how we will normally change flags. We will explore this in the next lesson