CS 33

Data Representation (Part 4)
Floating-Point Operations: Basic Idea

• \( x +_f y = \text{Round}(x + y) \)

• \( x \times_f y = \text{Round}(x \times y) \)

• Basic idea
  – first compute exact result
  – make it fit into desired precision
    » possibly overflow if exponent too large
    » possibly round to fit into \( \text{frac} \)
## Rounding

- **Rounding modes (illustrated with $ rounding)**

<table>
<thead>
<tr>
<th></th>
<th>$1.40</th>
<th>$1.60</th>
<th>$1.50</th>
<th>$2.50</th>
<th>$−1.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>towards zero</td>
<td>$1</td>
<td>$1</td>
<td>$1</td>
<td>$2</td>
<td>$−1</td>
</tr>
<tr>
<td>round down (−∞)</td>
<td>$1</td>
<td>$1</td>
<td>$1</td>
<td>$2</td>
<td>$−2</td>
</tr>
<tr>
<td>round up (+∞)</td>
<td>$2</td>
<td>$2</td>
<td>$2</td>
<td>$3</td>
<td>$−1</td>
</tr>
<tr>
<td>nearest integer</td>
<td>$1</td>
<td>$2</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>nearest even (default)</td>
<td>$1</td>
<td>$2</td>
<td>$2</td>
<td>$2</td>
<td>$−2</td>
</tr>
</tbody>
</table>
Creating a Floating Point Number

• **Steps**
  – normalize to have leading 1
  – round to fit within fraction
  – postnormalize to deal with effects of rounding

• **Case study**
  – convert 8-bit unsigned numbers to tiny floating-point format

<table>
<thead>
<tr>
<th>Example Number</th>
<th>Binary</th>
<th>Sci Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>100000000</td>
<td>1.0000000E+07</td>
</tr>
<tr>
<td>13</td>
<td>00001101</td>
<td>1.1101E+00</td>
</tr>
<tr>
<td>33</td>
<td>00010001</td>
<td>1.0010001E+00</td>
</tr>
<tr>
<td>35</td>
<td>00010011</td>
<td>1.0010011E+00</td>
</tr>
<tr>
<td>138</td>
<td>10001010</td>
<td>1.1001010E+01</td>
</tr>
<tr>
<td>63</td>
<td>0011111</td>
<td>1.111111E+01</td>
</tr>
</tbody>
</table>
Normalize

- Requirement
  - set binary point so that numbers of form 1.xxxxx
  - adjust all to have leading one
    » decrement exponent as shift left

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
<th>Fraction</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>10000000</td>
<td>1.00000000</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>00001101</td>
<td>1.10100000</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>00010001</td>
<td>1.0001000</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>00010011</td>
<td>1.0011000</td>
<td>4</td>
</tr>
<tr>
<td>138</td>
<td>10001010</td>
<td>1.0001010</td>
<td>7</td>
</tr>
<tr>
<td>63</td>
<td>00111111</td>
<td>1.11111000</td>
<td>5</td>
</tr>
</tbody>
</table>
Rounding

- **Round-up conditions**
  - $\text{round} = 1$, $\text{sticky} = 1 \Rightarrow > 0.5$
  - $\text{guard} = 1$, $\text{round} = 1$, $\text{sticky} = 0 \Rightarrow$ round up to even

<table>
<thead>
<tr>
<th>Value</th>
<th>Fraction</th>
<th>GRS</th>
<th>Incr?</th>
<th>Rounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.00000000</td>
<td>000</td>
<td>N</td>
<td>1.000</td>
</tr>
<tr>
<td>13</td>
<td>1.10100000</td>
<td>100</td>
<td>N</td>
<td>1.101</td>
</tr>
<tr>
<td>17</td>
<td>1.00010000</td>
<td>010</td>
<td>N</td>
<td>1.000</td>
</tr>
<tr>
<td>19</td>
<td>1.00110000</td>
<td>110</td>
<td>Y</td>
<td>1.010</td>
</tr>
<tr>
<td>138</td>
<td>1.00010100</td>
<td>011</td>
<td>Y</td>
<td>1.001</td>
</tr>
<tr>
<td>63</td>
<td>1.11111000</td>
<td>111</td>
<td>Y</td>
<td>10.000</td>
</tr>
</tbody>
</table>
Postnormalize

- **Issue**
  - rounding may have caused overflow
  - handle by shifting right once & incrementing exponent

<table>
<thead>
<tr>
<th>Value</th>
<th>Rounded</th>
<th>Exp</th>
<th>Adjusted</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.000</td>
<td>7</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>13</td>
<td>1.101</td>
<td>3</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>17</td>
<td>1.000</td>
<td>4</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>19</td>
<td>1.010</td>
<td>4</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>138</td>
<td>1.001</td>
<td>7</td>
<td></td>
<td>134</td>
</tr>
<tr>
<td>63</td>
<td>10.000</td>
<td>5</td>
<td>1.000*2^6</td>
<td>64</td>
</tr>
</tbody>
</table>
Floating-Point Multiplication

