Lecture 12

Arithmetic and Logical Operations

CPSC 275
Introduction to Computer Systems

Complete Memory Addressing Modes

- Most General Form
  D(Rb,Ri,S) Mem[Reg[Rb] + S * Reg[Ri] + D]
  - D: Constant "displacement" 1, 2, or 4 bytes
  - Rb: Base register: Any of 8 integer registers
  - Ri: Index register: Any, except for `%esp`
  - S: Scale: 1, 2, 4, or 8
- 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]]

Address Computation Examples

`%edx`	`0xf000`
`%ecx`	`0x0100`

Expression	Address Computation	Address
`0x8(%edx)`		
`(%edx,%ecx)`		
`(%edx,%ecx,4)`		
`0x80(,%edx,2)`		

Address Computation Instruction

- `leal` *src,dest*
  - *Load effective address*
  - *src* is address mode expression
  - Set *dest* to address denoted by expression
- Uses
  - Computing addresses without a memory reference
    - E.g., translation of `p = &x[i];`
  - Computing arithmetic expressions
    - Example

```
int mul12(int x)
{
  return x*12;
}
```

Converted to ASM by compiler:

```
leal (%eax,%eax,3),%eax  # t <- x + x*3
sall $2, %eax            # return t<<2
```

Some Arithmetic Operations

- Two Operand Instructions:

Format		*Computation*
`addl`	*Src,Dest*	Dest = Dest + Src
`subl`	*Src,Dest*	Dest = Dest - Src
`imull`	*Src,Dest*	Dest = Dest * Src
`sall`	*Src,Dest*	Dest = Dest << Src
`sarl`	*Src,Dest*	Dest = Dest >> Src
`shrl`	*Src,Dest*	Dest = Dest >> Src
`xorl`	*Src,Dest*	Dest = Dest ^ Src
`andl`	*Src,Dest*	Dest = Dest & Src
`orl`	*Src,Dest*	Dest = Dest | Src

- Watch out for argument order!
- No distinction between signed and unsigned int

Some Arithmetic Operations

- One Operand Instructions

`incl`	*Dest*	*Dest = Dest + 1*
`decl`	*Dest*	*Dest = Dest - 1*
`negl`	*Dest*	*Dest = - Dest*
`notl`	*Dest*	*Dest = ~Dest*

Arithmetic Expression Example

```
int arith(int x, int y,
          int z)
{
   int t1 = x + y;
   int t2 = z + t1;
   int t3 = x + 4;
   int t4 = y * 48;
   int t5 = t3 + t4;
   int rval = t2 * t5;
   return rval;
}
```

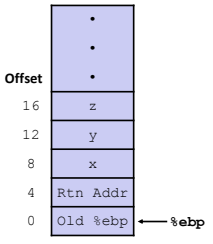
```
arith:
   pushl  %ebp          } Set
   movl   %esp, %ebp    } Up

   movl   8(%ebp), %ecx
   movl   12(%ebp), %edx
   leal   (%edx,%edx,2), %eax
   sall   $4, %eax
   leal   4(%ecx,%eax), %eax     } Body
   addl   %ecx, %edx
   addl   16(%ebp), %edx
   imull  %edx, %eax

   popl   %ebp          } Finish
   ret
```

Understanding `arith`

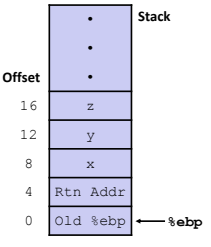
```
int arith(int x, int y, int z)
{
   int t1 = x + y;
   int t2 = z + t1;
   int t3 = x + 4;
   int t4 = y * 48;
   int t5 = t3 + t4;
   int rval = t2 * t5;
   return rval;
}
```




```
movl    8(%ebp), %ecx
movl    12(%ebp), %edx
leal    (%edx,%edx,2), %eax
sall    $4, %eax
leal    4(%ecx,%eax), %eax
addl    %ecx, %edx
addl    16(%ebp), %edx
imull   %edx, %eax
```

Understanding `arith`

```
int arith(int x, int y, int z)
{
   int t1 = x + y;
   int t2 = z + t1;
   int t3 = x + 4;
   int t4 = y * 48;
   int t5 = t3 + t4;
   int rval = t2 * t5;
   return rval;
}
```




```
movl    8(%ebp), %ecx          # ecx = x
movl    12(%ebp), %edx         # edx = y
leal    (%edx,%edx,2), %eax    # eax = y*3
sall    $4, %eax               # eax *= 16 (t4)
leal    4(%ecx,%eax), %eax     # eax = t4 + x + 4 (t5)
addl    %ecx, %edx             # edx = x + y (t1)
addl    16(%ebp), %edx         # edx += z (t2)
imull   %edx, %eax             # eax = t2 * t5 (rval)
```

Observations about `arith`

```
int arith(int x, int y, int z)
{
   int t1 = x + y;
   int t2 = z + t1;
   int t3 = x + 4;
   int t4 = y * 48;
   int t5 = t3 + t4;
   int rval = t2 * t5;
   return rval;
}
```

- Instructions in different order from C code
- Some expressions require multiple instructions
- Some instructions cover multiple expressions

```
movl    8(%ebp), %ecx          # ecx = x
movl    12(%ebp), %edx         # edx = y
leal    (%edx,%edx,2), %eax    # eax = y*3
sall    $4, %eax               # eax*= 16 (t4)
leal    4(%ecx,%eax), %eax     # eax = t4 +x+4 (t5)
addl    %ecx, %edx             # edx = x+y (t1)
addl    16(%ebp), %edx         # edx += z (t2)
imull   %edx, %eax             # eax = t2 * t5 (rval)
```

Practice Problems

- Read CSaPP Sec. 3.5.1-3.5.4 and try the following problems:

  3.6, 3.7, 3.8, 3.9, 3.10