2.5. Programming numbers in assembly language

w is signed 8-bit -128 to +127
    or unsigned 8-bit 0 to 255
n is signed 8-bit -128 to +127
u is unsigned 8-bit 0 to 255
W is signed 16-bit -32787 to +32767
    or unsigned 16-bit 0 to 65535
N is signed 16-bit -32787 to +32767
U is unsigned 16-bit 0 to 65535
=\[addr\] 8-bit read from addr
={addr} 16-bit read from addr
[addr]= 8-bit write to addr
{addr}= 16-bit write to addr

\[\text{ldaa } \#w \text{ RegA} = w\]
\[\text{ldaa } u \text{ RegA} = \{u\}\]
\[\text{ldaa } U \text{ RegA} = [U]\]
\[\text{staa } u \text{ \{u\} = RegA}\]
\[\text{staa } U \text{ [U] = RegA}\]
\[\text{bra } U \text{ PC} = U\]

Figure 2.10. The ldaa Data instruction loads
2.6. Logical operations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A&amp;B</th>
<th>A</th>
<th>B</th>
<th>A^B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 2.14. Logical operations.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A&amp;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 2.12. implemented with discrete digital gates.**

Table 2.15. Logical complement.

<table>
<thead>
<tr>
<th>A</th>
<th>\sim A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\text{anda}</th>
<th>#w</th>
<th>RegA=RegA&amp;w</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{anda}</td>
<td>u</td>
<td>RegA=RegA&amp;[u]</td>
</tr>
<tr>
<td>\text{anda}</td>
<td>U</td>
<td>RegA=RegA&amp;[U]</td>
</tr>
<tr>
<td>\text{oraa}</td>
<td>#w</td>
<td>RegA=RegA</td>
</tr>
<tr>
<td>\text{oraa}</td>
<td>u</td>
<td>RegA=RegA</td>
</tr>
<tr>
<td>\text{oraa}</td>
<td>U</td>
<td>RegA=RegA</td>
</tr>
<tr>
<td>\text{eora}</td>
<td>#w</td>
<td>RegA=RegA^w</td>
</tr>
<tr>
<td>\text{eora}</td>
<td>u</td>
<td>RegA=RegA^[u]</td>
</tr>
<tr>
<td>\text{eora}</td>
<td>U</td>
<td>RegA=RegA^[U]</td>
</tr>
<tr>
<td>\text{coma}</td>
<td></td>
<td>RegA=\sim RegA</td>
</tr>
</tbody>
</table>

The \textbf{and} operation to extract, or \textit{mask}, individual bits

\texttt{Pressed = PTT\&0x01;}

Jonathan W. Valvano
Figure 2.13. Interface of a switch.

```
ldaa PTT     read input Port T
anda #$01    clear bits except bit 0
staa Pressed true iff PT0 is high

| a7  a6  a5  a4  a3  a2  a1  a0 | value of PTT
| 0   0   0   0   0   0   0   1 | $01 constant
| 0   0   0   0   0   0   0   a0 | result of the anda instruction
```

The or operation to set bits 4 and 5 of the register DDRT. The other six bits of DDRT remain constant.

*Friendly* software modifies just the bits that need to be.

```
DDRT |= 0x30; /*PT4,PT5 outputs */
```

```
ldaa DDRT     read previous value
oraa #$30    set bits 4 and 5
staa DDRT    update

| c7  c6  c5  c4  c3  c2  c1  c0 | value of DDRT
| 0   0   1   1   0   0   0   0 | $30 constant
| c7  c6  1   1   c3  c2  c1  c0 | result of the oraa instruction
```
**Maintenance Tip:** When interacting with just some of the bits of an I/O register it is better to modify just the bits of interest, leaving the other bits unchanged. In this way, the action of one piece of software does not undo the action of another piece.

The **exclusive or** operation can also be used to toggle bits.

\[ \text{PTM} ^= 0x08; \quad \text{/* toggle PM3 */} \]

```
ldaa PTM   read output Port M
eora #$08  toggle bit 3
staa PTM   update
```

The output of an **open collector gate**, drawn with the ‘\(x\)’, has two states low (0V) and HiZ (floating.)

