### 2.12. Tutorial 2. Arithmetic and logical operations

<table>
<thead>
<tr>
<th>format</th>
<th>descriptions</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>8-bit unsigned hexadecimal</td>
<td>$00$ $12$ $FF$</td>
</tr>
<tr>
<td>d</td>
<td>8-bit unsigned decimal</td>
<td>0 18 255</td>
</tr>
<tr>
<td>b</td>
<td>8-bit unsigned binary</td>
<td>%00000000 %00010010</td>
</tr>
<tr>
<td>H</td>
<td>16-bit unsigned hexadecimal</td>
<td>$0000$ $1234$ $FFFF$</td>
</tr>
<tr>
<td>D</td>
<td>16-bit unsigned decimal</td>
<td>0 4660 65535</td>
</tr>
<tr>
<td>B</td>
<td>16-bit unsigned binary</td>
<td>%00010001000110100</td>
</tr>
<tr>
<td>-h or +h</td>
<td>8-bit signed hexadecimal</td>
<td>-$80$ $+$12 $+$7F</td>
</tr>
<tr>
<td>-d or +d</td>
<td>8-bit signed decimal</td>
<td>-128 +18 +127</td>
</tr>
<tr>
<td>-b or +b</td>
<td>8-bit signed binary</td>
<td>-$%10000000$ $+%00010010$</td>
</tr>
</tbody>
</table>
-H or +H 16-bit signed hexadecimal -$8000 +$1234 +$7FFF
-D or +D 16-bit signed decimal -32768 +4660 +32767
-B or +B 16-bit signed binary -+%0001001000110100
b3 3-bit binary (lsb) %000 %111
b4 4-bit binary (lsb) %0000 %1111
cc 8-bit binary CCR sXhInzvc
C or C ASCII character 'A' 'x' '0'
s or S ASCII string "Hello World"
v address itself, unsigned dec 2048
V address itself, unsigned hex $0800
+v or −v address itself, signed dec -32768 +0 +32767
+V or −V address itself, signed hex −$8000 +0 +$7FFF

Table 2.23. Available formats for displaying ViewBox.

Chapter 3 objectives are to:
• Present basic microcomputer architecture,
• List available 6812 microcomputers and their memory configurations,
• Define some of machine level instructions available on the 6812,
• Explain how the computer uses addressing modes to access memory,
• Introduce I/O ports

3.1. Introduction

Figure 3.1. A memory-mapped computer system.

The memory maps
9S12C32 MC68HC812A4
I/O $0000 to $03FF $0000 to $01FF
Ports AD, M, S, T A,B,C,D,E,F,H,J,S,T,AD
RAM $3800 to $3FFF $0800 to $0BFF
2K 1K
ROM $4000 to $7FFF $F000 to $FFFF
$8000 to $FFFF
32K 4K

******************* JWVLab4_2.RTF *******************
* The overall purpose EE319K Lab 4.2 Fall 2004

* Jonathan W. Valvano and your TA
* Robin Tsang or Nachiket Kharalkar
* Creation date: 8/4/2004 11:29:47 AM
* Last modification date: 8/24/2004 8:57:55 PM
* External hardware:
  * PT3 output connected to LED, P
  * PT1 input connected to a switch, M
  * PT0 input connected to a switch, N
  * Detailed description
  * Two input switches N,M
  * One output LED P
  * Function P = N|M
  * Define Input/Output ports
  PTT equ $0240
  DDRT equ $0242
* Global variables
  org $3800 globals go in RAM
  M rmb 1 boolean, true if pressed
  org $4000 object code goes in ROM
  main lds #$4000 initialize stack
  * stuff executed once, at start up
  loop
  * stuff executed over and over forever
    bra loop repeat
  * reset vector
    org $FFFE
    fdb main starting address after a RESET

Jonathan W. Valvano
**Isolated I/O System**

- Intel x86 processor
- RAM
- ROM
- Input Devices
- Output Devices
- I/O Control
  - IOR, IOW
- Address, Data

**Figure 3.2. An isolated-I/O computer system.**

<table>
<thead>
<tr>
<th>Memory mapped</th>
<th>I/O Mapped</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldax 0</td>
<td>in al,0</td>
<td>input port</td>
</tr>
<tr>
<td>staa M</td>
<td>mov M,al</td>
<td>save in memory</td>
</tr>
<tr>
<td>staa 1</td>
<td>out 1,al</td>
<td>output port</td>
</tr>
</tbody>
</table>

**bus**
- **address** where or which module
- **data** what
- **control** when and direction

**CPU Read Cycle**

- processor
- RAM
- ROM
- input devices
- output devices
- Bus

Jonathan W. Valvano
Figure 3.3. A read cycle copies data from RAM, ROM or input device into the processor.

![Read Cycle Diagram]

Figure 3.4. A write cycle copies data from the processor into RAM, or output device.

**Checkpoint 3.2:** The 6812 has a 16-bit address bus and a 16-bit data bus, but can still only address 65536 bytes of memory. Why?

![Checkpoint Diagram]

Figure 3.7. The four basic components of 6812 processor.

The **bus interface unit** (BIU)
- reads data from the bus during a read cycle,
- writes data onto the bus during a write cycle,
- always drives the address bus and the control signals
- **effective address register** (EAR) contains the data address

The **control unit** (CU)
- orchestrates the sequence of operations
- issues commands to ALU, BIU
- **instruction register** (IR) contains the op code

Jonathan W. Valvano
The **registers**
- high-speed storage devices located in the processor
- do not have addresses like regular memory
- specific functions explicitly defined by the instruction
- **Accumulators** contain data (A, B, D)
- **Index registers** contain addresses (X, Y)
- **Program counter** (PC) points to instruction to execute next
- **Stack pointer** (SP) points to the top element on the stack
  - context switch when calling and returning from a function
  - pass parameters
  - save temporary information
  - implement local variables
- **Condition code register** (CCR) the status of the previous operation

![Condition Code Register](image)

*Figure 3.8. The 6811 and 6812 have 6 registers.*

![Arithmetic Logic Unit](image)

*Figure 3.9. The 6811/6812 condition code bits.*

The **arithmetic logic unit** (ALU)
- Arithmetic operations
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 into IR,</td>
</tr>
<tr>
<td>2</td>
<td>Operand fetch</td>
<td>read</td>
<td>PC++</td>
<td>Immediate or calculate EA</td>
</tr>
<tr>
<td>3</td>
<td>Data read</td>
<td>read</td>
<td>SP,EAR</td>
<td>Data passes through ALU,</td>
</tr>
<tr>
<td>4</td>
<td>Free cycle</td>
<td>read</td>
<td>PC/SP/$FFFF</td>
<td>ALU operations, set CCR</td>
</tr>
<tr>
<td>5</td>
<td>Data store</td>
<td>write</td>
<td>SP,EAR</td>
<td>Results stored in memory</td>
</tr>
</tbody>
</table>

9S12C32
In the **single chip operating mode**, the 9S12C32 is a single chip complete microcomputer (microcontroller), where all its I/O ports are available.

### 3.2.2. Terminology

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

The **label field**
- optional
- starts in the first column
- used to identify the position in memory
- must choose a unique name for each label

The **opcode field**
- specifies the microcomputer command to execute.
- could be *pseudo op*, which are instructions to the assembler

The **operand field**
- specifies where to find the data to execute the instruction
- 0, 1, 2, or 3 operands

The **comment field**
- optional
- ignored by the computer
- makes it easier to understand
- explain how it works
- why design choices were made
- how to test it
- how to change it

<table>
<thead>
<tr>
<th>label</th>
<th>opcode</th>
<th>operand</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>here</td>
<td>ldaa</td>
<td>100</td>
<td>RegA={$0064}</td>
</tr>
</tbody>
</table>
Object code instruction comment
$96 $64 ldaa 100 RegA=[$0064]

3.2.3. Addressing modes
Where to find the data?

- **inherent** no operand or implied operand
- **immediate** in the instruction itself
- **direct** or **extended** absolute address of the data
- **relative** where to go for branch instructions

<table>
<thead>
<tr>
<th>object</th>
<th>op code</th>
<th>operand</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$87</td>
<td>clra</td>
<td>A = 0</td>
<td>(inherent)</td>
</tr>
<tr>
<td>$86 24</td>
<td>ldaa</td>
<td>#36 A = $24</td>
<td>(immediate)</td>
</tr>
</tbody>
</table>

Jonathan W. Valvano
Table 3.2. Simple addressing modes.

