AN1106
APPLICATION NOTE

TRANSLATING ASSEMBLY CODE FROM HC05 TO ST7

by Microcontroller Division Application Team

1 INTRODUCTION

This application note has been written to help users translate their HC05 assembly source code into ST7 source code.

Even if both assembly languages are quite similar, the philosophy and program structure are quite different.

A software translator (“migr2st7”) has been developed by STMicroelectronics and is available on the MCU ON CD.

For more information on the ST7 Assembly Tool Chain, please refer to the application note “Starting with ST7 Assembly Tool Chain” (AN988) and the software library available on Internet (www.st.com/product/support) or the ST7 CD ROM.

2 QUICK PROCEDURE

Follow these steps to use the conversion software:

1. Install the “migr2st7” software translator by double-clicking on migr2st7.exe (in the CD ROM)
2. Click on migr2st7.exe and enter the name and the path of the source file (HC05) and the target file (ST7). The input filename default extension is “.asm”.
3. Click on “Convert”: a message box titled “Finished-Trans” is displayed with the translation result “Translation 6805 to ST7 succeeded” or an error message.
5. Remember to rename all your.st7 translated sources files into .asm files before assembling with the ST7 assembler (you can directly call your translated files *.asm).
6. In order to use the HC05 to ST7 translator, you should have a source file with HC05 assembly code written for use with the 2500AD assembler. Otherwise, you will have to modify the software by hand. Refer to the ST7 assembler documentation and application notes for details using the ST7 instructions. See also Section 3.2 (RECOMMENDATIONS AND ADVICE).

3 DETAILED PROCEDURE

3.1 MIGRATING CODE

The first step in migrating HC05 code to ST7 is to use the code translator described in the previous section.

This free software has been designed to help convert HC05 assembly code into ST7 source code. Basically, it converts all mnemonics and it adapts some directives fit the ST7 assembler requirements.

This tool will not adapt the software routines for ST7 specific peripherals or use the specific features of the ST7 core (addressing modes, Y register...).

Therefore, further modification of the software have to be done by the user; refer to the ST7 assembler documentation and application notes for details on using of the ST7 instructions.

3.2 RECOMMENDATIONS AND ADVICE

The following gives some advice to help you in translate your code more quickly:

– Bear in mind that comment lines in ST7 assembly language begin with a semicolon ";".

– Remember to include the "**2500adcb.h**" file (this is a file included at the beginning of your source file during the translation). It contains definitions and macros for defining the Motorola code directives correctly if the assembler you use is not the 2500AD one. For example

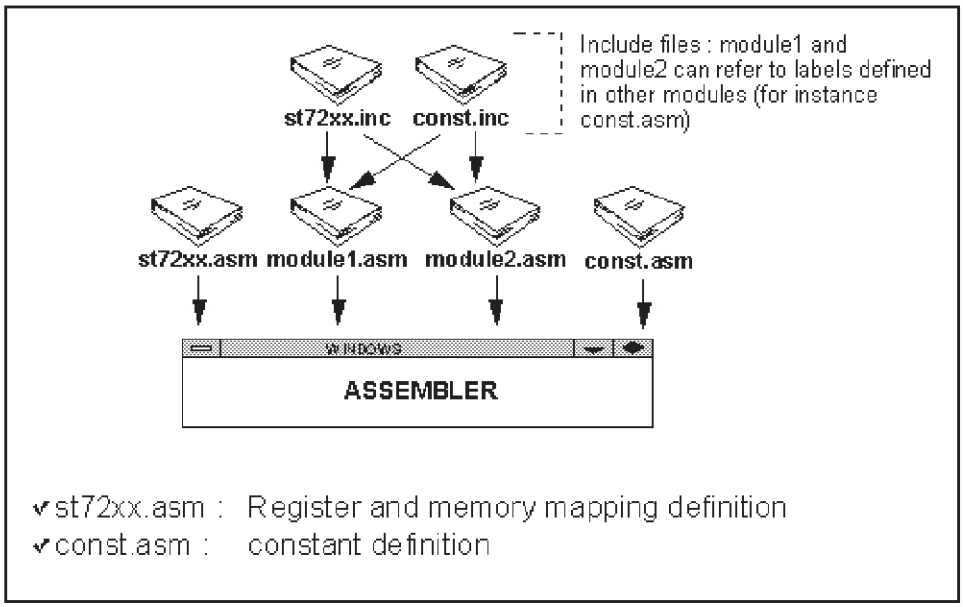
`#define EXPORT PUBLIC` or `#define IMPORT EXTERN.`

After using the translator, read all your files to make any corrections.

4 RECOMMENDED FILE STRUCTURE

The file structure described in the paragraphs below is the one recommended by ST.

The following diagram summarizes this structure:



This structure is composed of some source files (*.asm) and some include files (*.inc). It can be compared to a C structure with *.c and *.h files.

Remember to rename your include files as *.inc files after the code translation.

In the following ST7 code examples, some directives are shown followed by several possible Motorola directives in bold to allow the two codes to be easily compared.

