Assembly Program Examples (2A)

Young Won Lim
1/8/20

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

ARM Directive: **AREA**

AREA HelloW, READONLY

instructs the assembler to assemble
a new **code** or **data** area.
areas are independent, named, indivisible
chunks of **code** or **data**
that are manipulated by the linker.

Attributes: CODE,
DATA,
READONLY,
READWRITE,
COMMON

ARM Directive: **EQU**

```
SWI_WriteC      EQU     &0              ; SWI_WriteC = &0
SWI_Exit        EQU     &11             ; SWI_Exit = &11
```

The EQU directive gives a symbolic name
to a **numeric constant**.
* is a synonym for EQU.

Hexadecimal numbers are preceded by an ampersand &
to distinguish them from decimal numbers.

```
&0          ; 0x0     = 0
&11         ; 0x11    = 17
```

ARM Directive: **ENTRY**

ENTRY

The ENTRY directive declares
its offset in its containing AOF area
to be the unique entry point
to any program containing the area.

You must specify one and only one
ENTRY directive for a program.

> ARM Object Format (AOF) areas are
> independent, named, indivisible sequences of code or data.

ARM Directive: **END**

END

The END directive informs the assembler that it has reached the end of a source file.

ARM Directive: **ADR**

```
ADR        r1, TEXT        ; r1 ← TEXT
```

Load a program-relative or register-relative address into a register.

ARM Directive: =

```
TEXT        =        ”Hello World”, &0a, &0d, 0
```

= is usually suffixed by an immediate constant and
instructs the assembler to put the constant into a nearby literal pool
and generate a pc relative memory operand to load it.

This is useful since the ARM instruction format
doesn't have enough space to store a full 32 bit constant.

https://stackoverflow.com/questions/37840754/what-does-an-equals-sign-on-the-right-side-of-a-ldr-instruction-in-arm-mean

ARM Directive: **ALIGN**

ALIGN {expr{,offset{,pad{,padsize}}}}

expr : is a numeric expression evaluating
to any power of 2 from 20 to 231
offset : can be any numeric expression
pad : can be any numeric expression
padsize : can be 1, 2 or 4.

The ALIGN directive aligns the current location
to a specified boundary
by padding with zeros or NOP instructions.

ARM Directive:**DCD**

{label} DCD{U} expr{,expr}

expr is either:

a numeric expression.
a PC-relative expression.

The DCD directive allocates one or more words of memory,
aligned on four-byte boundaries,
and defines the initial runtime contents of the memory.

& is a synonym for DCD.

DCDU is the same, except that the memory alignment is arbitrar

ARM System Calls

SWI_WriteC (SWI 0)

Write a **byte**, passed in **r0**, to the debug channel.
When executed under the symbolic debugger,
the character will appear on the display device
connected to thedebugger.

SWI_Write0 (SWI 2)

Write the **null-terminated string**, pointed to by **r0**, to the debug channel.
When executed under the symbolic debugger,
the characters will appear on the display device
connected to the debugger.

SWI_ReadC (SWI 4)

Read a byte from the debug channel, returning it in register 0.
The read is notionally from the keyboard attached to the debugger.

SWI_Exit (SWI 0x11)

Halt emulation.
This is the way a program exits cleanly, returning control to the debugger.

http://www.ee.ic.ac.uk/pcheung/teaching/ee2_computing/swi.pdf

Example 1

```
                 AREA       HelloW, READONLY
SWI_WriteC          EQU     &0
SWI_Exit            EQU     &11
                 ENTRY
START            ADR        r1, TEXT
LOOP             LDRB       r0, [r1], #1           ; r0 ← [r1] ; r1 ← r1+1; byte
                 CMP        r0, #0                 ; r0 – #0
                 SWINE      SWI_WriteC             ; if ≠ 0
                 BNE        LOOP                   ; if ≠ 0
                 SWI        SWI_Exit
TEXT             =          ”Hello World”, &0a, &0d, 0
                 END
```