1. **Inherent** addressing mode has no operand field
   - sometimes there is no data
     - `stop`
   - sometimes the data for the instruction is implied.
     - `clra`
   - sometimes the data must be fetched, but the address is implied
     - `pula`

2. **Immediate** addressing mode uses a fixed data constant.

```
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:

<table>
<thead>
<tr>
<th>S</th>
<th>X</th>
<th>H</th>
<th>I</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>A</td>
<td>A</td>
<td>0</td>
<td>–</td>
</tr>
</tbody>
</table>

N: Set if MSB of result is set; cleared otherwise.
Z: Set if result is $00$; cleared otherwise.
V: 0; 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>LDA #0024</td>
<td>IMM</td>
<td>$86 11</td>
<td>1</td>
<td>p</td>
</tr>
<tr>
<td>LDA opr8a</td>
<td>DIR</td>
<td>$96 dd</td>
<td>3</td>
<td>rFP</td>
</tr>
<tr>
<td>LDA opr16a</td>
<td>EXT</td>
<td>$86 hh 11</td>
<td>3</td>
<td>rOP</td>
</tr>
<tr>
<td>LDA oprx0, ysp</td>
<td>IDX</td>
<td>A6 xb</td>
<td>3</td>
<td>rFP</td>
</tr>
<tr>
<td>LDA oprx9, ysp</td>
<td>IDX1</td>
<td>A6 xb ff</td>
<td>3</td>
<td>rPO</td>
</tr>
<tr>
<td>LDA oprx16, ysp</td>
<td>IDX2</td>
<td>A6 xb ee ff</td>
<td>4</td>
<td>frrPP</td>
</tr>
<tr>
<td>LDA [D, ysp]</td>
<td>[D,IDX]</td>
<td>A6 xb</td>
<td>6</td>
<td>frrPP</td>
</tr>
<tr>
<td>LDA [opr16, ysp]</td>
<td>[IDX2]</td>
<td>A6 xb ee ff</td>
<td>6</td>
<td>frrPP</td>
</tr>
</tbody>
</table>
```

Jonathan W. Valvano
The TExaS simulation of this instruction shows the following two cycles.
Opcode fetch   R 0xF801 0x86 from EEPROM
Operand fetch  R 0xF802 0x24 from EEPROM

3. **Direct Page** addressing mode uses an 8-bit address
   access from addresses 0 to $00FF
called zero-page.
on the 6812 they reference the I/O ports
the < operator forces direct addressing

The TExaS 6812 simulation this instruction shows the following three cycles.
Opcode fetch   R 0xF801 0x96 from EEPROM
Operand fetch  R 0xF802 0x24 from EEPROM
Fetch using EARR 0x0024 0x57 from I/O

4. **Extended addressing** mode uses a 16-bit address
   access all memory and I/O devices
outside Motorola family this addressing mode is called direct
the > operator forces extended addressing
Figure 3.12. Example of the extended addressing mode.

The TExaS 6812 simulation of this instruction shows the following four cycles.

- Opcode fetch: R 0xF801 0xB6 from EEPROM
- Operand fetch: R 0xF802 0x08 from EEPROM
- Operand fetch: R 0xF803 0x01 from EEPROM
- Fetch using EARR 0x0801 0x62 from RAM

**Common Error:** It is wrong to assume the `<` and `>` operators affect the amount of data that is transferred. The `<` and `>` operators will affect the addressing mode, that is how the address is represented.

**Observation:** Accesses to Registers A, B and CC transfer 8 bits, while accesses to Registers D, X, Y, SP, and PC transfer 16 bits regardless of the addressing mode.

5. **PC Relative** addressing mode is used for the branch instructions stored in the machine code is not the absolute address but the 8-bit signed offset relative distance from the current PC value the PC already points to the next instruction the assembler calculates it for us.
1) address of current instruction
$F880  bra  $F840
2) op code of branch instruction, and size of instruction
$F880  $20rr  bra  $F840
3) address of next instruction
$F880  $20rr  bra  $F840
$F882
4) calculate PC relative addressing
   \[ rr = \text{destination} - \text{address of next instruction} \]
   \[ = $F840 - $F882 = -$42 = $BE \]
The object code for this instruction will be $20BE.
\begin{verbatim}
  \textbf{rr} = \text{destination} - \text{address of next instruction}
  = \$F044 - \$F002 = \$42

  The object code for this instruction will be \$2042.

1) address of current instruction
  \$F000 \quad \text{bra} \quad \$F144

2) op code of branch instruction, and size of instruction
  \$F000 \quad \$20rr \quad \text{bra} \quad \$F144

3) address of next instruction
  \$F000 \quad \$20rr \quad \text{bra} \quad \$F144
  \$F002

4) calculate PC relative addressing
   \textbf{rr} = \text{destination} - \text{address of next instruction}
   = \$F144 - \$F002 = \$142

  “Branch out of range” assembly error.
\end{verbatim}