Overview

- Overflow in 2’s complement addition
- Comparison in signed and unsigned numbers
- Condition flags
- Characters and strings

Two’s Complement’s Arithmetic Examples

° Example 1: \(20 - 4 = 16\)

° Assume 8 bit architecture.

\[
20 - 4 = 20 + (-4) \\
= 0001 0100_{\text{two}} - 0000 0100_{\text{two}} \\
= 0001 0100_{\text{two}} \\
+ 1111 1100_{\text{two}} \\
= 10001 0000_{\text{two}}
\]

Carry Most significant bit (msb) No overflow.

° Example 2: \(-127 - 2 = -129\)?

° \(-127 - 2\)

\[
= -0111 1111_{\text{two}} - 0000 0010_{\text{two}} \\
= 1000 0001_{\text{two}} \\
+ 1111 1110_{\text{two}} \\
= 10111 1111_{\text{two}}
\]

Carry msb Overflow
Two’s Complement’s Arithmetic Examples

° Example 3: 127 + 2 = 129?
° 127 + 2

\[
\begin{align*}
&= 0111\ 1111_{\text{two}} + 0000\ 0010_{\text{two}} \\
&= 0111\ 1111_{\text{two}} \\
&\quad + 0000\ 0010_{\text{two}} \\
&= 1000\ 0001_{\text{two}} \\
& \text{msb Overflow}
\end{align*}
\]

When Overflow Occurs?

The ‘two’s complement overflow’ occurs when:
• both the msb’s being added are 0 and the msb of the result is 1
• both the msb’s being added are 1 and the msb of the result is 0

Signed vs. Unsigned Numbers

° C declaration int
  • Declares a signed number
  • Uses two’s complement

° C declaration unsigned int
  • Declares an unsigned number
  • Treats 32-bit number as unsigned integer, so most significant bit is part of the number, not a sign bit

° NOTE:
  • Hardware does all arithmetic in 2’s complement.
  • It is up to programmer to interpret numbers as signed or unsigned.

Signed and Unsigned Numbers in AVR

° AVR microcontrollers support only 8 bit signed and unsigned integers.
° Multi-byte signed and unsigned integers can be implemented by software.
° Question: How to compute

\[
\begin{align*}
&10001110\ 01110000\ 11100011\ 00101010_{\text{two}} \\
&\quad + 01110000\ 11001000\ 10001100\ 01110001_{\text{two}}
\end{align*}
\] on AVR?
Signed and Unsigned Numbers in AVR (Cont.)

Solution: Four-byte integer addition can be done by using four one-byte integer additions taking carries into account (lowest bytes are added first).

\[
\begin{align*}
10001110 & \quad 01100000 \quad 11000111 \quad 00101010 \\
+ 01110000 & \quad + 11001000 \quad + 10001100 \quad + 01110001 \\
= 11111111 & \quad 00111001 \quad 01101111 \quad 10011011 \\
\text{Carry bits} & \\
\text{The result is} & \quad 11111111 \quad 00111001 \quad 01101111 \quad 10011011_{\text{two}}
\end{align*}
\]

Signed v. Unsigned Comparison

- \(X = 1111\ 1100_{\text{two}}\)
- \(Y = 0000\ 0010_{\text{two}}\)

- Is \(X > Y\)?
  - unsigned: YES
  - signed: NO

Signed v. Unsigned Comparison (Hardware Help)

- \(X = 1111\ 1100_{\text{two}}\)
- \(Y = 0000\ 0010_{\text{two}}\)

- Is \(X > Y\)? Do the Subtraction \(X - Y\) and check result

\[
\begin{align*}
X - Y &= 1111\ 1100_{\text{two}} - 0000\ 0010_{\text{two}} \\
&= 1111\ 1100_{\text{two}} \\
& \quad + 1111\ 1110_{\text{two}} \\
&= 1111\ 1010_{\text{two}} \\
\text{Carry} & \\
\text{Msb (sign bit) indicates } X < Y \text{ (signed numbers)} \\
\text{Msb and carry indicates } X \text{ is greater than } Y \text{ (unsigned numbers)}.
\end{align*}
\]

Numbers are stored at addresses

- Memory is a place to store bits
- A word is a fixed number of bits (e.g., 16 in AVR assembler) at an address
- Addresses have fixed number of bits
- Addresses are naturally represented as unsigned numbers
- How multi-byte numbers are stored in memory is determined by the bitness.
- On AVR, programmers choose the bitness.
Status Flags in Program Status Register

The Processor Status Register in AVR

- C: 1 indicates a carry in arithmetic operation.
- Z: 1 indicates a zero result after a arithmetic or logical operation.
- N: 1 indicates a negative result.
- V: 1 indicates two’s complement overflow.
- S: Sign flag—exclusive OR between N and V.
- H: Half carry flag.

Experimentation with Condition Flags (#1/3)
Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

\[
\begin{align*}
0010 \ 0011 + 1010 \ 1111 &= 1101 \ 0010 \\
\text{• } N &= 1 \\
\text{• } V &= 0 \\
\text{• } Z &= 0 \\
\text{• } C &= 1 \\
\text{• } S &= 1 \\
\text{• } H &= 1
\end{align*}
\]

Experimentation with Condition Flags (#2/3)
Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

\[
\begin{align*}
1010 \ 0011 + 1010 \ 1111 &= 10101 \ 0010 \\
\text{• } N &= 0 \\
\text{• } V &= 1 \\
\text{• } Z &= 0 \\
\text{• } C &= 0 \\
\text{• } S &= 1 \\
\text{• } H &= 1
\end{align*}
\]

Experimentation with Condition Flags (#3/3)
Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

\[
\begin{align*}
0110 \ 0011 + 0011 \ 1011 &= 1001 \ 1110 \\
\text{• } N &= 1 \\
\text{• } V &= 1 \\
\text{• } Z &= 0 \\
\text{• } C &= 0 \\
\text{• } S &= 0 \\
\text{• } H &= 1
\end{align*}
\]
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>0</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>`</td>
<td>112</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>2</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</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>63</td>
<td>?</td>
<td>79</td>
<td>O</td>
<td>95</td>
<td>_</td>
<td>111</td>
<td>o</td>
</tr>
</tbody>
</table>

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

Strings

- Characters normally combined into strings, which have variable length
  - e.g., “Cal”, “M.A.D”, “COMP3221”
- How to 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 the end of a 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>48</td>
<td>0</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>`</td>
<td>112</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>2</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</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>63</td>
<td>?</td>
<td>79</td>
<td>O</td>
<td>95</td>
<td>_</td>
<td>111</td>
<td>o</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

```c
void strcpy (char x[], char y[])
{
    int i=0; /* declare and initialize i*/
    while ((x[i]=y[i])!='\0') /* 0 */
    i=i+1; /* copy and test byte */
}
```
String in AVR Assembly Language

- .db “Hello
” ; This is equivalent to .db ‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\n’
- What does the following instruction do?
  ldi r4, ‘1’

How to Represent A Machine Instruction?

- Some bits for the operation (addition, subtraction etc.).
- Some bits for each operand (the maximum number of operands in an instruction is determined by the instruction set).
- Example:

<table>
<thead>
<tr>
<th>operation</th>
<th>operand 1</th>
<th>operand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>4 bits</td>
<td>4 bits</td>
</tr>
</tbody>
</table>

- Will cover the details in next lecture.