"Computers in the future may weigh no more than 1.5 tons"
Popular Science, 1949

3.5.5. Additional addressing modes on the 6812

1. **Indexed** addressing mode
   - fixed offset with the 16-bit registers: X, Y, SP, or PC
   - 5-bit (-16 to +15),
   - 9-bit (-256 to +255), or
   - 16-bit unsigned (0 to 65535) or signed (-32768 to +32767.)

**Properties of 16-bit addition**
- offset can be unsigned (0 to 65535) or
- signed (-32768 to +32767.)

**Indexed mode is useful**
- data structures
- parameters on the stack.
- local variables on the stack.

2. **Auto Pre/Post Decrement/Increment Indexed**
   - uses the 16-bit registers: X, Y, or SP.
   - register is modified either before (pre) or after (post)
   - amount added to (subtracted from) 1 to 8

In each case assume Reg Y is initially 2345.
Post-increment examples:

- `staa 1, Y+`  \[2345\]=RegA, then Reg Y=2346
- `staa 4, Y+`  \[2345\]=RegA, then Reg Y=2349

Pre-increment examples:
\begin{align*}
\text{staa 1,}+Y & \quad \text{Reg Y}=2346, \text{ then } [2346]=\text{RegA} \\
\text{staa 4,}+Y & \quad \text{Reg Y}=2349, \text{ then } [2349]=\text{RegA}
\end{align*}

Post-decrement examples:
\begin{align*}
\text{staa 1,}Y- & \quad [2345]=\text{RegA}, \text{ then } \text{Reg Y}=2344 \\
\text{staa 4,}Y- & \quad [2345]=\text{RegA}, \text{ then } \text{Reg Y}=2341
\end{align*}

Pre-decrement examples:
\begin{align*}
\text{staa 1,}^{-Y} & \quad \text{Reg Y}=2344, \text{ then } [2344]=\text{RegA} \\
\text{staa 4,}^{-Y} & \quad \text{Reg Y}=2341, \text{ then } [2341]=\text{RegA}
\end{align*}

\textbf{Observation: Usually we would add/subtract one when accessing an 8-bit value and add/subtract two when accessing a 16-bit value.}
LDAA

**Load Accumulator A**

**Operation:** 
(M) ⇒ A

**Description:** Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

**Condition Codes and Boolean Formulas:**

```
S X H I N Z V C
- - - - Δ Δ 0 -
```

- **N:** Set if MSB of result is set; cleared otherwise.
- **Z:** Set if result is $0000$; cleared otherwise.
- **V:** $\Theta$; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Cycles</th>
<th>Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA #apr8</td>
<td>IMM</td>
<td>86 11</td>
<td>1</td>
<td>P</td>
</tr>
<tr>
<td>LDAA opr8a</td>
<td>DIR</td>
<td>96 dd</td>
<td>3</td>
<td>rFP</td>
</tr>
<tr>
<td>LDAA opr16a</td>
<td>EXT</td>
<td>B6 hh ll</td>
<td>3</td>
<td>rOP</td>
</tr>
<tr>
<td>LDAA oprx0,xyzp</td>
<td>IDX</td>
<td>A6 xb</td>
<td>3</td>
<td>rFP</td>
</tr>
<tr>
<td>LDAA oprx9,xyzp</td>
<td>IDX1</td>
<td>A6 xb ff</td>
<td>3</td>
<td>rPO</td>
</tr>
<tr>
<td>LDAA oprx16,xyzp</td>
<td>IDX2</td>
<td>A6 xb ee ff</td>
<td>4</td>
<td>frFP</td>
</tr>
<tr>
<td>LDAA [x,xyzp]</td>
<td>[D,IDX]</td>
<td>A6 xb</td>
<td>6</td>
<td>fIrfrFP</td>
</tr>
<tr>
<td>LDAA [oprx16,xyzp]</td>
<td>[IDX2]</td>
<td>A6 xb ee ff</td>
<td>6</td>
<td>EIPfrFP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rr</th>
<th>register</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>X</td>
</tr>
<tr>
<td>01</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>SP</td>
</tr>
<tr>
<td>11</td>
<td>PC</td>
</tr>
</tbody>
</table>

Table 3.23. The MC68HC812A4 indexed-register code.

**postbyte,xb syntax mode explanations**

| rr0000000 | ,r | IDX | 5-bit constant, n=0 |

Jonathan W. Valvano
The 6812 load effective address instructions

leax idx RegX=EA
leay idx RegY=EA
leas idx  \text{RegS}=\text{EA}

effective address is calculated in the usual manner. puts the effective address itself into the register

\begin{align*}
\text{leax } m, r & \quad \text{IDX 5-bit, } \text{EA}=r+m \\
\text{leax } v, +r & \quad \text{IDX pre-inc } r=r+v, \text{ EA}=r \\
\text{leax } v, -r & \quad \text{IDX pre-dec } r=r-v, \text{ EA}=r \\
\text{leax } v, r+ & \quad \text{IDX post-incr, } \text{EA}=r, \text{ r}=r+v \\
\text{leax } v, r- & \quad \text{IDX post-decr, } \text{EA}=r, \text{ r}=r-v \\
\text{leax } A, r & \quad \text{IDX Reg A offset } \text{EA}=r+A, \\
\text{leax } B, r & \quad \text{IDX Reg B offset } \text{EA}=r+B, \\
\text{leax } D, r & \quad \text{IDX Reg D offset } \text{EA}=r+D \\
\text{leax } q, r & \quad \text{IDX1 9-bit } \text{EA}=r+q \\
\text{leax } W, r & \quad \text{IDX2 16-bit } \text{EA}=r+W
\end{align*}

