Binary Arithmetic

These lecture notes were originally made to accompany Chapter 4 of the book *SPARC Architecture, Assembly Language Programming, and C*, by Richard P. Paul, 2nd edition, 2000. Now they are adapted to a JVM platform.

By Michael Weeks

Arithmetic

- Arithmetic involves
  - addition
  - subtraction
  - multiplication
  - division
- People use base 10
- Computers use base 2
- Base 2 is actually *easier* than base 10
Addition

- From right to left, we add each pair of digits
- We write the sum, and add the carry to the next column on the left

```
  1  9  8
+  2  6  4
  4  6  2
  0  1  1
```

```
  0  0  1
  0  1  0
  1  0  0
  1  1  1
```

Binary Sums and Carries

```
a  b  Sum  a  b  Carry
0  0  0    0  0  0
0  1  1    0  1  0
1  0  1    1  0  0
1  1  1    1  1  1
```

XOR  AND
Binary Addition Example

```
sipush 0x45
sipush 0xE7
iadd
```

• What value is on the stack after this?

\[
\begin{array}{c}
0x45 = \begin{array}{c}
0 0100 0101
\end{array} \\
0x\text{E7} = \begin{array}{c}
0 1110 0111
\end{array} \\
\hline
1 0010 1100 \\
\end{array}
\Rightarrow 0x12C
\]

so the stack has 0x12C

---

Half Adder / Full Adder

• A half adder
  – For adding 2 bits
  – Gives “carry out” and “sum”
  – 1 AND and 1 XOR gate

• A full adder
  – For adding 2 bits plus a “carry in”
  – Gives “carry out” and “sum”
  – 2 ANDs, 2 XORs, and 1 OR

• You’ll see more of this in the Computer Architecture class
Figure 4.2 – Full Adder

Modulus Arithmetic

- “Modulus arithmetic considers only numbers in the range $0 \leq n < M$, where $M$ is the modulus.” [Paul, page 117]
- Example: a car odometer
  if a car has 99999 miles on it,
  and you drive another mile,
  the odometer will then read 00000
Modulus Arithmetic

• Computers do this
• Local variables can store integer numbers
  – between 0 and $2^n-1$, where $n=32$
  – Treated as signed (wait about 3 slides)
• If a local variable has the value $2^n-1$, and you add 1 to this, it will then hold a value of 0

Subtraction

• Let $r$ be the base, and $n$ the number of digits
  $a - b = a + (r^n - 1 - b) + 1$
• since the result is modulus $r^n$, adding $r^n$ does not affect the result
  – Imagine if your car’s odometer read 54321 and you drove 100,000 miles, it would then read 54321 again.
• no borrowing is needed
• once $(r^n - 1 - b)$ is found, subtraction can be done with addition
Complement Arithmetic

• $r^n - 1 - b$ is called the
  – “nine’s complement” if $r=10$
  – “one’s complement” if $r=2$
• $r^n - 1 - b + 1$ is called the radix complement
  – “ten’s complement” if $r=10$
  – “two’s complement” if $r=2$
• Any number where the most significant digit $\geq (r/2)$ is considered negative
  10000000 means –128 when $r=2, n=8$
  84 means –16 when $r=10, n=2$ [see Paul, page 120]

Two’s Complement

• In binary, finding the one’s complement and the two’s complement are easy
• One’s complement:
  – Replace every 0 with a 1,
  – and replace every 1 with a 0
• Two’s complement:
  – Find the one’s complement,
  – and add 1
Two’s Complement Example

- What is –16 (decimal) in binary (r=2)?
- We’ll assume n=8

\[ 16 = 0001\ 0000 \text{ in binary} \]

\[ 1110\ 1111 \text{ one’s complement} \]

\[ +\ 0000\ 0001 \text{ add 1} \]

\[ 1111\ 0000 \text{ two’s complement} \]

