APPLICATION NOTE

EXECUTING CODE IN ST7 RAM

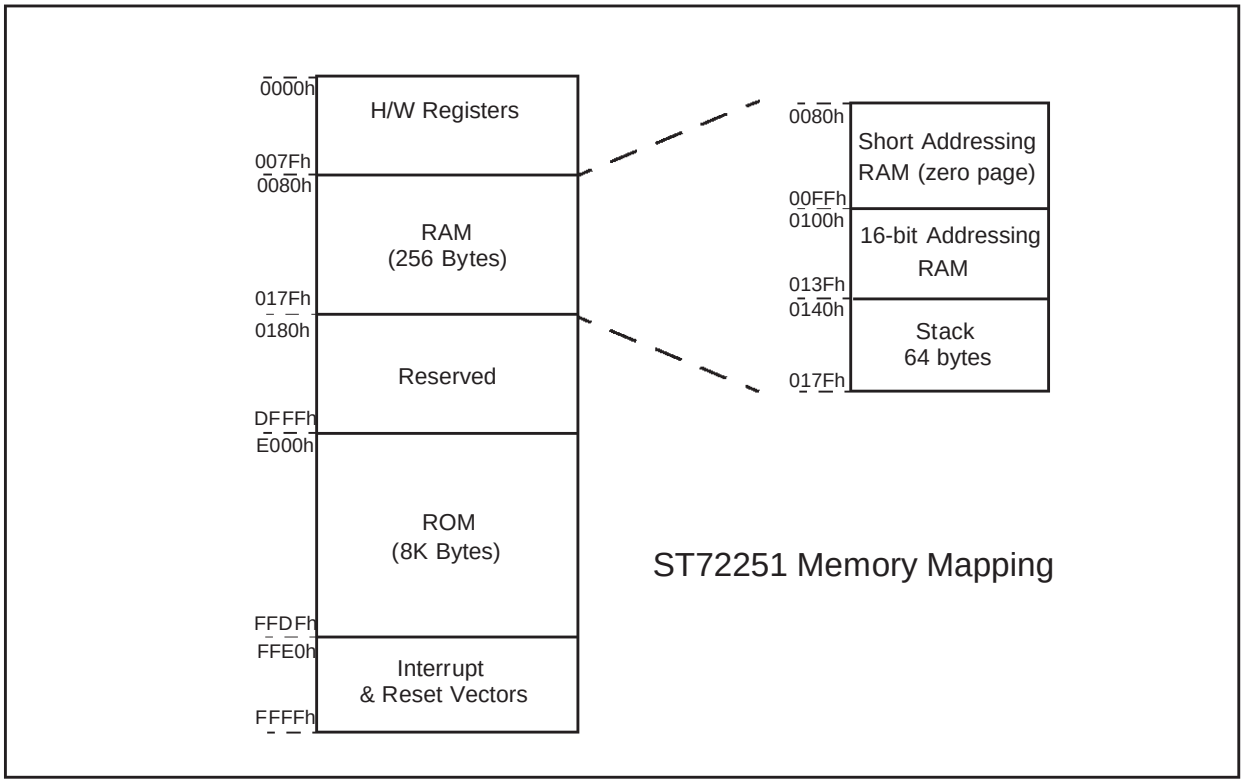
by 8-Bit Micro Application Team

1 INTRODUCTION

The purpose of this document is to give some guidelines about how to write and execute assembly code in RAM with the ST7 8-bit Microcontroller.

2 ST7 MEMORY MAPPING

The memory mapping in the ST72251 is shown in the following figure.



ST72251 Memory Mapping

The ST7 RAM and ROM locations are addressed and read in the same way. For the ST7 core fetching and decoding instructions located either in RAM or ROM, take the same time, there is no limitation executing code in RAM.

But the difficulties come from the fact that the RAM content is undetermined after power-up. So the RAM locations have to be filled with the code after reset. There are two ways to do that:

– The code is located in the ST7 ROM and is copied in RAM after reset.

– The code is loaded in the RAM through external communication.

The application does not need special instructions to execute code in RAM. But the management of the labels in RAM has to be done carefully.

The code has to follow the "BYTES" directive if it is located in the short address RAM named "zero page "(usually from 80h to FFh). All labels are treated into account as bytes.

If the code is located in RAM locations above the address 100h, it has to follow the "WORDS" directive.

3 CODE COPIED FROM ST7 ROM TO RAM

As described in the software example at the end of this document, we are using ST7 directives which allow you to store, in a ROM segment, the code to be used in RAM. The directives used in our example are:

```
segment byte at 100-13F 'ram1'
segment byte at EF00-FFDF 'romtocop'
segment 'ram1>romtocop'
```

The code written in the 'romtocop' segment is a copy of the 'ram1' segment. The volatile code is then stored in a non volatile location (the '`romtocop`' segment). So after each reset, the first instruction of the application must be translated from the '`romtocop`' segment to the '`ram1`' segment.

