

LA TROBE UNIVERSITY
DEPARTMENT OF ELECTRONIC ENGINEERING
ELECTRONICS-2

Microprocessors: Lab 2 - Assembly Language Programming

Aims:

- Study the MC681HC11's
 - addressing modes,
 - registers,
 - on chip Analog to Digital Converter
 - Reset vector and interrupt vectors.
- Learn to make BUFFALO "System Calls".

Equipment:

HC-COM, power supply, video terminal, Motorola HC11 Manual.

Introduction:

The instructions that a processor executes can access memory in many different ways. These different ways are known as the **addressing modes** of a processor. The MC68HC11 processor supports six addressing modes. These are *Inherent, Immediate, Direct, Extended, Indexed and Relative* addressing modes. These different addressing modes facilitates efficient programming.

The operating system of a computer, e.g. **BUFFALO**, manages the computer's resources and provides a set of software tools to make the programmer's life a bit easier. Such routines might deal with printing to the screen, reading from a keyboard, saving data to disk, etc. The way to access these routines is by a simple jump to subroutine (JSR) to the appropriate place within BUFFALO.

Note: All addresses, data and opcodes are in HEX.

Procedures:

Answer all questions, perform all the exercises and prepare your report in your lab book as you proceed. You may refer to lecture notes, hand-outs and other references. Refer to Lab 1 for instructions on how to use Hyperterminal.

Q1. The operation performed with the instruction **LDAA #FF** which loads the accumulator A with an immediate value FF is called *Immediate Addressing*. The value is always prefixed with #.

A) What will happen if you forgot the #, and typed in LDAA FF instead ?
It is called *Direct Addressing* . Try it, compare the opcodes and record you findings.

B) What is the signed 8-bit integer value of the unsigned numbers \$FF and \$FE?

Q2. Run the following program, write down all the opcodes and record the response from BUFFALO:

```
;Remember type A 2000 to commence assembling starting at $2000
LDAA #FF ; load accumulator A with the value FF
TAB ; transfer the contents of accumulator A to accumulator B
INCA ; increment the contents of accumulator A by 1
SWI ; return to BUFFALO.
; Type Control-C to exit the inline assembler.
; Type G 2000 to run the program just assembled at mem location $2000
```

In the above program, the instructions TAB, INCA and SWI **do not involve any memory locations at all**. The mode of operation is called *Inherent Addressing*. *The data that is manipulated is held in the registers internal to the Microcontroller. Write down the resulting register values after execution.*

Q3. Operations that load from or store to a location in memory use a form of addressing known as *Extended Addressing*. Run the following program:

```
LDAA FFFE ; load accumulator A from address FFFE.
LDAB FFFF ; load accumulator B from address FFFF.
SWI ; return to BUFFALO.
```

A) What values do you now have in accumulator A and accumulator B ?

B) What is the signed 16-bit integer value of the unsigned numbers \$FFFF and \$FFFE?

Q4. Concatenate the values in Accumulators A and B to form an address (in HEX of course). Start running the program at that address.

A) What is the result ?

B) Explain your findings.

DEBUGGING USING TRACE

Q5. In a long program, making one or two subtle mistakes is inevitable. The BUFFALO command "*T n*" allows us to **trace** through the next *n* instructions, to make sure that the program does exactly what was intended. We shall use this trace command to follow the execution of the following counting program, in which 1 is repeatedly added to the contents of A until a final value of 06 is reached.

Assemble the following code using BUFFALO at location \$2000 - enter: **A 2000**

```
LDAA #00 ; load accumulator A with value 00,
here: ; Define label "here" - take note of this address
INCA ; increment the contents of A by 1,
CMPA #06 ; compare the contents of A with the value 06,
BNE here ; repeat if A is not equal to 06,
SWI ; return to BUFFALO.
```

Note: The instruction **BNE** stands for "**branch if not equal to**".

Also note that "here:" is a label. Unlike AS11, you do not type the label into you BUFFALO assembler. Instead, you must take note of the memory address to which "**here**" corresponds. Then when you are typing in "**BNE ...**", use that address in place of the "**here**" label.

5A) After you have typed in the **BNE** line, what did you notice ?

B) Did BUFFALO use the address you typed, or did it replace this with something else ? Explain your results.

The "**BNE <address>**" line of codes is an example of *Relative Addressing* mode.

C) In what way is it different from all previous addressing modes ?

Set the program counter to **2000**. To do this type: **R P <enter> 2000 <enter>**. Then type **T** repeatedly until after the **SWI** line is executed.

D) Record all the responses from BUFFALO. Explain your results.

