Two’s Complement’s Arithmetic Examples

° Example 1: $20 - 4 = 16$

° Assume 8 bit architecture.

$20 - 4 = 20 + (-4)$

\[
\begin{align*}
&= 0001 0100_{\text{two}} - 0000 0100_{\text{two}} \\
&= 0001 0100_{\text{two}} \\
&+ 1111 1100_{\text{two}} \\
&= 10001 0000_{\text{two}}
\end{align*}
\]

Carry Most significant bit (msb)  No overflow.

Two’s Complement’s Arithmetic Examples

° Example 2: $-127 - 2 = -129$?

° $-127 - 2$

\[
\begin{align*}
&= -0111 1111_{\text{two}} - 0000 0010_{\text{two}} \\
&= 1000 0001_{\text{two}} \\
&+ 1111 1110_{\text{two}} \\
&= 10111 1111_{\text{two}}
\end{align*}
\]

Carry msb  Overflow

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 3: 127 + 2 = 129?
° 127 + 2
  = 0111 1111\text{two} + 0000 0010\text{two}
  = 0111 1111\text{two}
  + 0000 0010\text{two}
  = 1000 0001\text{two}
  \quad \text{msb \quad Overflow}

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 a 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

\[
10001110 \ 01110000 \ 11100011 \ 00101010_\text{two} \\
+ \ 01110000 \ 11001000 \ 10011000 \ 01110001_\text{two}
\]
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).

\[
\begin{align*}
10001110 & \quad 01110000 \quad 11100011 \quad 00101010 \\
+ 01110000 & \quad +11001000 \quad +10001100 \quad +01110001 \\
= 11111110 & \quad 10011100 \quad 10110111 \quad 01001101
\end{align*}
\]

The result is 11111111 00110011 01101111 10011011\text{\textsubscript{two}}

Signed v. Unsigned Comparison (Hardware Help)

° \( X = 1111 \ 1100\text{\textsubscript{two}} \)
° \( Y = 0000 \ 0010\text{\textsubscript{two}} \)
° Is \( X > Y \)? Do the Subtraction \( X - Y \) and check result

\[
\begin{align*}
X - Y &= 1111 \ 1100\text{\textsubscript{two}} - \ 0000 \ 0010\text{\textsubscript{two}} \\
&= 1111 \ 1100\text{\textsubscript{two}} \\
&\quad + 1111 \ 1110\text{\textsubscript{two}} \\
&= 1111 \ 1010\text{\textsubscript{two}}
\end{align*}
\]

Hardware needs to keep
° a special bit (S flag in AVR) which indicates the result of signed comparison, and
° a special bit (C flag in AVR) which indicates the result of unsigned comparison.

Signed v. Unsigned Comparison

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

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

Numbers are stored at addresses

- Memory is a place to store bits
- A word is a fixed number of bits (eg, 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 endianness.
- On AVR, programmers choose the endianness.
Status Flags in Program Status Register

<table>
<thead>
<tr>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

The Processor Status Register in AVR

- C: Its meaning depends on the operation.
  
  For addition X+Y, it is the carry from the most significant bit. In other words, \( C = R_7 \cdot R_r_7 + R_7 \cdot \text{NOT}(R_7) + \text{NOT}(R_7) \cdot R_d_7 \), where \( R_d_7 \) is bit 7 of x, \( R_r_7 \) is bit 7 of y, \( R_7 \) is bit 7 of x+y, \( \cdot \) is the logical AND and + is the logical OR.

  For subtraction x-y, where x and y are unsigned integer, it indicates if y<x. If y<x, the C=1; otherwise, C=0.

  In other words, \( C = \text{NOT}(R_d_7) \cdot R_r_7 + R_7 \cdot R_r_7 \cdot \text{NOT}(R_d_7) \).

- Z: 1 indicates a zero result after a arithmetic or logical operation.
- N: the most significant bit of the result.
- V: 1 indicates two’s complement overflow.
- S: Sign flag—exclusive OR between N and V.
  
  1: negative result.
  0: non-negative result.
- 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)

\[
\begin{align*}
0010 & \quad 0011 \\
+ \quad & \quad 1010 \quad 1111 \\
= \quad & \quad 1101 \quad 0010
\end{align*}
\]

- N=1
- V=0
- Z=0
- C=0
- S=1
- H=1

Experimentation with Condition Flags (#2/3)

Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

\[
\begin{align*}
1010 & \quad 0011 \\
+ \quad & \quad 1010 \quad 1111 \\
= & \quad 10101 \quad 0010
\end{align*}
\]

- N=0
- V=1
- Z=0
- C=1
- S=1
- H=1
**Experimentation with Condition Flags (#3/3)**

Indicate the changes in N, Z, C, V flags for the following arithmetic operations: (Assume 4 bit-numbers)

\[
\begin{align*}
01100011 & \quad 01100011 \\
-01111011 & = +10000101 \\
& = 11101000
\end{align*}
\]

- N = 1
- V = 0
- Z = 0
- C = 1
- S = 0
- H = 0

---

**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>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>@</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>char</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>49</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>51</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
</tr>
<tr>
<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>_</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>112</td>
<td>127</td>
<td>DEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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”?
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\n” ; This is equivalent to
  `.db ‘H’, ‘e’, ‘l’, ‘o’, ‘\n’

• What does the following instruction do?
  `ldi r4, ‘l’`

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.

Reading Material

1. Appendix A in Microcontrollers and Microcomputers.