In the software example given in this application note, the two routines named "addition" and "subtraction" are located in the segment '`ram1>romtocop`'. After being copied in the 'ram1' segment the routines can be executed.

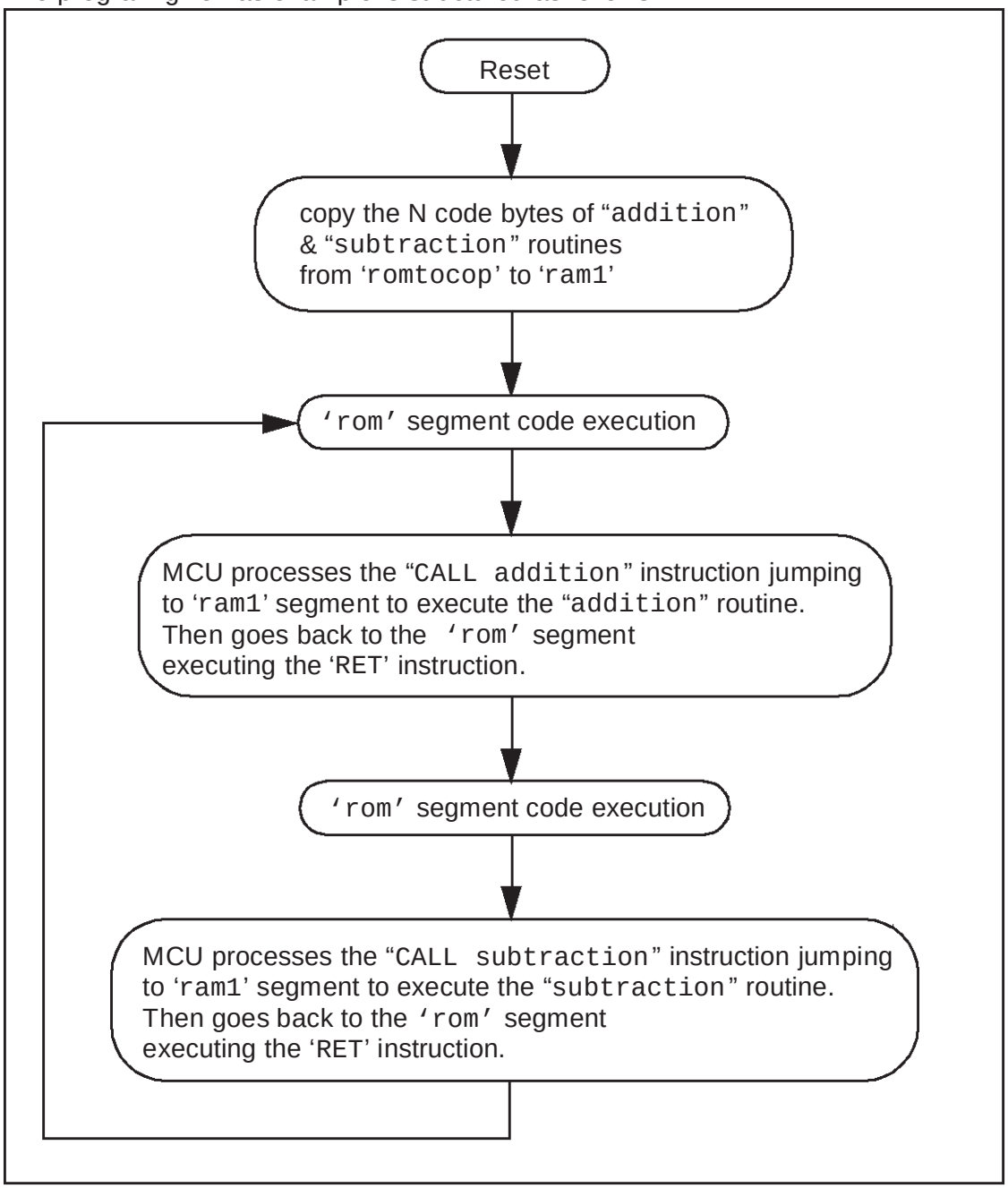
4 RAM CONTENT FILLED THROUGH EXTERNAL COMMUNICATION

The RAM content is filled with the code to execute using an external communication from a device to the ST7 through the ST7 IOs, SCI, SPI or I^2C peripherals. When the transfer is completed the code can be executed by loading the program counter with the first code address in RAM.

If the CALL instruction is used, the code located in RAM must contain the RET instruction to be able to go back to the code located in ROM.