Two’s Complement Numbers

<table>
<thead>
<tr>
<th>four-bit</th>
<th>eight-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111 7</td>
<td>0111 1111 127</td>
</tr>
<tr>
<td>0110 6</td>
<td>0111 1110 126</td>
</tr>
<tr>
<td>0101 5</td>
<td>....</td>
</tr>
<tr>
<td>0100 4</td>
<td>0000 0100 4</td>
</tr>
<tr>
<td>0011 3</td>
<td>0000 0011 3</td>
</tr>
<tr>
<td>0010 2</td>
<td>0000 0010 2</td>
</tr>
<tr>
<td>0001 1</td>
<td>0000 0001 1</td>
</tr>
<tr>
<td>0000 0</td>
<td>0000 0000 0</td>
</tr>
<tr>
<td>1111 -1</td>
<td>1111 1111 -1</td>
</tr>
<tr>
<td>1110 -2</td>
<td>1111 1110 -2</td>
</tr>
<tr>
<td>1101 -3</td>
<td>1111 1101 -3</td>
</tr>
<tr>
<td>1100 -4</td>
<td>1111 1100 -4</td>
</tr>
<tr>
<td>1011 -5</td>
<td>1111 1011 -5</td>
</tr>
<tr>
<td>1010 -6</td>
<td>....</td>
</tr>
<tr>
<td>1001 -7</td>
<td>1000 0001 -127</td>
</tr>
<tr>
<td>1000 -8</td>
<td>1000 0000 -128</td>
</tr>
</tbody>
</table>
Number Ranges

• A signed number has the range
  \(-2^{n-1}\) to \(2^{n-1} - 1\)
• An unsigned number has the range
  0 to \(2^n - 1\)
• What is the range of a local variable?
  -2147483648 to 2147483647, or
  0x80000000 to 0x7FFFFFFF

Integer, Long, Float, Double

– An integer has 32 bits
– A long is an integer with 64 bits
– A float is a floating-point value with 32 bits
– A double is a floating-point value with 64 bits
Integer versus Float

- An integer only stores whole numbers
- Could be positive, negative, or zero
- A float stores fractional numbers, e.g. 12.3
- In binary, integers and floats are stored very differently
  - 12 as integer is 0x0000 000c
  - 12 as float is 0x4140 0000

Float storage

- IEEE 754 standard
- sign (1 bit), exponent (8), significand (23)
- 01400000 is 0100 0001 0100 0000 0000
  0000 0000 0000
- Sign is 0
- Exponent is 100 0001 0, or 130 decimal
- Significand is 100...0

[For more information, see chapter 1 of Digital Signal Processing Using MATLAB and Wavelets, 2nd edition]
Float storage

- 0 sign means positive (1 for negative)
- Exponent is biased, subtract 127
  - $130 - 127 = 3$
- Significand needs to be prepended by "1."
  - Becomes 1.1
- Number is $+1.1 \times 2^3$
  - Shift 1.1 three times to get +1100.0
  - 1100, or 12 (decimal)

Addition and Subtraction

- “The two’s complement system is an interpretation of numbers in registers; the hardware always performs binary addition.” [Paul, page 122]
- To subtract, find the 2’s complement of the 2nd operand, and add
- There is no need for a hardware subtractor
Addition and Subtraction

- **Addition**: iadd, ladd, fadd, dadd
  - (integer, long, float, double)
- **Subtraction**: isub, lsub, fsub, dsub

Multiplication and Division

- **Mult.** can be done with additions
- Some architectures (e.g. SPARC) do not have multiplication or division instructions
- JVM has imul, lmul, fmul, dmul
  - (first letter for integer, long, float, double)
- Also idiv, ldiv, fdiv, ddiv
- Remainders: irem, lrem, frem, drem
Other Arithmetic Operations

- Remainders: irem, lrem, frem, drem
- Negation: ineg, lneg, fneg, dneg
  - Equivalent to 2's complement for integers
- Increment local variable: iinc
  - iinc localvariable number amount

[See chapter 2 of *The Java® Virtual Machine Specification, Java SE 7 Edition*]

Shift Operations

- A register’s contents can be shifted
  - left shift is like multiplying by 2
  - right shift is like dividing by 2
- Logical shift
  - copies 0 into most significant bit(s)
- Arithmetic shift
  - Copies the sign bit into most significant bit
    (otherwise, negatives could become positives)
Shift Instructions

