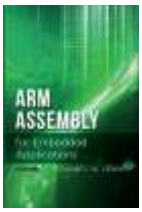


ARM Assembly for Embedded Applications, 2nd edition
ISBN 978-1-48359-041-7

CHAPTER 2: BINARY NUMBER SYSTEMS

Page 15, bottom of the page: Delete the incomplete sentence that begins, “To distinguish …”

CHAPTER 3: WRITING FUNCTIONS IN ASSEMBLY

Page 40, paragraph below Figure 3-10, 2nd sentence: Change the phrase "the LDR loads" to "the second loads".

Page 46: Add the following two sections:

3.7.6 Function PushButtonPressed

Function prototype:

```
int PushButtonPressed(void) ;
```

Returns 1 if the blue pushbutton is pressed, 0 if not.

3.7.7 Function GetRandomNumber

Function prototype:

```
uint32_t GetRandomNumber(void) ;
```

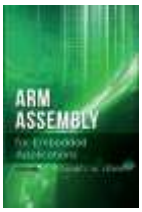
Returns a 32-bit value from the internal random number generator of the Cortex-M4F MCU.

CHAPTER 4: COPYING DATA

Section 4.6 (Addressing Modes), page 59, paragraph titled “PC-Relative addressing”:

The use of PC-Relative addressing with the STR instruction and its variants (STRB, STRH, STRD) has been deprecated, similar to the use of PC-Relative addressing with the VSTR instruction as described in Section 9.4.4. Although there are several examples such as `STR R0,x` found in chapter 4, they would actually be rejected by the assembler. To store into a memory location using its label requires a two-instruction sequence such as `ADR R1,x` or `LDR R1,=x` followed by `STR R0,[R1]`. The ADR instruction is only able to reference locations that are within 4095 bytes of the ADR, which would usually mean that the referenced location resides in flash memory (and therefore cannot be modified during execution). To use a label to reference data in read/write memory thus requires the `LDR R1,=x` instruction.

Note that since the text uses assembly to write small functions called from a C main program, there is rarely (if ever) a need for PC-Relative addressing because the operands of such functions are typically only the function parameters that are passed to the function in registers and the result is left in a register.



Page 52, end of 1st paragraph: ASCII constants are written with a single leading apostrophe, as in 'A. Note that this is different from how they are written in C (e.g., 'A'). C-style character constants also work ('a'), but not all escape sequences ('\0 or '\0') do. It's safer to use hex (0x00) instead.

Page 53: Add the following footnote: "The memory operand of LDRD must reside at a mod 4 (word aligned) address to avoid an address alignment fault".

Page 57: Add the following footnote: "The memory operand of STRD must reside at a mod 4 (word aligned) address to avoid an address alignment fault".

Page 68: Add the following footnote: "The memory operands of LDMIA, STMIA, LDMDB and STMDB must reside at a mod 4 (word aligned) address to avoid an address alignment fault".

Page 69, Listing 4-1: Add the following note just below the listing: "The dst and src parameters are assumed to hold word aligned addresses, or else an address fault will occur".

CHAPTER 5: INTEGER ARITHMETIC

Page 90, Table 5-5: Remove the entries for non-existent instructions UQADD and UQSUB.

CHAPTER 6: MAKING DECISIONS AND WRITING LOOPS

Page 100: Replace the last paragraph and subsequent code by the following:

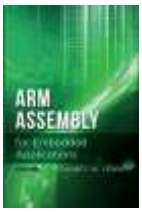
Since a compare is simply a subtraction that discards the difference but records the characteristics of that result in the flags, one might assume that all that is needed is to simply perform a 64-bit subtraction and check the resulting flags. However, although the `N`, `V` and `C` flags will be correct, the `Z` flag may not since it will only indicate if the most-significant half of the difference is zero.

When the condition to be tested does *not* depend on the `Z` flag (`GE,LT,HS,LO,MI,PL,VS,VC,AL`), then no correction is necessary and the condition test may immediately follow the 64-bit subtraction:

```
...                     // Load operands as before
SUBS      R0,R0,R2      // subtract LS halves, capture borrow
SBCS      R1,R1,R3      // subtract MS halves w/brw; set flags
...                     // OK to test GE,LT,HS,LO,MI,PL,VS,VC,AL
```

However, all other conditions (`EQ,NE,GT,LE,HI,LS`) require a different approach. The solution for `EQ` and `NE` is quite simple:

```
...                     // Load operands as before
SUBS      R0,R0,R2      // compute LS half of difference
SBC       R1,R1,R3      // compute MS half of difference
ORRS      R1,R0,R1      // Z=1 iff both halves are zero
...                     // Now OK to test for EQ or NE
```



For LE, GT, LS and HI we can avoid testing the Z flag by reversing the operands. Since x <= y is equivalent to y >= x, this allows us to replace LE by GE or LS by HS, which don't require testing the value of Z:

```
...                     // Load operands as before
SUBS      R2,R2,R0      // subtract LS halves (operands reversed)
SBCS      R3,R3,R1      // subtract MS halves (operands reversed)
...                     // Replace LE/GT by GE/LT, or
...                     // Replace LS/HI by HS/LO.
```

Finally, note that in all cases except EQ and NE, we can avoid modifying one register by replacing the SUBS instruction by a CMP instruction.

Page 102, bullet point 2 in the top middle of the page: Change "Only the last instructions…" to "Only the last instruction…" (singular form).

CHAPTER 7: MANIPULATING BITS

Page 120, Table 7-5: Replace second ORR by BIC and third ORR by EOR.

Page 129, Problem 4: Replace the first two sentences in the problem description by "Write a function in ARM assembly language that returns the number of bits in the parameter not including leading and trailing 0's. For example, if the parameter is 006203F016, the function should return the value 19."

CHAPTER 10: WORKING WITH FIXED-POINT REAL NUMBERS

Page 157, Table 9-8, last 2 rows: Change " VLMA" to "VMLA", and change " VLMS" to "VMLS".

Page 158, Listing 9-1: Change " VLMA" to "VMLA".

Page 176, bottom of the page: Inside the box, change the number on the second rule from "1" to "2".

Page 177, Figure 10-4: Change the comment that appears to the right of the B_{63}xA term to "If B<0, subtract A from the most-significant half of unsigned product".

Page 181, Figure 10-5: Change "$A_{HI}B_{LO}$" that appears to the right of the MLA instruction to "$A_{HI}B_{HI}$".

Page 187, Programming Problem 2: Insert the word "to" between the words "program" and "compute".

APPENDIX B: GRAPHICS LIBRARY FUNCTIONS

Page 233: Change the name of function ClearDisplay to ClearScreen

Page 233: Add the following two function prototypes just below the prototype for function SetColor:

```
void SetForeground(uint32_t color) ; // Same as SetColor
void SetBackground(uint32_t color) ;
```

Page 234: Remove the last parameter ("int alignment") from the function prototype for function DisplayStringAt.

APPENDIX C: TOUCH SCREEN LIBRARY FUNCTIONS

Page 235, Listing C-1: Insert a call to `TS_Init()` immediately before the `while` statement.