COMP 3221
Microprocessors and Embedded Systems
Lecture 7: Number Systems - III

http://www.cse.unsw.edu.au/~cs3221

August, 2003
Saeid Nooshabadi
Saeid@unsw.edu.au

Overview

° Condition Code Flag interpretation
° Characters and Strings
° In Conclusion

Review: int and unsigned int in C

° With N bits we can represent 2^N different Numbers:
  • 2^N numbers 0 to 2^N - 1: Only zero and Positive numbers
  • 2^N numbers -2^N/2 to 0 to 2^N/2- 1: Both Negative and positive numbers in 2's Complement

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>C</th>
<th>V</th>
<th>FP</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>T</td>
<td>0</td>
</tr>
</tbody>
</table>

Review: Condition Flags

Flags | Arithmetic Instruction
--- | ---
Negative (N='1') | Bit 31 of the result has been set
                    | Indicates a negative number in signed operations
Zero (Z='1') | Result of operation was zero
Carry (C='1') | Result was greater than 32 bits
Overflow (V='1') | Result was greater than 31 bits
                    | Indicates a possible corruption of the sign bit in signed numbers

Is 1000 > 0110?
1000 > 0110 if only +ve representation used
1000 < 0110 if both +ve and -ve representation in 2's complement used
Indicate the changes in $N$, $Z$, $C$, $V$ flags for the following arithmetic operations: (Assume 4 bit-numbers)

### Experimentation with Condition Flags (#1/4)

$$\begin{align*}
1110 & + 1111 = 10010 \\
\text{Signed interpretation: } -1 + 2 = 1. \text{ The number is within the range of } -8 \text{ to } +7. \text{ No Overflow (V), Ignore Carry out.}
\end{align*}$$

Unsigned interpretation: $15 + 2 = 17$. The number is out of the range of $0$ to +15. Carry Set and Overflow Not set. Indication for overflow in unsigned.

$$\begin{align*}
\text{NOTE: } V = \text{MSB Carry In (XOR) MSB Carry out}
\end{align*}$$

### Experimentation with Condition Flags (#2/4)

$$\begin{align*}
0110 & + 0111 = 01001 \\
\text{Signed interpretation: } 7 + 2 = 9. \text{ The number is out of the range of } -8 \text{ to } +7. \text{ Overflow (V), Ignore Carry out.}
\end{align*}$$

Unsigned interpretation: $7 + 2 = 9$. The number is within the range of $0$ to +15. Carry Not set and Overflow Set. Indication for No overflow in unsigned.

$$\begin{align*}
\text{NOTE: } V = \text{MSB Carry In (XOR) MSB Carry out}
\end{align*}$$

### Experimentation with Condition Flags (#3/4)

$$\begin{align*}
1000 & - 1110 = 01101 \\
\text{Signed interpretation: } -2 - 7 = -9. \text{ The number is out of the range of } -8 \text{ to } +7. \text{ Overflow (V), Ignore Carry out.}
\end{align*}$$

Unsigned interpretation: $14 - 7 = 7$. The number is in of the range of $0$ to +15. Carry Set and Overflow Set. Indication for No overflow in unsigned.

### Experimentation with Condition Flags (#4/4)

$$\begin{align*}
0010 & + 0011 = 00101 \\
\text{Signed interpretation: } 3 + 2 = 5. \text{ The number is within of the range } -8 \text{ to } +7. \text{ No Overflow (V), Ignore Carry out.}
\end{align*}$$

Unsigned interpretation: $3 + 2 = 5$. The number is within the range of $0$ to +15. Carry Not set and Overflow Not set. Indication for No overflow in unsigned.
**Signed /Unsigned Overflow Summary**

Signed Arithmetic overflow Condition:
- Overflow flag $V = 0$  NO OVERFLOW
- Overflow flag $V = 1$  OVERFLOW

**NOTE:** $V = MSB$ Carry In (XOR) MSB Carry out

Unsigned Arithmetic overflow Condition:
- Overflow: ($V = 0$) AND (Carry flag $C = 0$) NO OVERFLOW
- ($V = 0$) AND (Carry flag $C = 1$) OVERFLOW
- ($V = 1$) AND (Carry flag $C = 0$) NO OVERFLOW
- ($V = 1$) AND (Carry flag $C = 1$) NO OVERFLOW

---

**Sign Extension**

- Consider:
  - $1111 = -1$ in 4-bit representation
  - $1111\ 1111 = -1$ in 8-bit representation
  - $1111\ 1111\ 1111\ 1111 = -1$ in 16-bit representation

  - 2's comp. negative number has infinite 1s
    - $0111 = 7$ in 4-bit representation
    - $0000\ 0111 = 7$ in 8-bit representation
    - $0000\ 0000\ 0000\ 0111 = 7$ in 16-bit representation

  - 2's comp. positive number has infinite 0s
  - Bit representation hides leading bits