• Shift right logical (iushr,lushr) unsigned

• Shift right arithmetic (ishr,lshr)

• Shift left logical (ishl,lshl)

• Shift left arithmetic is not provided,
  – it would be the same as shift left logical

Shift Instructions

• Shift left logical
  
  iload_0 ; stack = localVar0
  iconst_1 ; number of shifts
  ishl

• Push the value to shift
• Push the number of shifts (1 in this case)
• ishl pops the two parameters, pushes result
Typical Flags

• Common flags used with branching:
  
  N  (negative)
  the most significant bit of the result
  not used with unsigned numbers
  V  (overflow)
  when result is too big for the register
  Z  (zero)
  set when all bits of the result are 0
  C  (carry)
  set when an addition has carry out,
  or when subtraction does not have carry out

Branching

• JVM has if<condition>, e.g.
  
  • ifeq label

• It pops the value from the stack

• Branch if value meets condition
  
  • In this case, branch (jump) to "label" if stack value is 0
## Branches

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>iflt</td>
<td>branch on less than</td>
<td>(N xor V) = 1</td>
</tr>
<tr>
<td>ifle</td>
<td>branch on less or equal</td>
<td>Z or (N xor V) = 1</td>
</tr>
<tr>
<td>ifeq</td>
<td>branch on equal</td>
<td>Z = 1</td>
</tr>
<tr>
<td>IFNE</td>
<td>branch on not equal</td>
<td>Z = 0</td>
</tr>
<tr>
<td>ifge</td>
<td>branch on greater or equal</td>
<td>(N xor V) = 0</td>
</tr>
<tr>
<td>ifgt</td>
<td>branch on greater than</td>
<td>Z or (N xor V) = 0</td>
</tr>
<tr>
<td>ifnull*</td>
<td>branch on NULL value</td>
<td></td>
</tr>
<tr>
<td>ifnonnull</td>
<td>branch on not NULL</td>
<td></td>
</tr>
</tbody>
</table>

* NULL is not the same as zero

## Branching

- JVM has if icmp<condition>, e.g.
  - if icmpeq label
- It pops 2 values from the stack
- Branch if values meet condition
  - In this case, branch (jump) to "label" if stack values are equal
    (if first-pushed minus second-pushed is 0)
Branches

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>if_icmplt</td>
<td>branch on less than</td>
<td>(N xor V) = 1</td>
</tr>
<tr>
<td>if_icmple</td>
<td>branch on less or equal</td>
<td>Z or (N xor V) = 1</td>
</tr>
<tr>
<td>if_icmpeq</td>
<td>branch on equal</td>
<td>Z = 1</td>
</tr>
<tr>
<td>if_icmpne</td>
<td>branch on not equal</td>
<td>Z = 0</td>
</tr>
<tr>
<td>if_icmpge</td>
<td>branch on greater or equal</td>
<td>(N xor V) = 0</td>
</tr>
<tr>
<td>if_icmpgt</td>
<td>branch on greater than</td>
<td>Z or (N xor V) = 0</td>
</tr>
<tr>
<td>if_acmpeq*</td>
<td>branch on equal</td>
<td></td>
</tr>
<tr>
<td>if_acmpne</td>
<td>branch on not equal</td>
<td></td>
</tr>
</tbody>
</table>

* a is for address, meaning a reference

Comparisons

- Comparison statements
  - lcmp, fcmpl, dcmpl
- Pops 2 values from stack
- Pushes -1, 0, or 1 onto stack
- Result indicates relationship
  - less than, equal, greater than
Integer Overflow

• "The Java Virtual Machine does not indicate overflow during operations on integer data types" [http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.8.1]

Labels

• Put a label on a line by itself
• e.g.

    icnst_0

    loop:

    iload_0
Summary

• sum and carry
• modulus arithmetic
  – one’s complement
  – two’s complement
• signed and unsigned arithmetic
• shifting
• conditions and branching