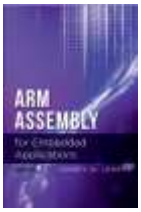


ARM Assembly for Embedded Applications, 3rd edition
ISBN 978-1-54390-804-6

The ISBN number (978-1-54390-804-6) printed on the rear of the title page inside the book is correct. However, the ISBN number (978-1483590417) printed on the outside of the rear cover is NOT correct. Corrected in a second printing; otherwise the 1st and 2nd printings are identical.

CHAPTER 4: COPYING DATA

Section 4.7 (Addressing Modes), page 56, 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 48, 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 51, Table 4-2: Add the following note just below the table: "The memory operand of LDRD must reside at a mod 4 (word aligned) address to avoid an address alignment fault".

Page 54, Table 4-3: Add the following note just below the table: "The memory operand of STRD must reside at a mod 4 (word aligned) address to avoid an address alignment fault".

Page 66, Table 4-7: Add the following note just below the table: "The memory operands of these instructions must reside at a mod 4 (word aligned) address to avoid an address alignment fault".

Page 67, 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 6: MAKING DECISIONS AND WRITING LOOPS

Page 95-96: 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.
```

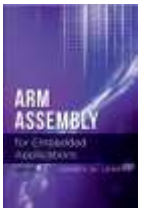
Finally, note that in all cases except EQ and NE, we can avoid modifying one register by replacing the SUBS instruction by a CMP instruction.

CHAPTER 7: MANIPULATING BITS

Page 122, Listing 7-2: Replace "SUB R0.R1.R0,ASR 31" by "ADD R0,R1,R0,LSR 31" . Modify the paragraph that follows accordingly. (The SUB instruction works, but the ADD is more intuitive.)

CHAPTER 8: MULTIPLICATION AND DIVISION REVISITED

Page 134: Table 8-2 was printed three times – once on page 134 and twice on page 135. Replace the entire section 8.1.1 with the document found here.



ISBN 978-1-54390-804-6

Page 136, Table 8-3, 3rd row (11x), 2nd column (Composition), 2nd line: Replace $R0 = R1 + 2xR0$ by $R0 = R0 + 2xR1$.

Page 138, below Table 8-5:
Page 139, last sentence before section 8.3:
Page 142, code sequences for division by + and – 7:

> Replace the SUB instruction by an ADD and (in the same line) replace the ASR 31 by LSR 31. (The SUB instruction works, but the ADD is more intuitive.)

Section 8.3 (Division by an Arbitrary Constant), page 142, right-hand column (code for "Divides by -7"):

Replace the instruction, "ADD R0,R1,**R0**,LSR 31" by "ADD R0,R1,R1,LSR 31"

Section 8.3 (Division by an Arbitrary Constant), page 143, paragraph in the middle of the page:

Replace the entire paragraph by the following:

> The SMMUL instruction computes the most-significant half of a 64-bit signed product and eliminates the need to use register R2 in the previous code. The SMMLA instruction does the same, but also adds the value of another 32-bit operand to the result. This instruction can thus replace the SMULL/ADDS.N sequence in the earlier code that divides by +7. The SMMLS instruction computes the difference of a 32-bit operand less the most-significant half of a 64-bit operand. Unfortunately, this doesn't help simplify the code that divides by -7. There are no unsigned versions of these instructions.

Section 8.3 (Division by an Arbitrary Constant), page 172, Table 8-6:

In the third column of the table, second row, replace "$Rd \leftarrow (Rn \times Rm)$ <63..32> + R_a" by "$Rd \leftarrow R_a + (Rn \times Rm)$ <63..32>".

In the third column of the table, third row, replace "$Rd \leftarrow (Rn \times Rm)$ <63..32> − R_a" by "$Rd \leftarrow R_a - (Rn \times Rm)$ <63..32>".

CHAPTER 10: WORKING WITH FIXED-POINT REALS

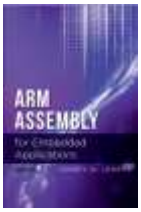
Page 190, Section 10.5.4, 1st paragraph: Replace every reference to R5 by R12.

Page 197, Function Q32Ratio: Edit the line that computes rounding so that it reads

```
rounding = (top >= 0) ? +btm/2 : -btm/2 ;
```

Page 198, Function Average: Replace the function call "Q32Ratio(total, n)" by the expression "total / n".

CHAPTER 11: MULTIMEDIA PROCESSING



ISBN 978-1-54390-804-6

Page 213, Table 11-4: Delete the last two rows of the table; the instructions UQADD and UQSUB re not implemented in the Cortex-M4F.

APPENDIX B: BASIC SUPPORT LIBRARY

Page 284: Add the following two functions:

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

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

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

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

APPENDIX C: GRAPHICS LIBRARY

Page 285: Change the name of function `ClearDisplay` to `ClearScreen`

Page 285: 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 286: Remove the last parameter ("int alignment") from the function prototype for function DisplayStringAt.

APPENDIX D: TOUCH SCREEN LIBRARY

Page 287, Listing C-1: Insert a call to `TS_Init()` immediately before the `while` statement.