
Structured Assembly Language Programming

Text: Chapter 8

TABLE 8-1 Assembly Language Program Elements

Program Element	Purpose
Program Header	Briefly describes the purpose of the program.
External Symbol Definitions	XREFs for symbols defined in some other source file.
Internal Symbol Definitions	XDEFs for symbols defined in this source file.
Assembler Equates	Definition of constants used in the program.
Code Section Start	Defines the following bytes to be in the code segment or section in ROM.
Program Initialization	Initializes the stack pointer, I/O devices, and other variables.
Main Program Body	This contains the main program.
Program End	Start the main program again or terminate in some way.
Program Subroutines	Subroutines and functions used in the main program.
Constant Data Section Start	Defines the following bytes to be in the constant data segment in ROM.
Constant Data Definitions	Definitions of constants in ROM.
Variable Data Section Start	Defines the following bytes to be variable data elements in RAM.
Variable Data Allocation	Allocation of space for variable data elements.

Example 8-1 The Completed Program

Metrowerks HC12-Assembler

(c) COPYRIGHT METROWERKS 1987-2003

```
Rel. Loc   Obj. code Source line
-----
1           ; HCS12 Assembler Example
2           ;
3           ; This program is to demonstrate a
4           ; readable programming style.
5           ; It initializes the A/D converter
6           ; and a bank of LEDs. It then reads the
7           ; value on the A/D, displays it, and delays
8           ; about 0.5 second. It then displays the
9           ; last value it converted for about 0.5
10          ; second and repeats.
11          ; Source File: M6812EX1_REL.ASM
12          ; Author: F. M. Cady
13          ; Created: 7/26/2004
14          ; Modifications: None
15          ;
16          ;*****
17          ; External symbol definitions
18          XREF get_AD, init_AD
19          XREF enable_LED, put_LED
20          XREF delay_X_ms
21          XREF __SEG_END_SSTACK
22          ;*****
23          ; Internal symbol definitions
24          XDEF Entry, main
25          ;*****
26          ; Constant Equates
27          0000 01F4 DELAY: EQU 500 ; Used for delay sub
28          ;*****
29          ; Code Section Start
30          MyCode: SECTION
31          Entry:
32          main:
33          ; Initialization section
34          ; Initialize stack pointer
```

```

35 000000 CFxx xx      lds  #__SEG_END_SSTACK
36                    ; Initialize all I/O devices
37 000003 16xx xx      jsr  init_AD    ; Init the A/D
38 000006 16xx xx      jsr  enable_LED ; Enable LED port
39                    ; Initialize the last data value
40 000009 79xx xx      clr  Last_Val
41                    ;*****
42                    ; Main process loop starts here:
43                    loop:
44                    ; Get value from A/D
45 00000C 16xx xx      jsr  get_AD
46 00000F 37          pshb  ; Save it
47                    ; Display on LEDs
48 000010 16xx xx      jsr  put_LED
49                    ; Delay about 0.5 second
50 000013 CE01 F4      ldx  #DELAY
51 000016 16xx xx      jsr  delay_X_ms
52                    ; Now display the last value
53 000019 F6xx xx      ldab Last_Val
54 00001C 16xx xx      jsr  put_LED
55 00001F 33          pulb  ; Get the value back
56 000020 7Bxx xx      stab Last_Val; Save it for next time
57                    ; Delay about 0.5 second
58 000023 FExx xx      ldx  Delay1
59 000026 16xx xx      jsr  delay_X_ms ; Delays # ms in X
60                    ; Do forever
61 000029 20E1        bra  loop
62                    ;*****
63                    ; Subroutines and functions
64                    ; (This relocatable assembler program does
65                    ; not have any subroutines in the main
66                    ; module. If you do, however, this is the
67                    ; place to put them.)
68                    ;*****
69                    ; Constant data area in ROM
70                    MyConst:SECTION
71 000000 01F4        Delay1: DC.W  DELAY
72                    ;*****
73                    ; Variable data area in RAM
74                    MyData: SECTION
75 000000            Last_Val: DS.B  1

```

Program Header

Program Element

Program Example

Program Header

```
; M68HC12 Assembler Example
;
; This program is to demonstrate a
; readable programming style.
; It counts the number of characters
; in a buffer and stores the result in
; a data location. It then prints
; the number of characters using
; D-Bug12 Monitor routines.
; Source File: M6812EX1.ASM
; Author: F. M. Cady
; Created: 5/15/97
; Modifications: None
```

Equates

Program Element

Program Example

System Equates.

```
; Monitor Equates
out2hex:EQU    $FE16    ; Output 2 hex nibbles
putchar:EQU    $FE04    ; Print a character
; I/O Ports
PORIH: EQU     $24      ; Port H address
PORTJ: EQU     $28      ; Port J address
```

Constant Equates

```
; Constant Equates
CR: EQU       $0d      ; CR code
LF: EQU       $0a      ; LF code
NULL: EQU     $00      ; End of ASCII string
NIL: EQU      0        ; Initial data value
```

Memory Map Equates

```
; Memory Map Equates
PROG: EQU     $4000    ; Locate the program
DATA: EQU     $6000    ; Variable data areas
STACK: EQU    $8000    ; Top of stack
```

Program Location

Program Element

Program Example

Program Code
Origination

ORG PROG ; Locate the program.