![LED Interface Diagram](image)

**Figure 2.16. LED interface.**

The **and** operation can be used to clear bits.

\[ \text{PTM} &= 0xF7; \quad \text{/* PM3 becomes 0 */} \]
ldaa PTM   read output Port M
anda #$F7   clear just bit 3
staa PTM   update

\[
\begin{array}{cccccccc}
  b7 & b6 & b5 & b4 & b3 & b2 & b1 & b0 \\
  1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
  b7 & b6 & b5 & b4 & 0 & b2 & b1 & b0 \\
\end{array}
\]

$F7$ constant

result of the anda instruction

**Checkpoint 2.33:** Write assembly code that clears bit 1 of Port M.

**Checkpoint 2.34:** Write assembly code that sets bit 7 of Port M.

### 2.7. Shift operations

**LSR**

\[ \begin{array}{c}
  0 \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow C \\
\end{array} \]

*Figure 2.19. 8-bit logical shift right.*

**ASR**

\[ \begin{array}{c}
  C \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \\
\end{array} \]

*Figure 2.21. 8-bit arithmetic shift right.*

**LSL/ASL**

\[ \begin{array}{c}
  0 \leftarrow \leftarrow \leftarrow \leftarrow \leftarrow \leftarrow \leftarrow \leftarrow C \\
\end{array} \]

*Figure 2.22. 8-bit shift left.*
Figure 2.23. 8-bit roll right and 8-bit roll left.

### Instructions

- **asla**  
  \[ \text{RegA} = \text{RegA} \times 2 \]

- **lsla**  
  \[ \text{RegA} = \text{RegA} \times 2 \]

- **asra**  
  \[ \text{RegA} = \text{RegA} / 2 \]

- **lsra**  
  \[ \text{RegA} = \text{RegA} / 2 \]

- **rola**  
  roll left RegA

- **rora**  
  roll right RegA

### Maintenance Tip:

Use the **asla** instruction when manipulating signed numbers, and use the **lsla** instruction when shifting unsigned numbers.

High and Low are unsigned 4-bit components, which will be combined into a single unsigned 8-bit Result.

\[
\text{Result} = (\text{High} \ll 4) \mid \text{Low};
\]

The assembly code for this operation is

- **lda High**  
  read value of High

- **lsla**  
  shift into position

- **lsla**  
  lsla

- **lsla**  
  oraa Low  
  combine the two parts

- **staa Result**  
  save answer
Lab 4.2. Logic Function

Hardware circuit for Lab 4.2.

unsigned char N,M,P;
void main(void){  // make PT3 is output
    DDRT=0x08;      // PT1,PT0 are inputs
    while(1){
        N=(PTT&0x01)<<3;  // 8 iff switch N on
        M=(PTT&0x02)<<2;  // 8 iff switch M on
        P=N|M;            // 8 iff either is on
        PTT=P;            // activate as needed
    }
}

Approach -> start with Lab4_2 template

2.8. Arithmetic operations

Checkpoint 2.37: How many bits does it take to store the result of two unsigned 8-bit numbers added together?
**Checkpoint 2.39:** How many bits does it take to store the result of two unsigned 8-bit numbers multiplied together?

```
adda #w  RegA=RegA+w
adda u   RegA=RegA+[u]
adda U   RegA=RegA+[U]
suba #w  RegA=RegA−w
suba u   RegA=RegA−[u]
suba U   RegA=RegA−[U]
```

*condition code register (CC or CCR)*

C set after an **unsigned** add if the answer is wrong
V set a **signed** add if the answer is wrong

<table>
<thead>
<tr>
<th>bit</th>
<th>name</th>
<th>meaning after add or sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>negative</td>
<td>result is negative</td>
</tr>
<tr>
<td>Z</td>
<td>zero</td>
<td>result is zero</td>
</tr>
<tr>
<td>V</td>
<td>overflow</td>
<td>signed overflow</td>
</tr>
<tr>
<td>C</td>
<td>carry</td>
<td>unsigned overflow</td>
</tr>
</tbody>
</table>

Table 2.18. Condition code bits.

96+64 224+64
Observation: The carry bit, C, is set after an unsigned addition or subtraction when the result is incorrect.
Figure 2.29. Signed number wheel.

Figure 2.30. Signed number wheel

**Observation:** The overflow bit, V, is set after a signed addition or subtraction when the result is incorrect.

Let the result R be the result of the addition A+B. 
N bit is set
if unsigned result is above 127 or
if signed result is negative.
\( N = R7 \)

**Z bit** is set if result is zero.
\[ Z = \overline{R7} \& \overline{R6} \& \overline{R5} \& \overline{R4} \& \overline{R3} \& \overline{R2} \& \overline{R1} \& \overline{R0} \]

**V bit** is set after a signed addition if result is incorrect
\[ V = \overline{A7} \& \overline{B7} \& \overline{R7} + \overline{A7} \& \overline{B7} \& R7 \]

**C bit** is set after an unsigned addition if result is incorrect
\[ C = \overline{A7} \& B7 + \overline{A7} \& \overline{R7} + B7 \& \overline{R7} \]

**Checkpoint 2.41:** Assume Register A is initially -100. After executing the instruction `adda #64` what is the value in Register A, and the NZVC bits?

