

CS 47

Midterm 1 Review

Topics

- Integer Arithmetic
- Floating Point
- Machine Instructions

class13.ppt

CS 47 Spring 2008

Integer Arithmetic

Addition is associative and commutative

Multiplication is associative and commutative and distributes over addition.

Answers are computed modulo 2^w , where w is the word size, 32 for most machines, often 8 or less for hand computation.

Compute the right answer mathematically, and adjust it by a multiple of 2^w to bring it into the proper range.

Unsigned range = $[0 \text{ to } 2^w-1]$

Signed range = $[-2^{w-1} \text{ to } 2^{w-1}-1]$

- 2 -

CS 47 Spring, 2008

Integer Arithmetic

Most tricky behavior happens around the wrap-around values ($2^w \rightarrow 0$) for unsigned and ($2^{w-1} \rightarrow -2^{w-1}$) for two's complement.

Two's complement: $-x = \sim x + 1 = \sim(x - 1)$ = invert all bits except the last 1.

Most ordering properties of integers can fail mod 2^w

$x > y \quad \Rightarrow \quad -x < -y \quad (y = \text{TMIN})$

$x * x \geq 0 \quad (x = \text{Ceil}(\text{Sqrt}(\text{TMAX})))$

Most algebraic properties hold

$(x - y)^2 == x^2 - 2 * x * y + y^2 \quad (\text{True})$

- 3 -

CS 47 Spring, 2008

Problem 2.49

The problem asks us to compute the full $2w$ -bit product of w -bit unsigned int's x' and y' on a machine that can do $2w$ -bit products only in two's complement form. Let $x = (\text{int}) x'$ and $y = (\text{int}) y'$.

We already know the low-order w bits of the product $x' * y'$ are the same as the low-order w bits of $x * y$.

By Equation 2.16 we have $x' = x + x_{w-1}2^w$ and

$$x' \cdot y' = x \cdot y + (x_{w-1}y + y_{w-1}x)2^w + x_{w-1}y_{w-1}2^{2w}.$$

The final term has no effect on the $2w$ -bit representation of $x' \cdot y'$, but the middle term represents a correction factor that must be added to the high order w bits.

- 4 -

CS 47 Spring, 2008

Example 1

The idea here is that the unsigned_high_prod is gotten from signed_high_prod by adding $2^w (x_{w-1}y + y_{w-1}x)$. That means if $y < 0$ add x and if $x < 0$ add y to the high order w bits, where x and y are the two's complement values.

Example 1: $w = 8$

$$x = 7C = 124$$

$$y = 83 = -125$$

Remembering to use sign extension, $x = 007C$, $y = FF83$

$$x * y = C374 = -15500$$

$$x' = 7C = 124$$

$$y' = 83 = 131$$

$$x' * y' = C374 + 7C00 = 3F74 = 16244$$

Example 2

Example 2: $w = 8$

$$x = FD = -3$$

$$y = 83 = -125$$

Remembering to use sign extension, $x = FFFD$, $y = FF83$

$$x * y = 0177 = 375$$

$$x' = FD = 253$$

$$y' = 83 = 131$$

$$x' * y' = 0177 + FD00 + 8300 = 8177 = 33143$$

C solution

This is implemented as follows:

```
1 unsigned unsigned_high_prod(unsigned x, unsigned y)
2 {
3     unsigned p = (unsigned) signed_high_prod((int) x, (int) y);
4
5     if ((int) x < 0)          /* x_{w-1} = 1 */
6         p += y;
7     if ((int) y < 0)          /* y_{w-1} = 1 */
8         p += x;
9     return p;
10 }
```

- 7 -

CS 47 Spring, 2008

Floating Point Formats

Sign, exp, fraction

| | | | |
|---|---|----|-------|
| 1 | 8 | 23 | float |
|---|---|----|-------|

| | | | |
|---|----|----|--------|
| 1 | 11 | 52 | double |
|---|----|----|--------|

$$\text{Val} = (-1)^s \times 2^E \times M$$

$$E = \text{exp} - \text{bias} \quad \text{bias} = 2^{\text{expdigits}-1} - 1 \text{ (127 or 1023)}$$

$$M = 1 + \text{fraction} \quad \text{if exp} \neq 0$$

$$M = 0 + \text{fraction} \quad \text{if exp} == 0$$

$$0 \leq \text{fraction} < 1$$

- 8 -

CS 47 Spring, 2008

Floating Point Formats

Sign, exp, fraction

| | | | | |
|---|---|----|--|--|
| 1 | 8 | 23 | | |
|---|---|----|--|--|

float

| | | | | |
|---|----|----|--|--|
| 1 | 11 | 52 | | |
|---|----|----|--|--|

double

**Inf = ∞ has exp = $2^{\text{expdigits}} - 1$ = all 1's (255 or 2047)
and fraction == 0.**

NaN has same exp and fraction != 0

Problem 2.33

| Bits | exp | E | frac | M | value |
|---------|-----|---|------|-----|-------|
| 0 00 00 | 0 | 0 | 0 | 0 | 0 |
| 0 00 01 | 0 | 0 | 1/4 | 1/4 | 1/4 |
| 0 00 10 | 0 | 0 | 2/4 | 2/4 | 2/4 |
| 0 00 11 | 0 | 0 | 3/4 | 3/4 | 3/4 |
| 0 01 00 | 1 | 0 | 0 | 4/4 | 4/4 |
| 0 01 01 | 1 | 0 | 1/4 | 5/4 | 5/4 |
| 0 01 10 | 1 | 0 | 2/4 | 6/4 | 6/4 |
| 0 01 11 | 1 | 0 | 3/4 | 7/4 | 7/4 |