5 FLOWCHART

The program given as example is structured as follows:



6 SOFTWARE

A software example is given below showing how to use code copied from ST7 ROM to RAM. The assembly code given is for guidance only. The file cannot be used on its own. The complete software environment can be found on the web.

```
st7/
;********************************** ************************************
; TITLE:              MODULE1.ASM
; AUTHOR:             PPG Microcontroller Applications Team
; DESCRIPTION:        Short Demonstration Program where code is executed
;                     in RAM after being copied from ROM to RAM.
;********************************** ************************************

    TITLE  "MODULE1.ASM"   ; this title will appear on each
                           ; page of the listing file
    MOTOROLA               ; this directive forces the Motorola
                           ; format for the assembly (default)
    #INCLUDE "st72251.inc"     ; include st72251 registers and memory mapping file
    #INCLUDE "constant.inc"    ; include general constants file


;********************************** **********************
;    Program code to copy in RAM
;********************************** **********************
    WORDS                  ; define subsequent addresses as words
                           ; meaning that all instructions are located
                           ; in the address area above 0FFh in the ST72251
                           ; memory mapping
    segment byte at 100-13F 'ram1'    ; these lines define the segment to
    segment byte at EF00-FFDF 'romtocop' ; copy from ROMTOCOP to RAM1
    segment 'ram1>romtocop'       ; the label values stored in the code located
                                        ; in the ROMTOCOP segment are defined
                           ; to be executed from the RAM1 segment

.addition                  ; when the routine is called d2 is already in A
    add   A,ramloc1        ; A = data2 + data1
    ld    X,A              ; Store (data1 + data2) in X register
    ret                    ; and return to main program in ROM
.subtraction               ; when the routine is called d2 is already in A
    sub   A,ramloc2        ; A= data1 - data2
    ld    X,A              ; Store (data1  - data2) in X register
.end_ram ret               ; and return to main program in ROM
```

```
segment 'rom'
;********************************* ****************************
; This instruction is one of the most important for the linker.
; It means that, until the next segment directive,
; all the code written below will be located in the rom part of the
; memory (start address E000h).
;********************************* ****************************
.main

.difference equ {end_ram-addition} ; this label contains the number of bytes to
                                   ; copy in RAM
       ld X,#difference.L   ; Load in X the number of bytes to copy
cont   ld  A,(romtocop,X) ; indexed addressing mode
       ld  (ram1,X),A    ; X bytes from  ROMTOCOP to RAM1
       dec  X
       jrpl  cont      ; the content of the ROMTOCOP segment is
                       ; copied to the RAM1 location to be executed
       clr   A         ;Its instruction's address is E000h.
loop   ld   A,#data1     ; Get data1 value
       ld   ramloc1,A    ; and put it in RAM0 segment
       ld   A,#data2     ; Do it again for data2
       ld   ramloc2,A
       call  addition     ; call the other module routine stored in RAM1
                       ; and execute it in this location
       ld   ramloc3,X    ; We stock the addition procedure's result
                                   ; in RAM0
       ld   A,#data2                       ; Get data2 value
       ld   ramloc2,A ; and put it in RAM0 segment
       ld   A,#data1     ; Do it again for data1
       ld   ramloc1,A
       call  subtraction ; call the other module routine stored in RAM1
                       ; and execute it in this location
       ld   ramloc3,X ; We stock the subtraction procedure's result
                       ; in RAM0

     jp loop          ; Do the loop once again

;    ****************************************
;    *                      *
;    * INTERRUPT SUB-ROUTINES LIBRARY SECTION *
;    *                      *
;    ****************************************

dummy  iret           ; Empty subroutine. Go back to main (iret instruction)

    segment 'vectit'
; ************************* ****************************************
```

```
; This last segment should always be present in your own programs.
; It defines the interrupt vector addresses and the interrupt routines' labels
; depending on the microcontroller you are using.
; Refer to the MCU's datasheet to see the number of interrupt vectors
; used and their addresses.
; Remember that this example is for a ST72251 based application.
; ************************* ****************************************
       DC.W  dummy      ;FFE0-FFE1h location
       DC.W  dummy      ;FFE2-FFE3h location
.i2c_it DC.W  dummy      ;FFE4-FFE5h location
       DC.W  dummy      ;FFE6-FFE7h location
       DC.W  dummy      ;FFE8-FFE9h location
       DC.W  dummy      ;FFEA-FFEBh location
       DC.W  dummy      ;FFEC-FFEDh location
.timb_it DC.W  dummy      ;FFEE-FFEFh location
       DC.W  dummy      ;FFF0-FFF1h location
.tima_it DC.W  dummy      ;FFF2-FFF3h location
.spi_it DC.W  dummy      ;FFF4-FFF5h location
       DC.W  dummy      ;FFF6-FFF7h location
.ext1_it DC.W  dummy      ;FFF8-FFF9h location
.ext0_it DC.W  dummy      ;FFFA-FFFBh location
.softit DC.W  dummy      ;FFFC-FFFDh location
.reset DC.W  main      ;FFFE-FFFFh location

    END
```

THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS.