Introduction to Microcontrollers II

brset, brclr

Indexed Addressing Example

µp Laboratory #2
BUFFALO
Assembling Code
EECE 143 Digital Design Project

**Purpose:** To allow students to design their own digital project in order to demonstrate the utilization of digital design concepts.

**Examples:** Digital Alarm Clock, Traffic Light Simulation, Soda pop machine controller Scoreboard Display with timers

**Design Criteria:** If required, all clock sources must be designed and built by students. You may use IC’s available in the Open Lab. You may use discrete LS or HC ICs as well as GALs.

**Grading:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
<th>Due/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Proposal</td>
<td>10</td>
<td>DUE: Monday, November 5&lt;sup&gt;th&lt;/sup&gt; in class</td>
</tr>
<tr>
<td>(Preliminary design diagrams, flowcharts)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Project</td>
<td>30</td>
<td>Presentations &amp; Reports: Thursday, Dec 6th</td>
</tr>
<tr>
<td>(Neatness of design, design complexity Making it work)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentations</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>(Visual Aids, Ability to communicate design, Peer, TA and Teacher Evals)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Report (Writeup)</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>(Purpose, Preparation, Experiment Procedure, Design Description, Schematics, Flowcharts, Code, Future Design Improvements and Considerations)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total** 200
Digital Design Projects

- Digital Alarm Clock
- Digital Temperature Buffer/Storage
- Soda Pop Machine with Password for Free Soda
- Car Turning Signal Light Controller
- Electric Door Lock Control Circuit
- Traffic Light Signal Controllers
- Digital Tic-Tac-Toe
- Digital Slot Machine
- Digital Kitchen Timer
- Sports Scoreboard with Timer
- Digital Sampler and Playback
Design Project Presentation

• Thursday December 6
• 5-10 minutes to present your project and demonstrate
• You will be graded on
  – How well you communicate your design
  – How well you answer questions
  – Audio/visual aids (Powerpoint presentations, poster boards…)

© J. Chris Perez 2001
Design Project Written Report

• Due at time of presentation
• Typed
• Should include the following
  – Purpose
  – Design Description: A module by module description of your design.
  – Equations used in your design. Truth Tables…
  – COMPLETE schematic diagrams, block diagrams
  – Flow charts and Code (if applicable)
  – Testing Procedures: Give a step by step account of how you test to verify your circuit works.
  – Future Design Improvements and Conclusion
Design Project Grade Breakdown

• Design Proposal 10
• Design Project 30
• Presentations 60
• Written Report 100
Total 200
A Look at Appendix A of the HC11 Reference Manual
Load Accumulator

LDA

Load Accumulator

Operation: AccX \leftarrow (M)

Description: Loads the contents of memory into the 8-bit accumulator. The condition codes are set according to the data.

Condition Codes and Boolean Formulae:

<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></td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

N  R7
Set if MSB of result is set; cleared otherwise

Z  R7' \cdot R6' \cdot R5' \cdot R4' \cdot R3' \cdot R2' \cdot R1' \cdot R0'
Set if result is $00$; cleared otherwise

V  0
cleared

Source Form: LDAA (opr); LDAB (opr)
Addressing Modes, Machine Code, and Cycle-by-Cycle Execution:

