Overview

- Computer representation of “things”
- Unsigned Numbers
- Signed Numbers: search for a good representation
- Shortcuts
- In Conclusion

Review: The Programmer’s Model of a Microcomputer

Instruction Set:
- ldr  r0 , [r2, #0]
- add  r2, r3, r4

Registers:
- r0 - r3, pc

Addressing Modes:
- ldr  r12, [r1,#0]
- mov  r1 , r3

Memory:
- 80000004  ldr r0 , [r2, #0]
- 80000008  add r2, r3, r4
- 8000000B  23456
- 80000010  AEF0

Memory mapped I/O:
- 80000100  input
- 80000108  output

Programmer’s Model

Review: Compilation

- How to turn notation programmers prefer into notation computer understands?
- Program to translate C statements into Assembly Language instructions; called a compiler

Example: compile by hand this C code:

```c
a = b + c;
d = a - e;
```

- Easy:
  - add r1, r2, r3
  - sub r4, r5, r6

- Big Idea: compiler translates notation from 1 level of abstraction to lower level
What do computers do?

- Computers **manipulate representations of things!**

- **What can you represent with N bits?**
  - \(2^N\) things!

- **Which things?**
  - Numbers! Characters! Pixels! Dollars! Position! Instructions!
  - Depends on what operations you do on them

---

**Decimal Numbers: Base 10**

- **Digits:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- **Example:**
  
  \[
  3271 = (3\times10^3) + (2\times10^2) + (7\times10^1) + (1\times10^0)
  \]

---

**Numbers: positional notation**

- **Number Base B => B symbols per digit:**
  - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Base 2 (Binary): 0, 1

- **Number representation:**
  - \(d_3d_2d_1d_0\) is a 32 digit number
  - value = \(d_3\times B^3 + d_2\times B^2 + d_1\times B^1 + d_0\times B^0\)

- **Binary:** 0, 1
  - \(1011010 = 1\times2^6 + 0\times2^5 + 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2 + 0\times1 = 64 + 16 + 8 + 2 = 90\)
  - Notice that 7 digit binary number turns into a 2 digit decimal number
  - A base that converts to binary easily?

---

**Hexadecimal Numbers: Base 16 (#1/2)**

- **Digits:** 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

- **Normal digits have expected values**

- **In addition:**
  - A \(\rightarrow\) 10
  - B \(\rightarrow\) 11
  - C \(\rightarrow\) 12
  - D \(\rightarrow\) 13
  - E \(\rightarrow\) 14
  - F \(\rightarrow\) 15
Hexadecimal Numbers: Base 16 (#2/2)

° Example (convert hex to decimal):
   B28F0DD = (Bx16^5) + (2x16^4) + (8x16^3) + (Fx16^2) +
   (0x16^1) + (Dx16^0)
   = (11x16^5) + (2x16^4) + (8x16^3) + (15x16^2) +
   (0x16^1) + (13x16^1) + (13x16^0)
   = 187232477 decimal

° Notice that a 7 digit hex number turns out to
   be a 9 digit decimal number

Decimal vs. Hexadecimal vs. Binary

° Examples:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>00</td>
</tr>
<tr>
<td>0001</td>
<td>01</td>
</tr>
<tr>
<td>0100</td>
<td>02</td>
</tr>
<tr>
<td>0101</td>
<td>03</td>
</tr>
<tr>
<td>0110</td>
<td>04</td>
</tr>
<tr>
<td>0111</td>
<td>05</td>
</tr>
<tr>
<td>1010</td>
<td>06</td>
</tr>
<tr>
<td>1011</td>
<td>07</td>
</tr>
</tbody>
</table>

° 1010 1100 0101 (binary) = ? (hex)
° 1011 1000 0111 (binary) = ? (hex)
° 3F9 (hex) = ? (binary)

Hex to Binary Conversion

° HEX is a more compact representation of Binary!
° Each hex digit represents 16 decimal values.
° Four binary digits represent 16 decimal values.
° Therefore, each hex digit can replace four binary
digits.