**Checkpoint 2.42:** Assume Register A is initially -100. After executing the instruction `adda #-64` what is the value in Register A, and the NZVC bits?

**Let the result R be the result of the subtraction A-B.**

**N bit** is set
if unsigned result is above 127 or
if signed result is negative.
\( N = R7 \)

**Z bit** is set if result is zero.
\[ Z = \overline{R7} \& \overline{R6} \& \overline{R5} \& \overline{R4} \& \overline{R3} \& \overline{R2} \& \overline{R1} \& \overline{R0} \]


**V bit** is set after a signed addition if result is incorrect

\[ V = A_7 \& B_7 \& R_7 + A_7 \& B_7 \& R_7 \]

**C bit** is set after an unsigned addition if result is incorrect

\[ C = A_7 \& B_7 + B_7 \& R_7 + A_7 \& R_7 \]

*Common Error:* Ignoring overflow (signed or unsigned) can result in significant errors.

*Observation:* Microcomputers have two sets of conditional branch instructions (if statements) that make program decisions based on either the C or V bit.

Promotion involves increasing the precision of the input numbers, and performing the operation at that higher precision.

<table>
<thead>
<tr>
<th>decimal</th>
<th>8-bit</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>224</td>
<td>1110,0000</td>
<td>0000,0000,1110,0000</td>
</tr>
<tr>
<td>+ 64</td>
<td>+0100,0000</td>
<td>+0000,0000,0100,0000</td>
</tr>
<tr>
<td>288</td>
<td>0010,0000</td>
<td>0000,0001,0010,0000</td>
</tr>
</tbody>
</table>

We can check the 16-bit intermediate result to see if the answer will fit back into the 8-bit result.
unsigned add
R = A + B

R = 255
R > 255
R < 255
end

promote A to A
promote B to B

R = R

16 16 16

ok

overflow

R = R

16

16

16

Figure 2.31. Promotion to detect and correct unsigned arithmetic errors.

To promote a signed number, we duplicate the sign bit
decimal 8-bit 16-bit
-96 1010,0000 1111,1111,1010,0000
-64 -0100,0000 -0000,0000,0100,0000
-160 0110,0000 1111,1111,0110,0000

signed add
promote A to A
promote B to B

R = A + B

R = -128
R < -128
R > -127
R = -128
R = 127
R = R

end

16 16

underflow

overflow

R = R

16

16

16

Figure 2.32. Flowcharts showing how to use promotion to detect and correct signed arithmetic errors.

Jonathan W. Valvano
**Common Error:** Even though most C compilers automatically promote to a higher precision during the intermediate calculations, they do not check for overflow when demoting the result back to the original format.

```
bcc l1    jump to l1 if C=0
bcs l2    jump to l2 if C=1
bvc l3    jump to l3 if V=0
bvs l4    jump to l4 if V=1
bpl l5    jump to l5 if V=0
bmi l6    jump to l6 if N=1
beq l7    jump to l7 if Z=0
bne l8    jump to l8 if Z=1
```

**ceiling and floor**

```
unsigned add
R=A+B

\[
\begin{align*}
\text{C} = 0 & \quad \text{R=255} \\
\text{C} = 1 & \quad \text{end}
\end{align*}
\]

unsigned sub
R=A-B

\[
\begin{align*}
\text{C} = 0 & \quad \text{R=0} \\
\text{C} = 1 & \quad \text{end}
\end{align*}
\]
```

Figure 2.33. Flowcharts showing how to use overflow bits to detect and correct unsigned arithmetic errors.

Assume A8 B8 and R8 are three 8-bit (1-byte) global variables defined in RAM.
A8    ds    1    Input
B8    ds   1    Input  
R8    ds   1    Output  

The following assembly language adds two unsigned 8-bit numbers, using the algorithm presented in Figure 2.33.

1d aa A8   get first input  
add a B8   A8+B8  
bcc  OK1  if C=0, then no error,  
1d aa #255 overflow  
OK1   staa R8

The following assembly language subtracts two unsigned 8-bit numbers.

1d aa A8   get first parameter  
sub a B8   A8–B8  
bcc  OK2  if C=0, then no error,  
1d aa #0 underflow  
OK2   staa R8