Post-index Address

LDRB **r0,** **[r1], #1**

Byte transfer

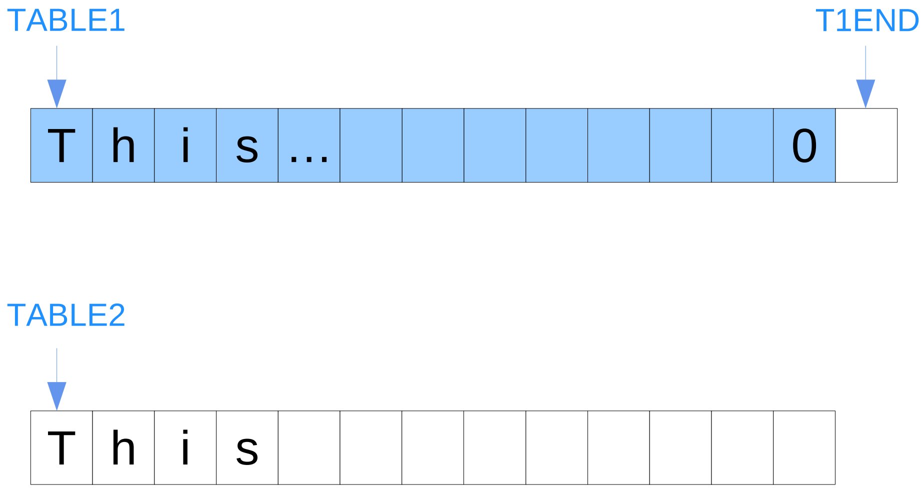
r0 ← [r1] ; access first

r1 ← (r1 + #1) ; then index

post-index – always updated
regardless of !

Example 2(a)

```
           AREA       BlkCpy, CODE, READONLY
SWI_WriteC     EQU    &0
SWI_Exit       EQU    &11
           ENTRY
           ADR        r1, TABLE1
           ADR        r2, TABLE2
           ADR        r3, T1END
```



Example 2(b)

```
LOOP1     LDR       r0, [r1], #4        ; r0 ← [r1] ; r1 ← r1+1
          STR       r0, [r2], #4        ; r0 → [r2] ; r2 ← r2+1
          CMP       r1, r3              ; r1 – T1END
          BLT       LOOP1               ; B if r1 < T1END
          ADR       r1, TABLE2
LOOP2     LDRB      r0, [r1], #1        ; r0 ← [r1] ; r1 ← r1+1; byte
          CMP       r0, #0              ; r0 – #0
          SWINE     SWI_WriteC          ; SWI if r0 ≠ 0
          BNE       LOOP2               ; B if r0 ≠ 0
          SWI       SWI_Exit

TABLE1    =         ”This is the right string!”, &0a, &0d, 0
T1END
          ALIGN
TABLE2    =         ”This is the right string!”, 0
          END
```

Example 3(a)

```
                AREA        Hex_Out, CODE, READONLY
SWI_WriteC              EQU     &0
SWI_Exit                EQU     &11
                ENTRY
                LDR         r1, VALUE           ; r1 ← & 12345678
                BL          HexOut
                SWI         SWI_Exit
VALUE           DCD         &12345678
```

Example 3(b)

```
HexOut    MOV       r2, #8                 ; nibble count 8
LOOP      MOV       r0, r1, LSR #28        ; r0 ← (r1 >> 28)   ; 4*7
          CMP       r0, #9                 ; r0 – #9
          ADDGT     r0, r0, #”A”-10        ; r0 ← r0 + (“A”-10)
          ADDLE     r0, r0, #”0”           ; r0 ← r0 + “0”
          SWI       SWI_WriteC
          MOV       r1, r1, LSL #4         ; r1 ← (r1 << 4)
          SUBS      r2, r2, #1             ; r2 ← r2 - 1
          BNE       LOOP
          MOV       pc, r14
          END
```

```
A=10  0 + ″A″  ″A″
B=11  1 + ″A″  ″B″
C=12  2 + ″A″  ″C″
D=13  3 + ″A″  ″D″
E=14  4 + ″A″  ″E″
F=15  5 + ″A″  ″F″
```

Example 3(b)

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	**65**	**A**	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	**48**	**0**	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

Example 4(a)

output a text string
without a separate **data** area for the text

But this will be inefficient when the processor has
separate **data** and **instruction** cache
StrongARM is such a case

```
                BL       TextOut
r14  →          =        ”Test string”, &0a, &0d, 0
                ALIGN
```