° Example:
   0011 1011 1001 1010 1100 1010 0000 0000two
   3  b  9  a  c  a  0  0hex
   C uses notation 0x3b9aca00

Which Base Should We Use?

° Decimal: Great for humans; most arithmetic
   is done with these.
° Binary: This is what computers use, so get
   used to them. Become familiar with how to
   do basic arithmetic with them (+,-,*)./
° Hex: Terrible for arithmetic; but if we are
   looking at long strings of binary numbers,
   it’s much easier to convert them to hex in
   order to look at four bits at a time.
How Do We Tell the Difference?

° In general, append a subscript at the end of a number stating the base:
  • $10_{10}$ is in decimal
  • $10_2$ is binary ($= 2_{10}$)
  • $10_{16}$ is hex ($= 16_{10}$)

° When dealing with ARM computer:
  • Hex numbers are preceded with “&” or “0x”
    - &10 == 0x10 == $10_{16} == 16_{10}$
    - Note: Lab software environment only supports “0x”
  • Binary numbers are preceded with “0b”
  • Octal numbers are preceded with “0”
  • Everything else by default is Decimal

Inside the Computer

° To a computer, numbers are always in binary; all that matters is how they are printed out: binary, decimal, hex, etc.

° As a result, it doesn’t matter what base a number in C is in...
  • $32_{10} == 0x20 == 100000_2$

° ... only the value of the number matters.

What to do with representations of numbers?

° Just what we do with numbers!
  • Add them
  • Subtract them
  • Multiply them
  • Divide them
  • Compare them

° Example: \[ \begin{align*}
10 + 7 &= 17 \\
100000_2 + 01111_2 &= 100011_2
\end{align*} \]

 Addition Table

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
</tbody>
</table>
Addition Table (binary)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Addition Table (Hex)

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
<td>1D</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
<td>1D</td>
<td>1E</td>
</tr>
</tbody>
</table>

Bicycle Computer (Embedded)

- P. Brain
  - wireless heart monitor strap
  - record 5 measures: speed, time, current distance, elevation and heart rate
  - Every 10 to 60 sec.
  - 8KB data => 33 hours
  - Stores information so can be uploaded through a serial port into PC to be analyzed

Limits of Computer Numbers

- Bits can represent anything!

- Characters?
  - 26 letter => 5 bits
  - upper/lower case + punctuation => 7 bits (in 8) (ASCII)
  - rest of the world’s languages => 16 bits (unicode)

- Logical values?
  - 0 => False, 1 => True

- Colors?

- Locations / addresses? commands?
  - but N bits => only $2^N$ things
What if too big?

- Binary bit patterns above are simply representatives of numbers.
- Numbers really have an infinite number of digits.
  - With almost all being zero except for a few of the rightmost digits, e.g.: 00000000 ... 000098 == 98
  - Just don’t normally show leading zeros.
- Computers have fixed number of digits.
  - In general, adding two n-bit numbers can produce an (n+1)-bit result.
  - Since computers use fixed, 32-bit integers, this is a problem.
  - If result of add (or any other arithmetic operation) cannot be represented by these rightmost hardware bits, overflow is said to have occurred.

Overflow Example

- Example (using 4-bit numbers):
  - +15 1111
  - +3 0011
  - +18 10010
  - But we don’t have room for 5-bit solution, so the solution would be 0010, which is +2, which is wrong.

How avoid overflow, allow it sometimes?

- Some languages detect overflow (Ada), some don’t (C and JAVA).
- ARM has N, Z, C and V flags to keep track of overflow.
  - Refer Book!
  - Will cover details later.

Comparison

- How do you tell if X > Y?
- See if X - Y > 0
  - We need representation for both +ve and -ve numbers.
How to Represent Negative Numbers?

- So far, unsigned numbers
- Obvious solution: define leftmost bit to be sign!
  - 0 => +, 1 => -
  - Rest of bits can be numerical value of number
- Representation called **sign and magnitude**
- ARM uses 32-bit integers. \(+1_{ten}\) would be:
  \[\begin{array}{c}
  0000 0000 0000 0000 0000 0000 0000 0001 \\
  \end{array}\]
- And \(-1_{ten}\) in sign and magnitude would be:
  \[\begin{array}{c}
  1000 0000 0000 0000 0000 0000 0000 0001 \\
  \end{array}\]

Shortcomings of sign and magnitude?

- Arithmetic circuit more complicated
  - Special steps depending whether signs are the same or not
- Also, Two zeros
  - \(0x00000000 = +0_{ten}\)
  - \(0x80000000 = -0_{ten}\)
  - What would it mean for programming?
- Sign and magnitude abandoned because another solution was better

Another try: complement the bits

- Example: \(7_{10} = 001112\), \(-7_{10} = 110002\)
- Called **one’s Complement**
- Note: positive numbers have leading 0s, negative numbers have leadings 1s.

\[\begin{array}{c}
  00000 \ 00001 \ ... \ 01111 \\
  \hline
  10000 \ ... \ 11110 \ 11111 \\
  \end{array}\]
- What is \(-00000\) ?
- How many positive numbers in \(N\) bits?
- How many negative ones?

Shortcomings of ones complement?

- Arithmetic not too hard
- Still two zeros
  - \(0x00000000 = +0_{ten}\)
  - \(0xFFFFFFF = -0_{ten}\)
  - What would it mean for programming?
- One’s complement eventually abandoned because another solution was better
Search for Negative Number Representation

° Obvious solution didn’t work, find another

° What is result for unsigned numbers if tried to subtract large number from a small one?
  • Would try to borrow from string of leading 0s, so result would have a string of leading 1s
  
    \[ 3 - 7 = -4 \]

  • With no obvious better alternative, pick representation that made the hardware simple:
    leading 0s => positive, leading 1s => negative
    
    \[ 0000...xxx \text{ is } \geq 0, \quad 11111...xxx \text{ is } < 0 \]

° This representation called **two's complement**

Two's Complement Number line

- \( 2^{N-1} \) non-negatives
- \( 2^{N-1} \) negatives
- one zero
- how many positives?
- comparison?
- overflow?

Two's Complement Formula, Example

° Recognizing role of sign bit, can represent positive and negative numbers in terms of the bit value times a power of 2:

\[
\begin{align*}
\text{Example} & \quad 1111 \ldots 1111 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \quad 1111 \quad 1100_{\text{two}} \\
& = 1 \times -2^{31} + 1 \times 2^{30} + 1 \times 2^{29} + \ldots + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
& = -2^{31} + 2^{30} + 2^{29} + \ldots + 2^2 + 0 + 0 \\
& = -2147483648_{\text{ten}} + 2147483644_{\text{ten}} \\
& = -4_{\text{ten}}
\end{align*}
\]

° One zero, 31st bit => \( \geq 0 \) or <0, called **sign bit**

• but one negative with no positive \(-2147483648_{\text{ten}}\)
Two’s complement shortcut: Negation

- Invert every 0 to 1 and every 1 to 0, then add 1 to the result
  - Sum of number and its inverted representation must be \(111...111_\text{two}\)
  - \(111...111_\text{two} = -1_{\text{ten}}\)
  - Let \(x'\) mean the inverted representation of \(x\)
  - Then \(x + x' = -1\) => \(x + x' + 1 = 0\) => \(x' + 1 = -x\)

Example: -4 to +4 to -4

\[
\begin{array}{c|c|c|c|c}
\text{x} & \text{x'} & +1 & ()' & +1 \\
\hline
1111 & 111100 & 111100 & 111111 & 111100 \\
\end{array}
\]

And in Conclusion...

- We represent “things” in computers as particular bit patterns: \(N\) bits \(\Rightarrow 2^N\)
  - numbers, characters, ... (data)
- Decimal for human calculations, binary to understand computers, hex to understand binary
- 2’s complement universal in computing: cannot avoid, so learn
- Computer operations on the representation correspond to real operations on the real thing
- Overflow: numbers infinite but computers finite, so errors occur