Q6. It is often useful to employ a register to point to a particular location in memory. This is done by loading a special register, known as an index register, with an address. Instructions can then access that location by using the index register to point to that location. Then to reference the next location, it is just a matter of incrementing the index register. The following example illustrates the techniques of *Indexed Addressing*. Here, the X register is used to hold the address, and the load operations reference this index.

```
LDX    #FFF0        ; Load the index register X with value FFF0
LDAB   0, X         ; Load accumulator B with the contents of the address
                          ; pointed to by the X register with zero displacement
INX    ; Increment X register to point to the next location
LDAA   0, X         ; Load accumulator A with the location of an address
                          ; with a zero displacement relative to the address
                          ; pointed to by the X register
SWI    ; Return to BUFFALO
```

A) At what addresses are the two load instructions directed ?

B) What were the contents of the A and B accumulators after the program ran ?

C) At what address would LDAA 3,X load from, if X was FFF1 ?

THE STACK

Q7. The stack is an area of RAM used for temporary storage both by programs and the processor. The stack pointer register (SP, sometimes simply S) points to the current location on the stack available for storage. Values are placed on the stack from a register by the PSH (push) instruction and retrieved from the stack into a register by the PUL (pull) instruction. **Enter in the following routine starting at address 2000.**

```
LDS    #7FFF        ; Load stack pointer register with $7FFF
LDAA   #51          ; Load accumulator A with value $51.
PSHA   ; Store contents of A on stack
LDAA   #61          ; Load accumulator A with new value $61.
PSHA   ; Store contents of A on stack.
LDAA   #FF          ; Overwrite contents of Accumulator A
TAB    ; copy Accumulator A to Accumulator B
SWI    ; Call BUFFALO monitor
```

Set the program counter to 2000 and trace the operation of the program.

- A) Take notes of the contents of all the registers involved at each step.
- B) Examine memory locations \$7FFF and \$7FFE and report their contents.

In the above code REPLACE THE **SWI** instruction with the following code:

```
PULB          ; Retrieve the upper most contents of stack and
              ; store it in accumulator B
PULA          ; Retrieve the next contents of stack and
              ; store it in accumulator A.
SWI           ; Return to BUFFALO.
```

Run the following routine starting at address 2000.

- C) What are the final contents of accumulators **A** & **B**?

Describe in detail the stack operations in terms of memory locations at \$7FFF, including a sketch of the changed memory and its contents.

Condition Code Register

Q8. The **Condition Code Register (CCR)**, register “**C**” in Buffalo) is a special register that keeps track of the status of the processor, such as whether the result of the last ALU operation was zero, or negative, or if the processor is handling an interrupt.

The Motorola M68HC11 Reference Manual, Appendix A describes the operation of all instructions, and each instruction’s effect on the CCR.

The CCR Bits are named: S X H I N Z V C

“•” or “-” means the bit is **not changed**,

“0” means the bit is **cleared** (cleared to 0),

“1” means the bit is **set** (set to 1),

“↕” means the bit is **changed** according to the Boolean Formulae.

“↓” means the bit **may change** From 1->0, 0->0, 1->1 but not 0->1 (see instruction RTI).

- A) What does each of the bits in the CCR represent?
- B) How would the CCR be affected if FF and FF were added together?
- C) Write a program that checks this.

BUFFALO System Calls

- Q9.** This next exercise is about making System Calls. For instance, to input a character from the terminal's keyboard and then print this to the terminal's screen, all that is required is a **JSR** to the subroutine **INCHAR** at address **FFCD**, followed by a **JSR** to the subroutine **OUTPUT** at address **FFAF**.

```
JSR    FFCD          ; Call INCHAR, return with char in A
JSR    FFAF          ; Call OUTPUT to print char.
SWI                                ; Return
```

A) Run this program at least twice, and type in a different letter each time. Record what you find, and pay particular attention to the contents of accumulator A.

- Q10.** Use these routines, write an assembly language program that reads characters from the keyboard (until a return, with ASCII code "0D", is typed) and prints these to the screen spread out with spaces between each character.

i.e. You type: hello world
The machine prints: hheelllloo wwoorrlldd

The 78HC11's Analog to Digital Converter

The Analog to Digital Converter (ADC) section in a 68HC11 has the following features:

It is based on the charge re-distribution technique

It has an internal charge pump to enable the analog input multiplexor transistors to switch to low impedance

It can automatically perform conversion of 4 input channels in succession and it does not require a sample-and-hold circuit because the input voltage is applied to an internal capacitor

- Q11.** Assemble the attached code to display the ADC value on the LED displays, then download it to HCCOM. **Run the code and attach a BNC-to-probe lead to input CHAN1.** Also you will need to connect a -15V supply to the blue power supply lead..

Note: You may need to disconnect the -15V lead after pressing reset the CPU to avoid latching up the CPU (because negative voltages may be present on the ADC inputs).

Probe around the +15V, +5V, 0V and -15V power supply connector and write down the HEX values measured for these voltages. Convert the HEX values to decimal.

Note Switch SW2 should be in the UP position, JP5 jumper on 5V, and Dip SW1-2 On (all others off).

```
* 68HC11Ax Analog to digital converter software
* Assemble with the command:
* as11 adc.asm -l > asm.lst
*
REGS    EQU    $1000          * 68HC11 Configuration Registers
TOF     EQU    %10000000     * Timer overflow flag
ADSET   EQU    %00000100     * A/D input on PE-4
CCF     EQU    %10000000     * Conversion complete flag
ADPU    EQU    %10000000     * A/D power-up bit
*
N1      EQU    300
```

```

TFLG2 EQU    $25
OPTION EQU    $39
ADCTL EQU    $30
ADR1 EQU     $31
*
* Monitor Equates
* Print left half
OUTLHF EQU    $FFB2
OUTRHF EQU    $FFB5 * Print right half
CRLF EQU     $FFC4 * Print Carriage Return/Line Feed (CRLF)
*
* Memory Map Equates
CODE EQU     $2000
DATA EQU     $4000
STACK EQU    $0040
*
*
*      ORG CODE
*      LDX    #REGS

* Power up the A/D converter charge pump
*      BSET   OPTION,x ADPU
*
*      JSR    delay100us
*
* Start the conversion SCAN=0, MULT=0
FOREVER
* Beginning of LOOP-FOREVER
*      LDAA   #ADSET * Select ADC input from Port E-4
*      STAA  ADCTL,x * Signal ADC to start conversion
*
* Wait until conversion done
convwait:
*      BRCLR  ADCTL,x CCF convwait
*
* Get the input and print it using the BUFFALO Monitor
*      LDAA  ADR1, x * Get ADC converted input voltage value
*      TAB   * save it (A) in B
*
*      JSR   OUTLHF * output high nybble (Left Half)
*      TBA   * restore it to A
*      JSR   OUTRHF * output low nybble (Right Half)
*      JSR   CRLF * output carriage return - line feed
*
* Delay approx 1 second using the timer overflow
*      LDAB  #N1
* Clear the TOF to start the delay
delay1:
*      ldaa  #TOF
*      staa  TFLG2,x
* and wait for N1 overflows
*
spin1:
*      TST   TFLG2,x
*      BPL   spin1
*
*      DECB
*      BNE  delay1
*      BRA  FOREVER * loop forever

* -----
* This subroutine delays (a little over) 100 micro-seconds
* -----
delay100us:
*      psha * save Accumulator A
*      * Generate a "short" delay > 100 microsec
*      LDAA #40 * 40 loops for 200 clock cycles at 0.5 micro seconds/clock

DELAYLOOP:
*      DECA *
*      BNE  DELAYLOOP
*      PULA * restore Accumulator A
*      RTS *

```

Table 1: 68HC11 - Interrupt and Reset Vectors
 Derived from Table 9-3 of 11A8TD.pdf

Vector Address	Interrupt Source	Condition Codes Register Mask	Local Mask
FFC0..FFD5	Reserved	-	
FFD6..FFD7	SCI Serial System	I Bit	See Table 9-4
FFD8..FFD9	SPI Serial Transfer Complete	I Bit	SPIE
FFDA..FFDB	Pulse Accumulator Input Edge	I Bit	PAII
FFDC..FFDD	Pulse Accumulator Overflow	I Bit	PAOVI
FFDE..FFDF	Timer Overflow	I Bit	TOI
FFE0..FFE1	Timer Output Compare 5	I Bit	OC51
FFE2..FFE3	Timer Output Compare 4	I Bit	OC41
FFE4..FFE5	Timer Output Compare 3	I Bit	OC31
FFE6..FFE7	Timer Output Compare 2	I Bit	OC21
FFE8..FFE9	Timer Output Compare 1	I Bit	OCi I
FFEA..FFEB	Timer Input Capture 3	I Bit	OC31
FFEC..FFED	Timer Input Capture 2	I Bit	OC21
FFEE..FFEF	Timer Input Capture 1	I Bit	OCi I
FFF0..FFF1	Real Time Interrupt	I Bit	RTII
FFF2..FFF3	IRQ (External Pin or Parallel I/O)	I Bit	See Table 9-2
FFF4..FFF5	XIRQ Pin (Pseudo Non-Maskable Interrupt)	X Bit	None
FFF6..FFF7	SWI Software Interrupt	None	None
FFF8..FFF9	Illegal Opcode Trap	None	None
FFFA..FFFB	COP Failure (Reset)	None	NOCOP
FFFC..FFFD	COP Clock Monitor Fail (Reset)	None	CME
FFFE..FFFF	RESET Vector - Power On Reset	None	None

Table 2: BUFFALO SOURCE EXCERPT: LOCATION OF HC-COM's I/O ROUTINES

```

ROMBS      EQU    $E000                start of EPROM
ORG        ROMBS+$1F7C
FF7C .WARMST JMP    MAIN                warm start
FF7F .BPCLR  JMP    BPCLR               clear breakpoint table
FF82 .RPRINT JMP    RPRINT              display user registers
FF85 .HEXBIN JMP    HEXBIN              convert ascii hex char to binary
FF88 .BUFFAR JMP    BUFFARG             build hex argument from buffer
FF8B .TERMAR JMP    TERMARG             read hex argument from terminal
FF8E .CHGBYT JMP    CHGBYT              modify memory at address in x
FF91 .READBU JMP    READBUFF            read character from buffer
FF94 .INCBUF JMP    INCBUFF             increment buffer pointer
FF97 .DECBUF JMP    DECBUFF             decrement buffer pointer
FF9A .WSKIP  JMP    WSKIP                find non-whitespace char in buffer
FF9D .CHKABR JMP    CHKABRT              check for abort from terminal
    
```

**Table 2 continued: BUFFALO SOURCE CODE EXCERPT:
LOCATION OF HC-COM's I/O ROUTINES**

```

                ORG      ROMBS+$1FA0
FFA0  .UPCASE  JMP      UPCASE      convert to upper case
FFA3  .WCHEK  JMP      WCHEK       check for white space
FFA6  .DCHEK  JMP      DCHEK       check for delimiter
FFA9  .INIT   JMP      INIT        initialize i/o device
FFAC  .INPUT  JMP      INPUT       low level input routine
FFAF  .OUTPUT JMP      OUTPUT      low level output routine
FFB2  .OUTLHL JMP      OUTLHLF     display top 4 bits as hex digit
FFB5  .OUTRHL JMP      OUTRHLF     display bottom 4 bits as hex digit
FFB8  .OUTA   JMP      OUTA        output ascii character in A
FFBB  .OUT1BY JMP      OUT1BYT     display the hex value of byte at X
FFBE  .OUT1BS JMP      OUT1BSP     out1byt followed by space
FFC1  .OUT2BS JMP      OUT2BSP     display 2 hex bytes at x and a space
FFC4  .OUTCRL JMP      OUTCRLF     carriage return, line feed to terminal
FFC7  .OUTSTR JMP      OUTSTRG     display string at X (term with $04)
FFCA  .OUTST0 JMP      OUTSTRG0    outstrg with no initial carr ret
FFCD  .INCHAR JMP      INCHAR      wait for and input a char from term
FFD0  .VECINT JMP      VECINIT     initialize RAM vector table

```

**BUFFALO SOURCE CODE EXCERPT:
HCCOM's Interrupt & Reset Vector Names & Addresses**

```

                ORG      ROMBS+$1FD6
*** Vectors ***
FFD6  VSCI    FDB      JSCI
FFD8  VSPI    FDB      JSPI
FFDA  VPAIE   FDB      JPAIE
FFDC  VPAO    FDB      JPAO
FFDE  VTOF    FDB      JTOF
FFE0  VTOC5   FDB      JTOC5
FFE2  VTOC4   FDB      JTOC4
FFE4  VTOC3   FDB      JTOC3
FFE6  VTOC2   FDB      JTOC2
FFE8  VTOC1   FDB      JTOC1
FFEA  VTIC3   FDB      JTIC3
FFEC  VTIC2   FDB      JTIC2
FFEE  VTIC1   FDB      JTIC1
FFF0  VRTI    FDB      JRTI
FFF2  VIRQ    FDB      JIRQ
FFF4  VXIRQ   FDB      JXIRQ
FFF6  VSWI    FDB      JSWI
FFF8  VILLOP  FDB      JILLOP
FFFA  VCOPI   FDB      JCOP
FFFC  VCLM    FDB      JCLM
FFFE  VRST    FDB      BUFFALO

```