BL will use the link register r14
as a return address from the call

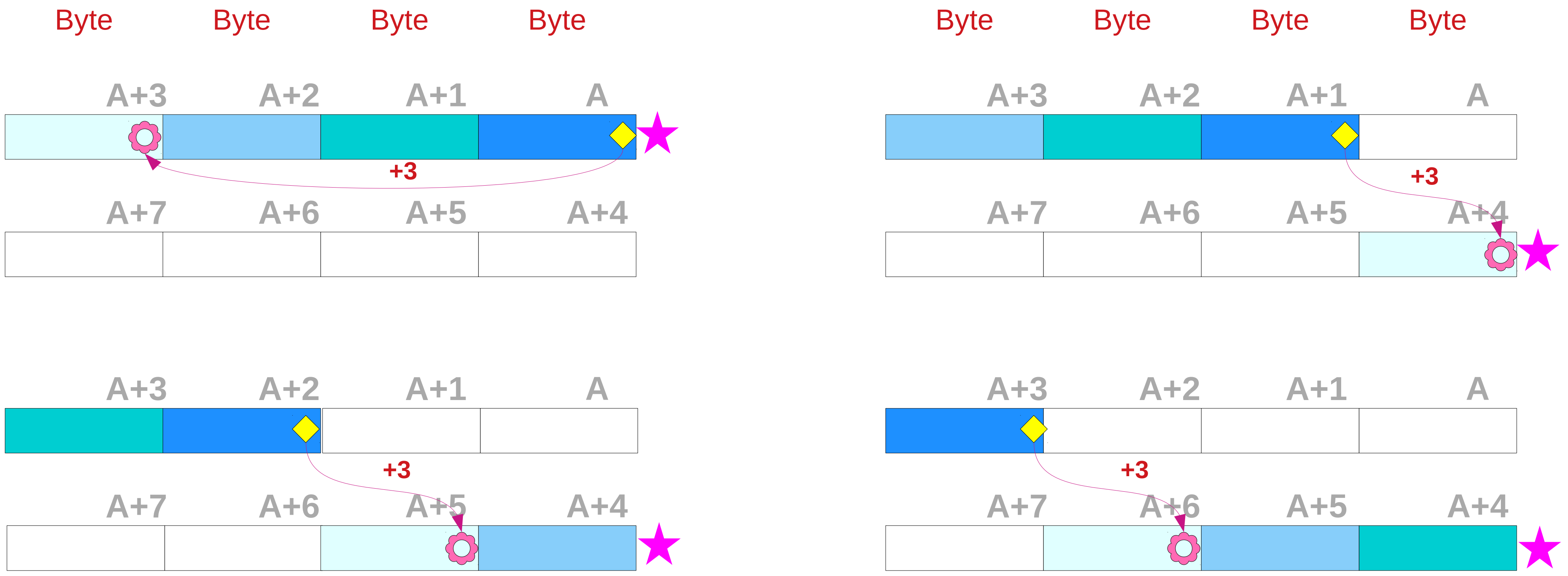
Example 4(b)

```
            AREA        Text_Out, CODE, READONLY
SWI_WriteC          EQU         &0
SWI_Exit            EQU         &
            ENTRY
            BL      TextOut
            =       ”Test String”, &0a, &0d, 0
            ALIGN
            SWI     SWI_Exit

TextOut     LDRB        r0, [r14], #1   ; r0 ← [r14],  r14 ← r14+1
            CMP         r0, #0          ; r0 – #0
            SWINE       SWI_WriteC
            BNE         TextOut
            ADD         r14, r14, #3    ; r14 ← r14+3
            BIC         r14, r14, #3    ; r14 ← r14 & (!3)
            MOV         pc, r14
            END
```

Example 4(b)

```
            AREA        Text_Out, CODE, READONLY
SWI_WriteC         EQU         &0
SWI_Exit           EQU         &
            ENTRY
            BL     TextOut
            =      ”Test String”, &0a, &0d, 0
            ALIGN
            SWI    SWI_Exit

TextOut     LDRB        r0, [r14], #1        ; r0 ← [r14],  r14 ← r14+1
            CMP         r0, #0               ; r0 – #0
            SWINE       SWI_WriteC
            BNE         TextOut
            ADD         r14, r14, #3         ; r14 ← r14+3
            BIC         r14, r14, #3         ; r14 ← r14 & (!3)
            MOV         pc, r14
            END
```

Align operations



```
                ✿    ◆
ADD        r14, r14, #3        ; r14 ← r14+3
BIC        r14, r14, #3        ; r14 ← r14 & (!3)
           ★    ✿
```

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf