

## ECE331 Handout 5: ASM Program Examples

---

### Simple Arithmetic ASM Program Examples (Chapter 2)

#### **Example 1**

*Write a program to add the numbers stored at memory locations \$800, \$801, and \$802, and store the sum at memory location \$900.*

**Solution:** This problem can be solved by the following steps:

**Step 1:** Load the contents of the memory location at \$800 into accumulator A.

**Step 2:** Add the contents of the memory location at \$801 into accumulator A.

**Step 3:** Add the contents of the memory location at \$802 into accumulator A.

**Step 4:** Store the contents of accumulator A at memory location \$900.

The assembly program is as follows:

```
org    $1000 ; starting address of the program
ldaa   $800 ; place the contents of the memory location $800 into A
adda   $801 ; add the contents of the memory location $801 into A
adda   $802 ; add the contents of the memory location $802 into A
staa   $900 ; store the sum at the memory location $900
end
```

#### **Example 2**

*Write a program to subtract the contents of the memory location at \$805 from the sum of the memory locations at \$800 and \$802, and store the result at the memory location \$900.*

**Solution:** The logic flow of this program is illustrated here. The assembly program is as follows:

```
org    $1000 ; starting address of the program
ldaa   $800 ; copy the contents of the memory location at $800 to A
adda   $802 ; add the contents of memory location at $802 to A
suba   $805 ; subtract the contents of memory location at $805 from A
staa   $900 ; store the contents of accumulator A to $805
end
```

#### **Example 3**

*Write a program to add two 16-bit numbers that are stored at \$800~\$801 and \$802~\$803, and store the sum at \$900~\$901.*

**Solution:** This program is very straightforward:

```
org    $1000
ldd    $800 ; place the 16-bit number at $800~$801 in D
addd   $802 ; add the 16-bit number at $802~$803 to D
std    $900 ; save the sum at $900~$901
end
```

#### **Example 4**

*Write a program to subtract five (5) from four memory locations at \$800, \$801, \$802, and \$803.*

**Solution:** In the 68HC12, a memory location cannot be the destination of an ADD or SUB instruction. Therefore, three steps must be followed to add or subtract a number to or from a memory location:

**Step 1:** Load the memory contents into an accumulator.

**Step 2:** Add (or subtract) the number to (from) the accumulator.

**Step 3:** Store the result at the specified memory location.

The program is as follows:

```

org    $1000
ldaa  $800 ; copy the contents of memory location $800 to A
suba  #5 ; subtract 5 from A
staa  $800 ; store the result back to memory location $800
ldaa  $801
suba  #5
staa  $801
ldaa  $802
suba  #5
staa  $802
ldaa  $803
suba  #5
staa  $803
end

```

---

## Assembly and Execution Example (Chapter 2)

The following program will add two numbers stored in memory and then store the resulting sum into memory. The ASM code, assembly output (.lst) and instruction execution are shown.

### Assembly Code

```

; begin program
org    $C200
ldaa  N1
adda  N2
staa  SUM
swi

; store data to memory and assign address labels
; data automatically placed at end of program
N1    fcb    $02    ;first number
N2    fcb    $29    ;second number
SUM   fcb    $00    ;placeholder for sum

```

### Program Function

Mnemonic	Operation	Action	Op-Code
LDAA	load accA from memory	$A \leftarrow M$	B6 hh ll
ADDA	add memory to A	$A \leftarrow A + M$	BB hh ll
STAA	store accA to memory	$M \leftarrow A$	7A hh ll

### Assembled Code (.lst file)

```

address  op-codes  ASM
          1 ; begin program
C200          2    org    $C200
C200 [03] B6C20A  3    ldaa  N1
C203 [03] BBC20B  4    adda  N2
C206 [03] 7AC20C  5    staa  SUM
C209 [09] 3F      6    swi
          7 ; store data to memory and assign address labels
C20A  02        8 N1 fcb    $02
C20B  29        9 N2 fcb    $29
C20C  00       10 SUM fcb    $00 ;placeholder for sum

```

### Symbol Table

```

N1      C20A
N2      C20B
SUM     C20C

```

Program Memory (after storing program to microcontroller memory)

Address	Value		Instruction/Function	Note
C200	B6	p r o g r a m	LDAA	Origin
C201	C2			
C202	0A			
C203	BB		ADDA	
C204	C2			
C205	0B			
C206	7A		STAA	
C207	C2			
C208	0C			
C209	3F		SWI	end program
C20A	02	d a t a		N1
C20B	29			N2
C20C	00			SUM

Execution of Program

1. Initial Values –CPU Registers and Data Memory

**CPU Registers**

PC	
C2	00
A	B
-	-

**Data Memory**

addr	value	label
C20A	02	N1
C20B	29	N2
C20C	00	SUM

2. After LDAA

PC advances to next instruction ( $PC \leftarrow PC+3$ ); Value at N1 loads into accA ( $A \leftarrow \$02$ )

**CPU Registers**

PC	
C2	03
A	B
02	-

**Data Memory**

addr	value	label
C20A	02	N1
C20B	29	N2
C20C	00	SUM

3. After ADDA

PC advances to next instruction ( $PC \leftarrow PC+3$ ); Sum of values at N1 and N2 are in accA ( $A \leftarrow \$2B$ )

**CPU Registers**

PC	
C2	06
A	B
2B	-

**Data Memory**

addr	value	label
C20A	02	N1
C20B	29	N2
C20C	00	SUM

4. After STAA

PC advances to next instruction ( $PC \leftarrow PC+3$ ); Value in accA stored to memory at SUM ( $SUM \leftarrow accA$ )

**CPU Registers**

PC	
C2	09
A	B
2B	-

**Data Memory**

addr	value	label
C20A	02	N1
C20B	29	N2
C20C	2B	SUM

5. Software interrupt; program stops

## Branches & Reading Assembled List File Example (Chapter 2)

The following list output (.lst) file shows memory addresses of program bytes, clock cycles for each instruction, and the ASM code. This code was written and compiled in the WinIDE Development Environment. Filename: *branch.asm*

Memory Address	#Clock Cycles	Op-Codes	Line#	Label	Instruction	Operand	Comment
			1				; ECE331 Example of Branches
			2				; program will copy a list of data at HERE to THERE
			3				; number of bytes to copy set by BYTES
0000			4	HERE	EQU	\$2000	
0000			5	THERE	EQU	\$2020	
0000			6	BYTES	EQU	\$06	
1000			7		ORG	\$1000	
1000	[01]	C600	8		LDAB	#\$00	;initialize item counter
1002	[02]	CE2000	9		LDX	#HERE	;initialize index reg as memory pointers
1005	[02]	CD2020	10		LDY	#THERE	
1008	[03]	A600	11	LOOP	LDAA	0,X	;using indexed addressing
100A	[02]	6A40	12		STAA	0,Y	
100C	[01]	52	13		INCB		;increment counter
100D	[01]	C106	14		CMPB	#BYTES	;check for end of list
100F	[03]	2704	15		BEQ	DONE	;if we are done
1011	[01]	08	16		INX		;increment index registers
1012	[01]	02	17		INY		
1013	[03]	20F3	18		BRA	LOOP	;continue at top
1015			19	DONE	SWI		;use SWI for any program that stops running
1016			20		END		
			21				; data storage
2000			22		ORG	HERE	
2000		10111213	23		FCB	\$10,\$11,\$12,\$13,\$14,\$15,\$16	
		141516					
2020			24		ORG	THERE	
2020			25		RMB	BYTES	;reserve locations to copy data

Symbol Table	
BYTES	0006
DONE	1015
HERE	2000
LOOP	1008
THERE	2020

### Can you answer the following questions?

- Where is the program stored in memory (what addresses)?
- What is the op-code for the ASM instruction INCB?
- What value is loaded into index in line 9?
- What address mode is used to store data to memory in line 12?
- What is the value of the relative\_address\_mode offset byte for BEQ in line 15? Forward or backward?
- What is the value of the relative\_address\_mode offset byte for BRA in line 18? Forward or backward?
- What does the program do? Where is the main loop (from what line to what line)?
- What is the purpose of line 14?
- How many times does the copy loop execute? Does the value \$16 get copied?
- Could you explain the purpose and operation of each line in this ASM code?

## Example Loop using FOR Looping Structure

Write a program to add an array of N 8-bit numbers and store the sum at memory location \$800-\$801. Use the *For i = n1 to n2 do* looping construct.

**Solution:** We will use variable *i* as the array index. This variable can also be used to keep track of the number of iterations remained to be performed. We will use a two-byte variable **sum** to hold the sum of array elements. The logic flow of the program is illustrated in Figure 2.9.

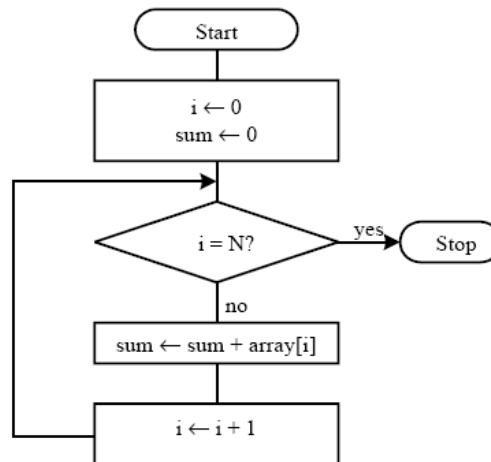


Figure 2.9 ■ Logic flow of example 2.14

The program is a direct translation of the flowchart shown in Figure 2.9.

```

N      equ      20                ; array count
      org      $800              ; starting address of on-chip SRAM
sum    rmb      2                ; array sum
i      rmb      1                ; array index
      org      $1000            ; starting address of the program
      ldaa     #0
      staa     i                ; initialize loop (array) index to 0
      staa     sum              ; initialize sum to 0
      staa     sum+1            ; "

loop   ldab     i
      cmpb    #N                ; is i = N?
      beq     done              ; if done, then branch
      ldx     #array            ; use index register X as a pointer to the array
      abx
      ldab    0,x               ; place array[i] in B
      dy      sum               ; place sum in Y
      aby
      sty     sum               ; compute sum <- sum + array[i]
      sty     sum               ; update sum
      inc     i                 ; increment the loop count by 1
      bra     loop

done   swi
      ; return to D-Bug12 monitor
; the array is defined in the following statement
array db      1,2,3,4,5,6,7, 8,9,10,11,12,13,14,15,16,17,18,19,20
end
  
```