Program Initialization

Program Element

Program Example

Stack Pointer
Initialization

```
lds    #STACK ; Initialize stack pointer
```

Variable Data
Initialization

```
; Initialize the data area to zero  
clr    Counter
```

Main Program Body

Program Element

Program Example

Main program body

```
; Count the characters in the string
    ldd    #STRING
    jsr    count_en
; Output the result string
    ldd    #RESULT ; Point to the string
    jsr    outstr
; Output the counter
    ldab   counter
    jsr    [out2hex,PCR]
; Now output a CRLF
    jsr    CRLF
```

Program End

Program Element

Return to the
monitor

Program Example

`swi`

`; Return to the monitor`

Program Subroutines

TABLE 8-3 Subroutine or Function Header

```
;      Subroutine calling sequence or invocation.  
;  
;      Subroutine name.  
;  
;      Purpose of subroutine.  
;  
;      Name of file containing the source.  
;  
;      Author.  
;  
;      Date of creation or release.  
;  
;      Input and output variables.  
;  
;      Registers modified.  
;  
;      Global data elements modified.  
;  
;      Local data elements modified.  
;  
;      Brief description of the algorithm.  
;  
;      Functions or subroutines called.
```

Program Subroutines

Program Element

Subroutines and functions

Program Example

```
; Subroutine "count_em( char *StringP )"
; This routine counts the characters in a string
; until the NULL character is found.
; Entry:    D register pointing to the start
; Exit:     None
; Reg Mod:  CCR
; Data Mod: Data location Counter contains the
;           number of characters
;
count_em
; Save the registers
    psbx
    tfr    d,x    ; Put the address in X
; WHILE the char is not ZERO
while_do:
    tst    0,x
    beq    found_eot
; DO count the chars
    inc    Counter
    inx
    bra    while_do

; ENDO
found_eot:
; ENDWHILE the char is not NULL
; Restore the registers
    pulx
    rts
; END subroutine "count_em( )"
```

Constant Data Definitions

Program Element

Main program constants and strings.

Program Example

```
; Constant data area in ROM  
STRING: DB    "This is a string",NULL  
RESULT: DB   "The number of characters is $",NULL
```

Variable Data Allocation

Program Element

Variable data area origination.

Program Example

```
; Variable data area in RAM  
ORG DATA
```

Example 8-2 Include Files

```
1 ; include.asm
2 ; Demonstration of the use of include files
3 ;
0000 4 include "y:\hc12\casm\include1.h"
5 ; This is the include1.h file
6 ; A useful include file is one which defines
7 ; commonly used constants
0000 8 EOT: EQU 04 ; End of transmission
0000 9 CR: EQU $0d ; Carriage return
0000 10 LF: EQU $0a ; Line feed
0000 11 include "y:\hc12\casm\include2.asm"
12 ; include2.asm
13 ; This is the second include file
14 ; You can have source code and macro
15 ; definitions.
```

Example 8-2 (cont)

```

                                16 ; Here is a macro definition
0000                            17 MU100: EQU !199 ; Delay loop counter
0000                            18 $MACRO Delay_100
                                19 ; Macro to delay 100 microseconds for the A/D start up
                                20 psha ; save the A reg
                                21 ldaa #MU100
                                22 loop: deca
                                23 bne \@loop
                                24 pula ; restore the A reg
0000                            25 $MACROEND
                                26
```

Example 8-3 Assembly Language for a Sequence Block

Example 8-3 Assembly Language for Sequence Block

```
; *****  
; DO_A  
; Comments describing the function of this sequence block  
. . . . (Assembly language code to do the function)  
; ENDO_A  
; *****
```

Decision Element Assembly Language Program (Example 8-4)

Pseudo-code design:

Get Temperature

IF Temperature > Allowed Maximum

 THEN

 Turn the water valve off

 ELSE

 Turn the water valve on

ENDIF Temperature > Allowed Maximum

IF-THEN-ELSE Assembly Lang. (Example 8-4)

```
AD_PORT:      EQU  $91          ; A/D Data Port
MAX_TEMP:     EQU  128         ; Maximum temperature
VALVE_OFF:    EQU  0           ; Bits for valve off
VALVE_ON:     EQU  1           ; Bits for valve on
VALVE_PORT:   EQU  $258        ; Port P for the valve
```

...

```
; Get the temperature
    ldaa AD_PORT
; IF Temperature > Allowed Maximum
    cmpa #MAX_TEMP
    bls ELSE_PART
; THEN Turn the water valve off
    ldaa VALVE_OFF
    staa VALVE_PORT
    bra END_IF
; ELSE Turn the water valve on
ELSE_PART:
    ldaa VALVE_ON
    staa VALVE_PORT
END_IF:
; END IF temperature > Allowed Maximum
```

Example 8-5

For each of the logic statements, give the appropriate M68HC12 code to set the condition code register and to branch to the **ELSE part** of an IF-THEN-ELSE. Assume P and Q are 8-bit, signed numbers in memory locations P and Q.

- A. IF P \geq Q
- B. IF Q > P
- C. IF P = Q

```
A.      ; IF P  $\geq$  Q
        ldaa    P
        cmpa    Q
        blt     ELSE_PART

B.      ; IF Q > P
        ldaa    Q
        cmpa    P
        ble     ELSE_PART

C.      ; IF P = Q
        ldaa    P
        cmpa    Q
        bne     ELSE_PART
```

Example 8-6

For each of the logic statements, give the appropriate M68HC12 code to set the condition code register and to branch to the **THEN part** of an IF-THEN-ELSE. Assume P and Q are 8-bit, signed numbers in memory locations P and Q.

- A. IF P \geq Q
- B. IF Q > P
- C. IF P = Q

```
A.      ; IF P  $\geq$  Q
        ldaa    P
        cmpa    Q
        bge     THEN_PART

B.      ; IF Q > P
        ldaa    Q
        cmpa    P
        bgt     THEN_PART

C.      ; IF P = Q
        ldaa    P
        cmpa    Q
        beq     THEN_PART
```

Assembly Code for a WHILE-DO

Example 8-7

Pseudocode design.

```
Get the temperature from the A/D
WHILE the temperature > maximum allowed
    DO
        Flash light 0.5 sec on, 0.5 sec off
        Get the temperature from the A/D
    END_DO
END_WHILE the temperature > maximum allowed
```

```

AD_PORT: EQU $91 ; A/D Data port
MAX_ALLOWED: EQU 128 ; Maximum Temp
LIGHT_ON: EQU 1
LIGHT_OFF: EQU 0
LIGHT_PORT: EQU $258 ; Port P
; ---
; Get the temperature from the A/D
    ldaa AD_PORT
; WHILE the temperature > maximum allowed
WHILE_START:
    cmpa MAX_ALLOWED
    bls END_WHILE
; DO - Flash light 0.5 sec on, 0.5 sec off
    ldaa LIGHT_ON
    staa LIGHT_PORT ; Turn the light
    jsr delay ; 0.5 sec delay
    ldaa LIGHT_OFF
    staa LIGHT_PORT ; Turn the light off
    jsr delay
; End flashing the light, Get the temperature from the A/D
    ldaa AD_PORT
; END_DO
    bra WHILE_START
END_WHILE:

```

Assembly Code for a WHILE-DO

(Example 8-7 cont)

```

                                17  ; DO
                                18  ; Flash light 0.5 sec on, 0.5 sec off
0006 9601                       19      ldaa    LIGHT_ON
0008 5A24                       20      staa    LIGHT_PORT      ; Turn the light
000A 160018                     21      jsr     delay          ; 0.5 sec delay
000D 9600                       22      ldaa    LIGHT_OFF
000F 5A24                       23      staa    LIGHT_PORT      ; Turn the light off
0011 160018                     24      jsr     delay
                                25  ; End flashing the light
                                26  ; Get the temperature from the A/D
0014 9670                       27      ldaa    AD_PORT
                                28  ; END_DO
0016 20EA                       29      bra     WHILE_START
                                30  END_WHILE:
                                31  ; END_WHILE the temperature > maximum allowed
                                32
                                33  ; Dummy subroutine
0018 3D                         34  delay:  rts
```

DO-WHILE Assembly Language

(Example 8-8)

Pseudocode design

DO

 Get data from the switches

 Output the value to the LEDs

ENDO

WHILE Any switch is set

DO-WHILE Assembly Language

(Example 8-8 cont)

```

1 ; 68HC12 Structured assembly code
2 ; DO-WHILE example
3 ; Equates needed for this example
0000 4 SW_PORT: EQU $28 ; The switches are on Port J
0000 5 LEDES: EQU $24 ; The LEDs are on Port H
6 ; - - -
7 ; DO
8 DO_BEGIN:
9 ; Get data from the switches
0000 9628 10 ldaa SW_PORT
11 ; Output the data to the LEDs
0002 5A24 12 staa LEDES
13 ; END_DO
14 ; WHILE Any switch is set
0004 F70028 15 tst SW_PORT
0007 26F7 16 bne DO_BEGIN
17 ; END_WHILE
```

DO-WHILE Example

One of the most common structures found in assembly language programming is a loop controlled by a counter. Show the pseudocode and the structured assembly code to do this.

Pseudocode design:

```
Initialize the counter for 26 repetitions
DO
    The things that need to be done
ENDDO
WHILE the counter is not equal to zero
```

Example

Structured assembly code:

```

1 ; dowhl2.asm
2 ; Equates needed for this example
0000 3 COUNT: EQU !26 ; Need to do loop 26 times
4
5 ; Initialize the counter for 26 repetitions
0000 C61A 6 ldab #COUNT
7 ; DO
8 DO_BEGIN:
9 ; The things that need to be done
10 ; - - -
11 ; END_DO
12 ; WHILE the counter is not equal to zero
0002 0431FD 13 dbne b,DO_BEGIN ; Decrement the counter
14 ; and branch if not zero
15 ; END_WHILE
```