where \( r \) is Reg X, Y, SP, or PC, and the fixed constants are

\begin{align*}
\text{m} & \quad \text{is any signed 5-bit -16 to +15} \\
\text{q} & \quad \text{is any signed 9-bit -256 to +255} \\
\text{v} & \quad \text{is any unsigned 3 bit 1 to 8} \\
\text{W} & \quad \text{is any signed 16-bit -32768 to +32767} \\
& \quad \text{or any unsigned 16-bit 0 to 65535}
\end{align*}

\textbf{Observation:} The \texttt{leas -4,sp} instruction subtracts four from the stack pointer.
Review: simplified execution has five phases:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Function</th>
<th>R/W</th>
<th>Address</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Op code fetch</td>
<td>read</td>
<td>PC++</td>
<td>Put data in IR</td>
</tr>
<tr>
<td>2</td>
<td>Operand fetch</td>
<td>read</td>
<td>PC++</td>
<td>Calculate EA</td>
</tr>
<tr>
<td>3</td>
<td>Data read</td>
<td>read</td>
<td>SP, EAR</td>
<td>Data (ALU)</td>
</tr>
<tr>
<td>4</td>
<td>Free cycle</td>
<td>read</td>
<td>PC/SP/$FFFF</td>
<td>ALU (sets CCR)</td>
</tr>
<tr>
<td>5</td>
<td>Data store</td>
<td>write</td>
<td>SP, EAR</td>
<td>Results stored</td>
</tr>
</tbody>
</table>

**machine code** execution time (cycles) **source code**

- \$F02F FE0800 [3] (45) {ROP} ldx $0800
- \$F032 7E0802 [3] (48) {WOP} stx $0802
- \$F035 720800 [4] (51) {rOPw} inc $0800

address total execution time (cycles) description

grouped by instruction as seen at the BIU/memory interface

```plaintext
ldx  stx  inc
previous...P}{ROP}{WOP}{rOPw}{next...}
```
3.2.5. Simulated 6812 machine language execution

<table>
<thead>
<tr>
<th>actual 6812 cycle-by-cycle</th>
<th>simplified cycle-by-cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>sometimes 8, sometimes 16-bit data</td>
<td>sequence of 8-bit accesses</td>
</tr>
<tr>
<td>special case for misaligned 16-bit</td>
<td>even/odd addresses treated the same</td>
</tr>
<tr>
<td>pipeline enhances execution speed</td>
<td>simple fetch-execute sequence</td>
</tr>
<tr>
<td>fetches op codes for later execution</td>
<td>op codes for immediate execution</td>
</tr>
<tr>
<td>fetches op codes that are never executed</td>
<td>fetched op codes are always executed</td>
</tr>
<tr>
<td>68HC812A4 allows for 22-bit address</td>
<td>always 16-bit address</td>
</tr>
<tr>
<td>variable length off-chip accesses</td>
<td>all accesses exactly same cycle period</td>
</tr>
</tbody>
</table>

Table 3.3. Differences between a real 6812 and the TExaS bus cycle simulation.

During a read cycle (R/W=1),
- memory at specified address puts information on data bus, and
- processor transfers information into appropriate place within the processor.

**instruction fetch.** The address is the PC and the 8-bit data is loaded into the instruction register, IR.

**operand fetch.** The address is also the PC, but the 8-bit data is used to calculate the effective address.

**data fetch.** The address is the EAR, and the 8-bit data is loaded into a register or sent to the ALU.

**stack pull.** First, the 8-bit data is read from memory pointed to by SP and stored in a register, then the stack pointer is incremented SP=SP+1.

During a write cycle (R/W=0),
- processor puts information on data bus, and
- memory transfers information into specified location

**data write.** The 8-bit data from a register or ALU is stored in memory at the address specified by the EAR.

**stack write.** First, the stack pointer is decremented SP=SP-1, then the 8-bit data from a register is stored in memory at the address specified by the SP.

The simplified execution has five phases:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Function</th>
<th>R/W</th>
<th>Address</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Op code fetch</td>
<td>read</td>
<td>PC++</td>
<td>Put data in IR</td>
</tr>
<tr>
<td>Step</td>
<td>Instruction</td>
<td>Read/Write</td>
<td>Address</td>
<td>Function</td>
</tr>
<tr>
<td>------</td>
<td>----------------------</td>
<td>------------</td>
<td>---------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>Operand fetch</td>
<td>read</td>
<td>PC++</td>
<td>Calculate EA</td>
</tr>
<tr>
<td>3</td>
<td>Data read</td>
<td>read</td>
<td>SP, EAR</td>
<td>Data (ALU)</td>
</tr>
<tr>
<td>4</td>
<td>Free cycle</td>
<td>read</td>
<td>PC/SP/$FFFF</td>
<td>ALU (sets CCR)</td>
</tr>
<tr>
<td>5</td>
<td>Data store</td>
<td>write</td>
<td>SP, EAR</td>
<td>Results stored</td>
</tr>
</tbody>
</table>