<table>
<thead>
<tr>
<th>Cycle</th>
<th>LDAA (IMM)</th>
<th>LDAA (DIR)</th>
<th>LDAA (EXT)</th>
<th>LDAA (IND,X)</th>
<th>LDAA (IND,Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Addr</td>
<td>Data</td>
<td>R/W*</td>
<td>Addr</td>
<td>Data</td>
</tr>
<tr>
<td>1</td>
<td>OP</td>
<td>86</td>
<td>1</td>
<td>OP+1</td>
<td>ii</td>
</tr>
<tr>
<td>2</td>
<td>OP+1</td>
<td>ii</td>
<td>1</td>
<td>OP+1</td>
<td>hh</td>
</tr>
<tr>
<td>3</td>
<td>00dd</td>
<td>(00dd)</td>
<td>1</td>
<td>OP+2</td>
<td>ll</td>
</tr>
<tr>
<td>4</td>
<td>hhll</td>
<td>(hhll)</td>
<td>1</td>
<td>X+ff</td>
<td>(X+ff)</td>
</tr>
<tr>
<td>5</td>
<td>Y+ff</td>
<td>(Y+ff)</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle</th>
<th>LDAB (IMM)</th>
<th>LDAB (DIR)</th>
<th>LDAB (EXT)</th>
<th>LDAB (IND,X)</th>
<th>LDAB (IND,Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Addr</td>
<td>Data</td>
<td>R/W*</td>
<td>Addr</td>
<td>Data</td>
</tr>
<tr>
<td>1</td>
<td>OP</td>
<td>C6</td>
<td>1</td>
<td>OP+1</td>
<td>ii</td>
</tr>
<tr>
<td>2</td>
<td>OP+1</td>
<td>ii</td>
<td>1</td>
<td>OP+1</td>
<td>hh</td>
</tr>
<tr>
<td>3</td>
<td>00dd</td>
<td>(00dd)</td>
<td>1</td>
<td>OP+2</td>
<td>ll</td>
</tr>
<tr>
<td>4</td>
<td>hhll</td>
<td>(hhll)</td>
<td>1</td>
<td>X+ff</td>
<td>(X+ff)</td>
</tr>
<tr>
<td>5</td>
<td>Y+ff</td>
<td>(Y+ff)</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Motorola M68HC11 Reference Manual A-17
brset command

• Branch if Bit(s) Set:
  Performs the logical AND of location M inverted and the mask supplied with the instruction, then branches if the result is zero (only if all bits corresponding to ones in the mask byte are ones in the tested byte)

brset $1004 %00001111 GOHERE

Will branch to location GOHERE if 4 LSBs of PortE are all 1’s
brclr command

• Branch if Bit(s) Clear:

Performs the logical AND of location M and the mask supplied with the instruction, then branches if the result is zero (only if all bits corresponding to ones in the mask byte are zeros in the tested byte)

brset $1004 %00001111 GOHERE

Will branch to location GOHERE if 4 LSBs of PortE are all 0’s
Example:

```
count_br.lst
```

This program uses the brset brclr commands which check to see if a bit or bits are set or cleared.
When you compare this code with that of count.a11 you will see it is a lot shorter and easier to understand
Using Indexed Addressing in Loops

PORTE equ $100A ; Set definition for PORTE
BUFFALO equ $e00a ; Set definition for BUFFALO

org $c000 ; Store program at $c000
ldx #$D000 ; Load initial value of x register

READ ldaa PORTE ; Load value into AccA
staa 0,x ; Store value of AccA into memory location
inx ; Increment Register x
cmpx #$D00A ; Is x Register = $D00A?
blt READ ; If less than get another value
jmp BUFFALO ; If it is, end program
end
ldx #$D000 : Loads Index Register X with $D000
immediate mode

AccA

IndexX

$D000

© J. Chris Perez 2001
ldaa PORTE : loads Acca with the contents of PORTE ($100A)

AccA  $00

IndexX $D000
staa 0,x ; Puts the contents of AccA into the memory location [D000 + 0]

AccA $00

IndexX $D000

$00

$100A

D000
D001
D002
D003

D009
D00A
inx : increments the Index register X

AccA

$00

IndexX

$D001

$00

$100A

D000

D001

D002

D003

D009

D00A

© J. Chris Perez 2001
cmpx #$D00A : compares the contents of index register X with $D00A immediate mode

AccA $00

IndexX $D001

$00 $100A

D000 D001 D002 D003

$00

D009 D00A

<=$D00A?

If less than, it branches back to READ, otherwise jmp BUFFALO
ldaa PORTE : loads Acca with the contents of PORTE ($100A)

AccA $00

IndexX $D001

© J. Chris Perez 2001
staa 0,x ;Puts the contents of AccA into the memory location [D001 + 0]
**inx : increments the Index register X**

- **AccA**: $00
- **IndexX**: $D002

```
$00
$00
$00
$00
$00
D000
D001
D002
D003
D009
D00A
```
cmpx #$D00A : compares the contents of index register X with $D00A immediate mode

AccA $00

IndexX $D001

=$D00A?

If less than, it branches back to READ, otherwise jmp BUFFALO

© J. Chris Perez 2001
Guide to A11 Files

PORTB equ $1004
org $C000
MAIN clra
  staa PORTB
WAIT0 ldab $100a
  andb #$01
  bne WAIT0
WAIT1 LDAB $100a
  andb #$01
  beq WAIT1
  adda #$01
  daa
  staa $1004
  jmp WAIT0

;constant declarations should be put before the org statement
;org <address to store program>
;labels and constants should start in column 1, all other
  statements should be tabbed to the right
;Default is decimal, use $ for HEX or % for binary
;Anything after a “;” or a * is a comment
AS11 Common Mistakes

see page 9-23

• Missing “#” Motorola defaults to direct or extended addressing mode.
• Using signed branches on unsigned data. AS11 has both signed and unsigned conditional branches. Unsigned (BHI BLO BHS BLS) Signed (BGT BLT BGE BLE)
• Flow charts drawn after code is written. Know what you want to do before you try to write the code.
• Missing a “$”. AS11 defaults to decimal. NOTE: BUFFALO only allows HEX. No “$” is needed when using the onboard assembler ASM.
• Loading a 16-bit value into an 8-bit location or vice versa.
• Improper ending for embedded code. Use (in)finite loop or jump to BUFFALO.
• On reads: high byte first, low byte second.
• Errors when pushing and pulling from stack.
• Memory usage
• high byte AccD == AccA, low byte AccD == AccB
Code Size and Execution Time

0001  * IndStor.a11
0002  * J. Chris Perez
0003  * Program uses indexed registering to loop
0004  * through a set of instructions
0005  * to increment value located
0006  * in AccA and store result in consecutive
0007  * Memory locations
0008
0009 c000          org $c000 ;Set location of program in memory
0010 c000 4f       clra ;Clear Accumulator A
0011 c001 ce d0 00  ldx #$d000 ; Load Index X with initial location
0012 c004 a7 00    HERE staa 0,x ;Store contents of AccA
0013 c006 8b 01    adda #$01 ;Increment A
0014 c008 08       inx ;Increment Index X
0015 c009 8c d0 0a  cp x #$d00a ;Compare to $d00a
0016 c00c 2d f6    blt HERE ; If less than, jump to store new value
0017 c00e 7e e0 0a  jmp $e00a ;Jump to BUFFALO
HERE    c004 *0012 0016
Program is in C000-C010, therefore the program takes up 17 memory spaces.

    0001                         * IndStor.a11
    0002                         * J. Chris Perez
    0003                         * Program uses indexed registering to loop
    0004                         * through a set of instructions
    0005                         * to increment value located
    0006                         * in AccA and store result in consecutive
    0007                         * Memory locations
    0008
    0009  c000                 org $c000 ;Set location of program in memory
    0010  c000 4f               clra    ;Clear Accumulator A
    0011  c001 ce d0 00         ldx #$d000 ; Load Index X with initial location
    0012  c004 a7 00            HERE staa 0,x ;Store contents of AccA
    0013  c006 8b 01            adda #$01 ;Increment A
    0014  c008 08               inx    ;Increment Index X
    0015  c009 8c d0 0a         cpx #$d00a ;Compare to $d00a
    0016  c00c 2d f6            blt HERE ; If less than, jump to store new value
    0017  c00e 7e e0 0a         jmp $e00a ;Jump to BUFFALO
    HERE   c004 *0012 0016

Program is in C000-C010, therefore the program takes up 17 memory spaces.
This program is in C000-C009, therefore the program takes up 10 memory spaces.

0001 * INNOT2.A11  Read word, NOT it, Write it
0002 *    Bruce Hoeppner 01 JAN 94
0003 *    Chris Perez 14 June 2000
0004 *
0005 *    Read an 8-bit word from PortE.
0006 *    Complement the word.
0007 *    Write the word to PortB.
0008 * Definitions
0009 1004 PORTB equ $1004
0010 e00a BUFFALO equ $e00a
0011 ********************************************
0012 * Load program into 8k user RAM
0013 c000 org $C000
0014 *
0015 1004 ldaa $100a
0016 *    Read 8-bit word from PortE into AccA
0017 c003 43 coma
0018 *    Complement the word.
0019 c004 b7 10 04 staa PORTB
0020 *
0021 *    Jump back to beginning of program
0022 7e e0 0a jmp BUFFALO beginning of program
BUFFALO e00a *0010 0022
PORTB 1004 *0009 0019

© J. Chris Perez 2001
This program takes 13 cycles \((6.5 \times 10^{-6} \text{ sec})\) to execute.

From Appendix A:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>addressing mode</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldaa</td>
<td>extended</td>
<td>4</td>
</tr>
<tr>
<td>coma</td>
<td>inherent</td>
<td>2</td>
</tr>
<tr>
<td>staa</td>
<td>extended</td>
<td>4</td>
</tr>
<tr>
<td>jmp</td>
<td>extended</td>
<td>3</td>
</tr>
</tbody>
</table>

Total: 13 cycles

13 cycles/freq = 13 / 2MHz = 6.5 \times 10^{-6} \text{ sec}

```
0001                         * INNOT2.A11  Read word, NOT it, Write it
0002                         *    Bruce Hoeppner     01 JAN 94
0003                         *    Chris Perez        14 June 2000
0004                         *
0005                         *    Read an 8-bit word from PortE.
0006                         *    Complement the word.
0007                         *    Write the word to PortB.
0008                         *
0009                         * Definitions
0010 1004                    PORTB  equ  $1004
0011 e00a                    BUFFALO equ  $e00a
0012                         ********************************************
0013                         * Load program into 8k user RAM
0014 c000 0012                org  $C000
0015                         * Read 8-bit word from PortE into AccA
0016 c000 b6 10 0a            ldaa $100a
0017                         * Complement the word.
0018                         * Write the word to PortB
0019 c004 b7 10 04            staa PORTB
0020                         * Jump back to beginning of program
0021 c007 7e e0 0a            jmp  BUFFALO beginning of program
0022 BUFFALO e00a *0010 0022
0023 PORTB 1004 *0009 0019
```

© J. Chris Perez 2001
Prelab: Create flowcharts and write code for the following:
1. Data Entry – Enter eight 4-bit numbers using a dipswitch (or BCD switch). Read the number when a button is pressed. Store the numbers in sequential memory locations, start at $D000. Display each number as it is entered.
2. Data Sort — Sort eight 4-bit numbers in memory. (Largest to smallest)
3. Data Sum – Compute the sum of eight 4-bit numbers in memory. Display result in BCD.
4. Combine Routines – Combine the functions of 1, 2 and 3 into one program. Use 2 bits of PortA to specify which function to perform. 00=Data Entry, 01=Data Sort, 10=Data Sum, 11=Exit Program

Design Rules:
1. Use proper documentation when creating your source code.
M68HC11EVB I/O limitations: PortB, PortE, PortA (excluding PA7 and PA3)

Remember for Prelab: Show schematic diagrams for all external hardware designs.
Open your hyperterminal connection to the EVB.
Power up your EVB and press the EVB RESET button.
Note the contents of the screen:
BUFFALO-stands for “Bit User Fast Friendly Aid to Logical Operations”
BUFFALO 2.5 (ext) - Bit User Fast Friendly Aid to Logical Operation.
Press Enter. This brings up the BUFFALO prompt > You are ready to use the 68HC11EVB.
ASM == on-board assembler
ASM *address*

Displays the assembly language for specified address. User may change the instruction and/or data. Labels may not be used. Instead, use the actual memory location where the label points to. Use your .LST file for help.

Use <CR> to advance to next line of code. Use <CTRL-C> to abort on-board assembly.
BUFFALO 2.5 (ext) - Bit User Fast Friendly Aid to Logical Operation

>asm c000

C000 SEI
>clra
4F
C001 SEI
>ldab d001
F6 D0 01
C004 SEI
>clr d002
7F D0 02
C007 SEI
>clr d003
7F D0 03
C00A ROLA
>cmpb #00
Cl 00
C00C ROLA
>
G Go / Execute command

G address

Begins execution of a program at the specific address. The program must provide a jump instruction to get back to the BUFFALO prompt. (jmp $e00a)

$E00A is the location of BUFFALO in memory.

LOAD T

Download an assembled .S19 file from a PC

After entering the LOAD T command, go to the Transfer menu:Send Text File. Choose the .S19 file you wish to load. When the file is loaded the word done appears on the screen. You can now verify the file was loaded by using the Memory Display command (MD) and then the G command to run your program.
BUFFALO 2.5 (ext) - Bit User Fast Friendly Aid to Logical Operation
>load t

done
> -
MD Memory Display: Display memory to terminal screen

**MD address_start [address_stop]**

Displays 16 bytes per line. Display will begin on an even 16 byte memory boundary. If no address_stop is given, 9 lines will be displayed. If address_stop is less than address_start, one line will be displayed.
BUFFALO 2.5 (ext) - Bit User Fast Friendly Aid to Logical Operation

>load t

done

>md c000

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>C000</td>
<td>4F B7 10 04 F6 10 0A C4 01 26 F9 F6 10 0A C4 01 0</td>
</tr>
<tr>
<td>C010</td>
<td>27 F9 8B 01 19 B7 10 04 7E C0 04 B6 B6 B6 B6 B6</td>
</tr>
<tr>
<td>C020</td>
<td>0F 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49</td>
</tr>
<tr>
<td>C030</td>
<td>F0 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6</td>
</tr>
<tr>
<td>C040</td>
<td>0F 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49</td>
</tr>
<tr>
<td>C050</td>
<td>F0 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6</td>
</tr>
<tr>
<td>C060</td>
<td>0F 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49</td>
</tr>
<tr>
<td>C070</td>
<td>F0 F0 F0 F0 F0 F0 F0 F0 F0 B6 B6 B6 B6 B6 B6</td>
</tr>
<tr>
<td>C080</td>
<td>0F 0F 0F 0F 0F 0F 0F 0F 0F 49 49 49 49 49 49 49</td>
</tr>
</tbody>
</table>

>
MM Memory modify: Display and modify memory contents. **MM address**
Displays memory and gives the user a chance to modify it. Use the <SPACE> key to advance to the next byte. Use CTRL-H to backup one byte. Use <CR> to return to the BUFFALO prompt.
>mm d000
D000 0F 04
>mm d000
D000 04 55 0F 03 0F 00 0F 00
>md d000 0
D000 55 03 00 00 0F 0F 0F 0F 49 49 49 49 49 49 49 49 U
>
Connected 0:15:33 Auto detect 2400 8-N-1 SCROLL CAPS NUM Capture Print echo
RM Register Modify: Display and modify 68HC11 registers.

\textbf{RM} \[p,y,x,a,b,c,s\]

Displays the contents of the 68HC11’s registers. Also gives the user the chance to modify them.

Registers include:
- \texttt{P} – program counter
- \texttt{Y} – index register \texttt{Y}
- \texttt{X} – index register \texttt{X}
- \texttt{A} – Accumulator \texttt{A}
- \texttt{B} – Accumulator \texttt{B}
- \texttt{C} – Condition code register
- \texttt{S} – Stack pointer
> rm p
P-C000 Y-FFFF X-FFFF A-FF B-FF C-DO S-004A
P-C000 c000

> rm a
P-C000 Y-FFFF X-FFFF A-FF B-FF C-DO S-004A
A-FF 00

> rm
P-C000 Y-FFFF X-FFFF A-00 B-FF C-DO S-004A
P-C000
Y-FFFF 0000
X-FFFF 0000
A-00 00
B-FF 00
C-DO
S-004A

> rm
P-C000 Y-0000 X-0000 A-00 B-00 C-DO S-004A
P-C000
>
T Trace Instructions
T [n]
The trace instruction allows the user to execute a program n instructions at a time. The user must set the program counter to the correct starting address before using the trace command. The machine code for the instruction will be displayed along with the registers after each instruction.