Problem 2.33 (cont.)

| Bits | exp | E | frac | M | value |
|---------|-----|---|------|-----|----------|
| 0 10 00 | 2 | 1 | 0 | 4/4 | 8/4 |
| 0 10 01 | 2 | 1 | 1/4 | 5/4 | 10/4 |
| 0 10 10 | 2 | 1 | 2/4 | 6/4 | 12/4 |
| 0 10 11 | 2 | 1 | 3/4 | 7/4 | 14/4 |
| 0 11 00 | 3 | - | 0 | - | ∞ |
| 0 11 01 | 3 | - | 1/4 | - | NaN |
| 0 11 10 | 3 | - | 2/4 | - | NaN |
| 0 11 11 | 3 | - | 3/4 | - | NaN |

- 11 -

CS 47 Spring, 2008

Floating Point Arithmetic

Addition is commutative but not always associative

Multiplication is commutative and not associative and does not distribute over addition.

Answers are rounded to the floating point value nearest to the exact answer. Round to even in case of ties.

Most tricky behavior happens when rounding errors depend on the order of the operations.

Every value has an exact negative: $x = -(-x)$

- 12 -

CS 47 Spring, 2008

Floating Point Arithmetic

Most ordering properties of reals hold for floats.

$d < 0.0 \Rightarrow ((d*2) < 0.0)$ **True**

$d > f \Rightarrow -f > -d$ **True**

$a + b + c == c + (a + b)$ **Yes! commutative**

Many algebraic properties of reals don't hold for floats.

$(d + f) - d == f$ **No: Not associative**

$a + b + c == c + a + b$ **No: Not associative**

Machine Level Programs

Topics

instructions

mov, lea, push, pop

add, sub, imul, and, or, xor, shl, sar, slr

not, neg, inc, dec

addressing

src, dest Imm(E_b , E_i , s) or E_b

control

test, cmp, set, jmp, je, jne, jg, etc.

procedures

call, ret, leave

Example of Assembly Code from C

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

} Set Up

} Body

} Finish

- 15 -

CS 47 Spring, 2008

Address Computation Examples

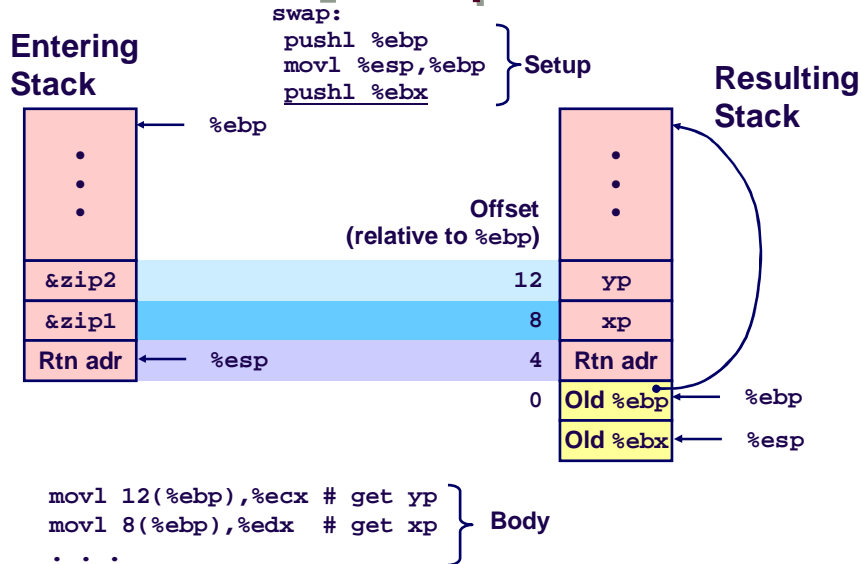
| | |
|------|--------|
| %edx | 0xf000 |
| %ecx | 0x100 |

| Expression | Computation | Address |
|---------------|------------------|---------|
| 0x8(%edx) | 0xf000 + 0x8 | 0xf008 |
| (%edx,%ecx) | 0xf000 + 0x100 | 0xf100 |
| (%edx,%ecx,4) | 0xf000 + 4*0x100 | 0xf400 |
| 0x80(,%edx,2) | 2*0xf000 + 0x80 | 0x1e080 |

- 16 -

CS 47 Spring, 2008

Effect of swap Setup



- 17 -

CS 47 Spring, 2008

Practice Problems

Study suggestions

Practice Problems 3.9, 3.10, 3.11, 3.12 are all similar. They ask you to find registers and memory locations corresponding to C variables and machine instructions corresponding to C operators.

Sometimes you need to fill in C code for given assembly code or vice versa.

Answers are given in the back of the chapter. If you can do these problems without peeking you will be well prepared for this part of the exam.

- 18 -

CS 47 Spring, 2008