- \((-1)^{s_1} M_1 \ 2^{E_1} \times (-1)^{s_2} M_2 \ 2^{E_2}\)

- **Exact result:** \((-1)^s \ M \ 2^E\)
  - **sign s:** \(s_1 \wedge s_2\)
  - **significand M:** \(M_1 \times M_2\)
  - **exponent E:** \(E_1 + E_2\)

- **Fixing**
  - if \(M \geq 2\), shift \(M\) right, increment \(E\)
  - if \(E\) out of range, overflow (or underflow)
  - round \(M\) to fit \(\frac{\text{precision}}{}\)

- **Implementation**
  - biggest chore is multiplying significands
Floating-Point Addition

• \((-1)^{s_1} M_1 \ 2^{E_1} + (-1)^{s_2} M_2 \ 2^{E_2}\)
  – assume \(E_1 > E_2\)

• **Exact result:** \((-1)^{s} M \ 2^{E}\)
  – sign \(s\), significand \(M\):
    » result of signed align & add
  – exponent \(E\): \(E_1\)

• **Fixing**
  – if \(M \geq 2\), shift \(M\) right, increment \(E\)
  – if \(M < 1\), shift \(M\) left \(k\) positions, decrement \(E\) by \(k\)
  – overflow if \(E\) out of range
  – round \(M\) to fit \(\text{frac}\) precision
Floating Point

• Single precision (float)

\[
\begin{array}{c|cc}
\text{s} & \text{exp} & \text{frac} \\
1 & 8\text{-bits} & 23\text{-bits} \\
\end{array}
\]

– range: \( \pm 1.8 \times 10^{-38} - \pm 3.4 \times 10^{38} \), \(~7\) decimal digits

• Double Precision (double)

\[
\begin{array}{c|cc}
\text{s} & \text{exp} & \text{frac} \\
1 & 11\text{-bits} & 52\text{-bits} \\
\end{array}
\]

– range: \( \pm 2.23 \times 10^{-308} - \pm 1.8 \times 10^{308} \), \(~16\) decimal digits
Floating Point in C

• Conversions/casting
  –casting between int, float, and double changes bit representation
  –double/float → int
    » truncates fractional part
    » like rounding toward zero
    » not defined when out of range or NaN: generally sets to Tmin
  –int → double
    » exact conversion, as long as int has ≤ 53-bit word size
  –int → float
    » will round according to rounding mode
Suppose $f$, declared to be a `float`, is assigned the largest possible floating-point positive value (other than $+\infty$). What is the value of $g = f + 1.0$?

a) $f$
b) $+\infty$
c) NaN
d) 0
Float is not Rational …

- Floating addition
  - commutative: \( a +^f b = b +^f a \)
    » yes!
  - associative: \( a +^f (b +^f c) = (a +^f b) +^f c \)
    » no!
  - \( 2 +^f (1e38 +^f -1e38) = 2 \)
  - \( (2 +^f 1e38) +^f -1e38 = 0 \)
Float is not Rational …

- **Multiplication**
  - commutative: $a \times f b = b \times f a$
    » yes!
  - associative: $a \times f (b \times f c) = (a \times f b) \times f c$
    » no!
  - $1e37 \times f (1e37 \times f 1e-37) = 1e37$
  - $(1e37 \times f 1e37) \times f 1e-37 = +\infty$
Float is not Rational …

- More …
  - multiplication distributes over addition:
    \[ a \times b \times (c + d) = (a \times b + c \times d) \]
  - no!
    \[ 1 \times (1 \times -1) = 0 \]
    \[ (1 \times 1) + (1 \times -1) = \text{NaN} \]
  - loss of significance:
    \[ x = y + 1 \]
    \[ z = 2 / (x - y) \]
    \[ z == 2? \]
    - not necessarily!
      - consider \( y = 1e38 \)
CS 33

Intro to Machine Programming
Machine Model

Processor (aka CPU)  Memory (aka RAM)

instructions and data

data
Memory

Instructions

Data

or

Instructions are Data
Processor: Some Details

Execution engine

Instruction pointer

Condition codes
Processor: Basic Operation

while (forever) {
    fetch instruction IP points at
    decode instruction
    fetch operands
    execute
    store results
    update IP and condition code
}
Instructions ...