4.1 VARIABLES

It's recommended to write all your variables in a file (called variable.asm for example) and to declare those you want to be public in a file called variable.inc as shown in the following example:

variable.asm:

```
st7/
                        TITLE "variable.asm"
      MOTOROLA                ; Motorola format (default), other formats are
also available:
                        ; Intel, Zilog or Texas.
      BYTES                   ; this directive means that the addresses of the
following
                  ; variables are bytes and not that the variables are bytes.
      segment 'ram0'; this directive means that the following variables are going
to be
      ; ORG RAM0     ; placed in ram0 from $00 to $FF .
      ; section .ram0      ; (segment byte 'ram0' at 00-FF : this line is in the map
file)
      var1          DS.B  1     ; 1 byte space is reserved in ram0 for var1 which
is local.
                    ; DEFS        ; corresponding MOTOROLA directive.
      .var2         DS.W  1     ; 1 word space is reserved in ram0 for var 2 which
is
                                ; public (a dot is placed just before the variable
name)
      WORDS                 ; this directive means that the addresses of the follow-
ing
                            ; variables are words.
      segment 'ram1'        ; this directive means that following variables are
going to be
      ; ORG RAM1             ; placed in RAM1 from $100 to $13F (for the ST72251)
      ; section .ram1
      .var3         DS.B  1     ; 1 byte space reserved in RAM1 for this public
variable.
                    ; DS or RMB
      segment 'rom'         ; this directive means that the following variables are
going to be
      ; ORG ROM              ; placed in ROM.
      ; section .rom
      .table        DC.B  $10,$AA,50,%11111010 ;table of values defined in ROM.
                    ; BYTE or DB or DEFB or FCB
      END
```

There are two ways to define a variable as public (which means visible from another module, the C equivalent directive is “extern”). The first one is the one used in the above example (the dot before the variable). The second one is to write the PUBLIC directive at the beginning of the variable.asm file followed by the name of the variables you want to be public.

variable.inc: (file to include in all .asm source files)

```
                                        TITLE “variable.inc”

      EXTERN            var2.b                                  ; the ad-
dress of var2 is a byte.
      EXTERN            var3.w; the address of var3 is a word.
      EXTERN            table.w; the address of table is a word.
```

In this .inc file, only the variables which have been defined as public in the corresponding .asm file have to appear. That’s the reason why var1 doesn’t appear (it’s a local variable).

Instead of writing a “.b” or “.w” after the variable to specify that its address is a byte, you can also write the BYTES or WORDS directives directly before the corresponding variables.

4.2 CONSTANTS

The best way to define all your constants is to declare them in a file (called constant.inc for example) with the directive “**#define constant_name value**”. Constants defined in this way don’t take any space in memory. All you have to do is include this .inc file in all your source files (in all files where you use the constants):

constant.inc:

```
      #define      const125    ; const1 = 25 (decimal)
      #define      const2 $1F  ; const2 = 1F (hexadecimal)
      #define      const3%11100001   ; const3=11100001 (binary)
........
```

You can define as many constants as you want. Just make sure that the “#” symbol is always the first character of the line but not in the first column.

4.3 LABELS

In order to be able to see all the labels from the emulator (in the disassembler window, a label name is more significant than an address), it's better to declare them all as public (a simple dot placed before the label name).

For labels which are called from another module, create a .inc file to declare them as "EXTERN":

appli.asm:

```
st7/
                    TITLE "appli.asm"
       MOTOROLA                    ; Motorola format (default), other formats are
also available:
                           ; Intel, Zilog or Texas.
       #include "appli.inc"
       ; lib appli.s05
       #include "variable.inc"
       #include "constant.inc"
       #include "ST72251.inc"; this is the map file (see below).

       WORDS
       segment 'rom'       ; the source code is written in the ROM segment.
       ; ORG ROM
.main
       LD            A,#$10
       LD            var2,A
.again DEC           var2
       JRNE          again

       CLR           X
.go    LD            A,(table,X)
       ADD           A,#1
       LD            (table,X),A
       INC           X
       CP            X,#4
       JRNE          go
........
.loop
       JRA           loop
; Don't forget to put the interrupt sub-routine library section here even if you
don't use any interrupts.
       END
```

4.4 MACROS

You can define your macros directly in the source file where they are used or if there are too many and/or if you use one macro in several source files, you can define them in a macro.inc file that you will have to include in all the source files (where the macros are used):

appli.asm:

Let's consider the same appli.asm source file given above. Let's imagine that instead of the missing source code in this file (....., see above) we call a macro:

```
MACRO1
```

Remember that a macro is a faster way than a functions for executing the same code several times. However program size is bigger when you use macros instead of subroutines; each time you invoke a macro to do a particular job, the whole macro assembly code is inserted into your source code. Macros are very useful when a small section of code must be used repeatedly.

The macro syntax is the following:

```
macro_name          MACRO [parameter_1] [,parameter_2...]
       [LOCAL label_name]
       [Body_of_the_macro]
       MEND
       ; ENDM
```

macro.inc:

```
.MACRO1MACRO
       LOCAL loop1
loop1  LD           A,$C000; A loaded with the value at the address $C000
       JRNE         loop1       ; loop until this value equals 0 (after an inter-
rupt for
                          ; example).
       MEND
```

The "LOCAL" directive avoids getting duplicate label errors for labels inside the macro (because the same macro can be called several times in the same module).

4.5 MAP FILE

Don't forget you also have to include the map file of the device you use, for example in your source files, you can have among the included files:

#include "ST72251.inc"

The corresponding ST72251.asm file will have to appear in the batch file.

5 THE BATCH FILE

Let's create the batch file corresponding to the example we have used in this note:

appli.bat:

```
del cbe.err          ; to have an error file cleaned at each batch.
asm -li st72251.asm
asm -li variable.asm
asm -li appli.asm
pause                      ; pause just before the link (possibility).
lyn st72251.obj+variable.obj+appli.obj,appli.cod;
asm st72251 -sym -fi=appli.map
asm variable -sym -fi=appli.map
asm appli -sym -fi=appli.map
obsend appli.cod,f,appli.s19,s;creation of the executable file (formatter).
```

Here is a list of the different options used in this batch file and their meaning:

* -li : enable the listing generation

* -sym: enable the symbol table generation

* -fi: updates the symbol table in the listing files