---

**Two’s comp. shortcut: Sign extension**

- Convert 2’s complement number using $n$ bits to more than $n$ bits
  - Simply replicate the most significant bit (sign bit) of smaller to fill new bits
    - 2’s comp. positive number has infinite 0s
    - 2’s comp. negative number has infinite 1s
    - Bit representation hides leading bits; sign extension restores some of them
    - 16-bit $-4_{\text{ten}}$ to 32-bit:
      - $1111\ 1111\ 1111\ 1100_{\text{two}}$
      - $1111\ 1111\ 1111\ 1111\ 1100_{\text{two}}$

---

**Beyond Integers (Characters)**

- 8-bit bytes represent characters, nearly every computer uses American Standard Code for Information Interchange (ASCII)

<table>
<thead>
<tr>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>48</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>`</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>2</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
</tr>
</tbody>
</table>

- Uppercase $+ \ 32 = $ Lowercase (e.g., B$+32=b$)
- tab$=9$, carriage return$=13$, backspace$=8$, Null$=0$

(Table in CD-ROM)
Strings

° Characters normally combined into strings, which have variable length
  • e.g., “Cal”, “M.A.D”, “COMP3221”

° How represent a variable length string?
  1) 1st position of string reserved for length of string (Pascal)
  2) an accompanying variable has the length of string (as in a structure)
  3) last position of string is indicated by a character used to mark end of string (C)

° C uses 0 (Null in ASCII) to mark end of string

Example String

° How many bytes to represent string “Popa”?
° What are values of the bytes for “Popa”?

<table>
<thead>
<tr>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
<th>No.</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>@</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>`</td>
<td>112</td>
<td>`</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>69</td>
<td>O</td>
<td>95</td>
<td>_</td>
<td>111</td>
<td>o</td>
<td>127</td>
<td>DEL</td>
</tr>
</tbody>
</table>

° 80, 111, 112, 97, 0 DEC
° 50, 6F, 70, 61, 0 HEX

Strings in C: Example

° String simply an array of char
    void strcpy (char x[], char y[]){
      int i = 0; /* declare,initialize i*/

      while ((x[i] = y[i]) != ‘\0’) /* 0 */
          i = i + 1; /* copy and test byte */
    }

What about non-Roman Alphabet?

° Unicode, universal encoding of the characters of most human languages
  • Java uses Unicode
  • needs 16 bits to represent a character
  • 16-bits called half word in ARM
### ASCII v. Binary

- Why not ASCII computers vs. binary computers?
  - Harder to build hardware for add, subtract, multiply, divide
  - Memory space to store numbers
- How many bytes to represent 1 billion?
  - ASCII: “1000000000” => 11 bytes
  - Binary: 0011 1011 1001 1010 1000 0000 0000 0000 => 4 bytes
- up to 11/4 or almost 3X expansion of data size

### What else is useful to represent?

- Numbers, Characters, logicals, ...
- Addresses
- Commands (operations)
  - example:
    - 0 => clap your hands
    - 1 => snap your fingers
    - 2 => slap your hands down
  - execute: 1 0 2 0 1 0 2 0 1 0 2 0
  - another example
    - 0 => add
    - 1 => subtract
    - 2 => compare
    - 3 => multiply

### How can we represent a machine instruction?

- Some bits for the operation
- Some bits for the address of each operand
- Some bits for the address of the result

<table>
<thead>
<tr>
<th>operation</th>
<th>result addr</th>
<th>op1 addr</th>
<th>op2 addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = x + y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### The Stored Program Computer

- Memory holds instructions and data as bits
- Instructions are fetched from memory and executed
  - operands fetched, manipulated, and stored
- Example 4-digit Instruction

<table>
<thead>
<tr>
<th>operation</th>
<th>result addr</th>
<th>op1 addr</th>
<th>op2 addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 7 4 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 8 7 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 9 8 9 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Example Data
  - 4 digit unsigned value
- What’s in memory after executing 0,1,2?
So what’s it all mean?

° We can write a program that will translate strings of ‘characters’ into ‘computer instructions’
  • called a compiler or an assembler

° We can load these particular bits into the computer and execute them.
  • may manipulate numbers, characters, pixels... (application)
  • may translate strings to instructions (compiler)
  • may load and run more programs (operating system)

To remember

° We represent “things” in computers as particular bit patterns
  • numbers, characters, ... (data)
    - base, digits, positional notation
    - unsigned, 2s complement, 1s complement
  • addresses (where to find it)
  • instructions (what to do)

° Computer operations on the representation correspond to real operations on the real thing
  • representation of 2 plus representation of 3 = representation of 5

° two big ideas already!
  • Pliable Data: a program determines what it is
  • Stored program concept: instructions are just data

And in Conclusion...

° 2’s complement universal in computing: cannot avoid, so learn

° Overflow: numbers infinite but computers finite, so errors occur

° Computers provide help to detect overflow

° Condition code flags N, Z, C and V provide help to deal with arithmetic computation and interpretation in signed and unsigned representation.