<table>
<thead>
<tr>
<th>Op code</th>
<th>Operand1</th>
<th>Operand2</th>
<th>...</th>
</tr>
</thead>
</table>

Operands

• Form
  – immediate vs. reference
    » value vs. address

• How many?
  – 3
    » add a,b,c
      • c = a + b
  – 2
    » add a,b
      • b += a
Operands (continued)

• Accumulator
  – special memory in the processor
    » known as a *register*
    » fast access
  – allows single-operand instructions
    » add a
      • acc += a
    » add b
      • acc += b
From C to Assembler ...

\[ a = (b + c) \times d; \]

\begin{align*}
&\text{mov} \quad b, \%\text{acc} \\
&\text{add} \quad c, \%\text{acc} \\
&\text{mul} \quad d, \%\text{acc} \\
&\text{mov} \quad \%\text{acc}, a
\end{align*}

\text{if} \ (a < b)
\begin{align*}
&c = 1; \\
\text{else}
&d = 1;
\end{align*}

\begin{align*}
&\text{cmp} \quad a, b \\
&\text{jge} \quad .L1 \\
&\text{mov} \quad \$1, c \\
&\text{jmp} \quad .L2 \\
.L1
&\text{mov} \quad \$1, d \\
.L2
\end{align*}

immediate operand

immediate operand
Condition Codes

• Set of flags giving status of most recent operation:
  – zero flag
    » result was zero
  – sign flag
    » for signed arithmetic interpretation: sign bit is set
  – overflow flag
    » for signed arithmetic interpretation
  – carry flag (generated by carry or borrow out of most-significant bit)
    » for unsigned arithmetic interpretation

• Set implicitly by arithmetic instructions
• Set explicitly by compare instruction
  – cmp a,b
    » sets flags based on result of b-a
Examples (1)

• Assume 32-bit arithmetic

• x is 0x80000000
  – TMIN if interpreted as two’s-complement
  – $2^{31}$ if interpreted as unsigned

• x-1 (0x7fffffff)
  – TMAX if interpreted as two’s-complement
  – $2^{31}-1$ if interpreted as unsigned
  – zero flag is not set
  – sign flag is not set
  – overflow flag is set
  – carry flag is not set
Examples (2)

- x is 0xffffffff
  - -1 if interpreted as two’s-complement
  - UMAX (2^{32}-1) if interpreted as unsigned
- x+1 (0x00000000)
  - zero under either interpretation
  - zero flag is set
  - sign flag is not set
  - overflow flag is not set
  - carry flag is set
Examples (3)

• x is 0xffffffff
  – -1 if interpreted as two’s-complement
  – UMAX (2^{32}-1) if interpreted as unsigned

• x+2 (0x00000001)
  – (+)1 under either interpretation
  – zero flag is not set
  – sign flag is not set
  – overflow flag is not set
  – carry flag is set
Quiz 2

• Set of flags giving status of most recent operation:
  – zero flag
    » result was zero
  – sign flag
    » for signed arithmetic interpretation: sign bit is set
  – overflow flag
    » for signed arithmetic interpretation
  – carry flag (generated by carry or borrow out of most-significant bit)
    » for unsigned arithmetic interpretation

• Set explicitly by compare instruction
  – cmp a,b
    » sets flags based on result of b-a

Which flags are set to one by “cmp 2,1”?

a) overflow flag only
b) carry flag only
c) sign and carry flags only
d) sign and overflow flags only
e) sign, overflow, and carry flags
Jump Instructions

• Unconditional jump
  – just do it

• Conditional jump
  – to jump or not to jump determined by condition-code flags
  – field in the op code indicates how this is computed
  – in assembler language, simply say
    » je
      • jump on equal
    » jne
      • jump on not equal
    » jgt
      • jump on greater than
    » etc.
Addresses

```c
int a, b, c, d;

int main() {
    a = (b + c) * d;
    ...
}
```

| Memory | 1012: | d |
|        | 1008: | c |
|        | 1004: | b  | global |
|        | 1000: | a  | variables |

mov b, %acc
add c, %acc
mul d, %acc
mov %acc, a

mov 1004, %acc
add 1008, %acc
mul 1012, %acc
mov %acc, 1000
Addresses

```c
int b;

int func(int c, int d) {
    int a;
    a = (b + c) * d;
    ...
}
```

- One copy of `b` for duration of program’s execution
  - `b`’s address is the same for each call to `func`
- Different copies of `a`, `c`, and `d` for each call to `func`
  - addresses are different in each call
Relative Addresses

- **Absolute address**
  - actual location in memory
- **Relative address**
  - offset from some other location

- Blob’s absolute address is 10000
- Datum’s relative address (to Blob) is 100
  - its absolute address is 10100
Base Registers

mov $10000, %base
mov $10, 100(%base)
Addresses

```c
long b;

int func(long c, long d) {
    long a;
    a = (b + c) * d;
    ...
}
```

```
mov 1000,%acc
add -8(%base),%acc
mul -12(%base),%acc
mov %acc,-16(%base)
```
Quiz 3

Suppose the value in base is 10,000. What is the address of c?

a) 9992  
b) 9996  
c) 10,004  
d) 10,008

mov 1000,%acc
add -8(%base),%acc
mul -12(%base),%acc
mov %acc,-16(%base)
Registers

- Instruction pointer
- Accumulator
- Base register
- More
- Condition codes

Execution engine

Interchangeable
Registers vs. Memory

- Execution engine
  - Instruction pointer
  - Accumulator
  - Base register
  - More
  - Condition codes

Memory (aka RAM)

- Instructions and data
- Data

A relatively long distance