Pseudo Instructions in GNU ARM Assembler

Peng-Sheng Chen

Fall, 2017

NOP

- This pseudo op will always evaluate to a legal ARM instruction that does nothing.
- Currently it will evaluate to **MOV r0, r0**.

LDR <register>, =<exp>

- **`LDR <register>,=label`**
  - If you plan to reference labels in other sections of code
- **`LDR <register>,=constant`**
  - If expression evaluates to a numeric constant then a MOV or MVN instruction will be used in place of the LDR instruction, if the constant can be generated by either of these instructions. Otherwise the constant will be placed into the nearest literal pool (if it not already there) and a PC relative LDR instruction will be generated.

Example (1)

```
/* ============================ */
/*        DATA section          */
/* ============================ */
        .data

/* --- variable a --- */
a:
        .word 1
        .word 2

/* --- variable b --- */
b:
        .word 3

/* --- variable b --- */
c:
        .space 8, 0
```

$$a = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

$$b = \begin{bmatrix} 3 \end{bmatrix}$$

$$c = b \times a$$

$$c = \begin{bmatrix} ? & ? \end{bmatrix}$$

Example (2)

```
/* ========================== */
/*        TEXT section        */
/* ========================== */
        .section .text
        .global main
        .type main,%function
.matrix:
        .word a
        .word b
        .word c
main:
        ldr r0, .matrix
        ldr r1, [r0], #4    /* r1 := mem32[r0] */
                            /* r0 := r0 + 4    */
        ldr r2, [r0]

        ldr r0, .matrix + 4
        ldr r3, [r0]        /* r3 := mem32[r0] */

        ldr r4, .matrix + 8

        mul r5, r3, r1
        mul r6, r3, r2

        str r5, [r4], #4    /* mem32[r4] := r5 */
                              /* r4 := r4 + 4    */
        str r6, [r4]
        nop
```

抓到a的address，放入r0

抓到第一個a的值

抓到第二個a的值

Example (3)

```
/* ========================== */
/*        TEXT section        */
/* ========================== */
        .section .text
        .global main
        .type main,%function

main:
        ldr r0, =a
        ldr r1, [r0], #4    /* r1 := mem32[r0] */
                            /* r0 := r0 + 4    */
        ldr r2, [r0]

        ldr r0, =b
        ldr r3, [r0]        /* r3 := mem32[r0] */

        ldr r4, =c

        mul r5, r3, r1
        mul r6, r3, r2

        str r5, [r4], #4    /* mem32[r4] := r5 */
                              /* r4 := r4 + 4    */
        str r6, [r4]
        nop
```

抓到a的address，放入r0

抓到第一個a的值

抓到第二個a的值

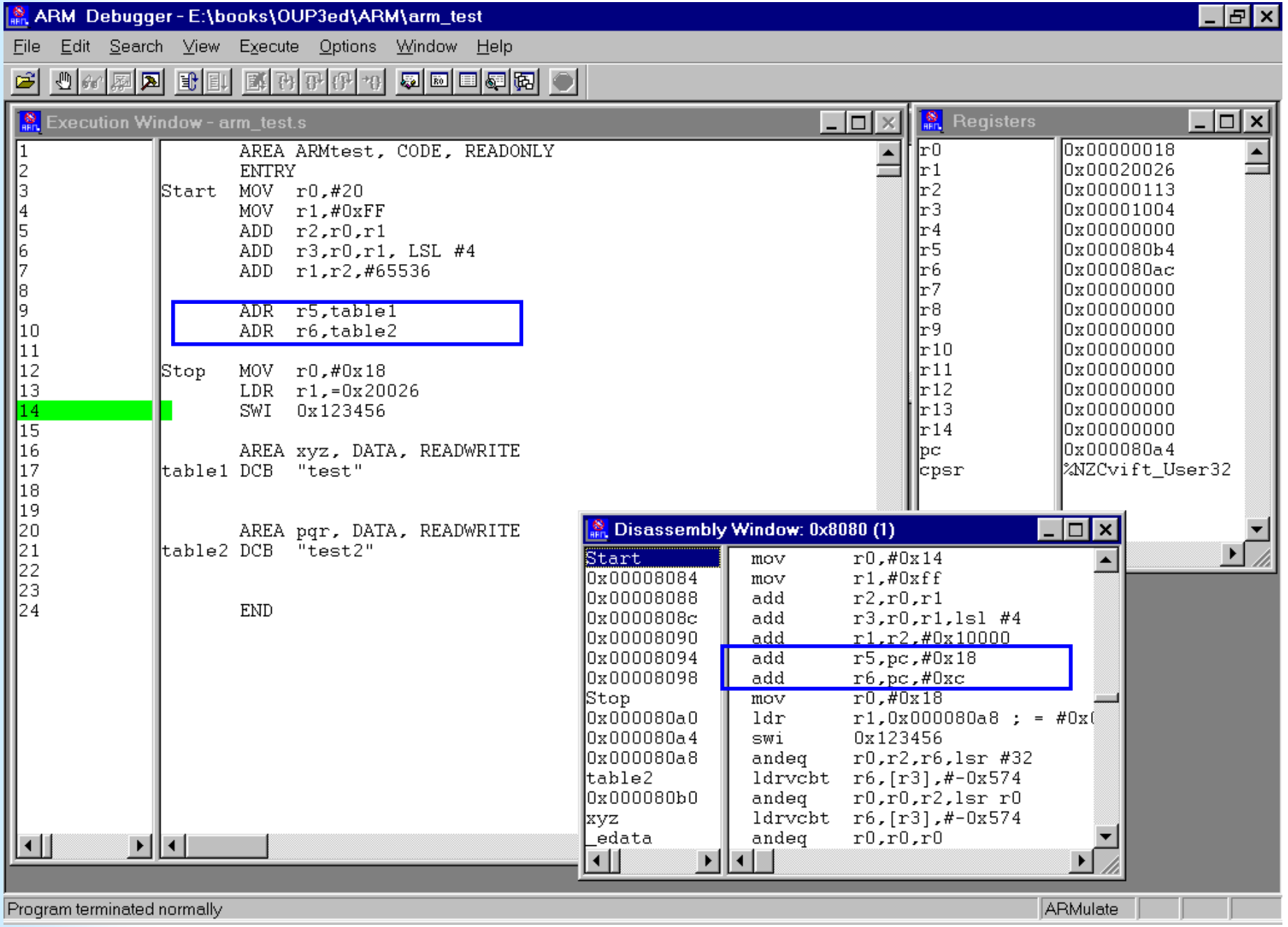
LDR <register>, =<exp>

```
ldr     r0, =0x12345678
```

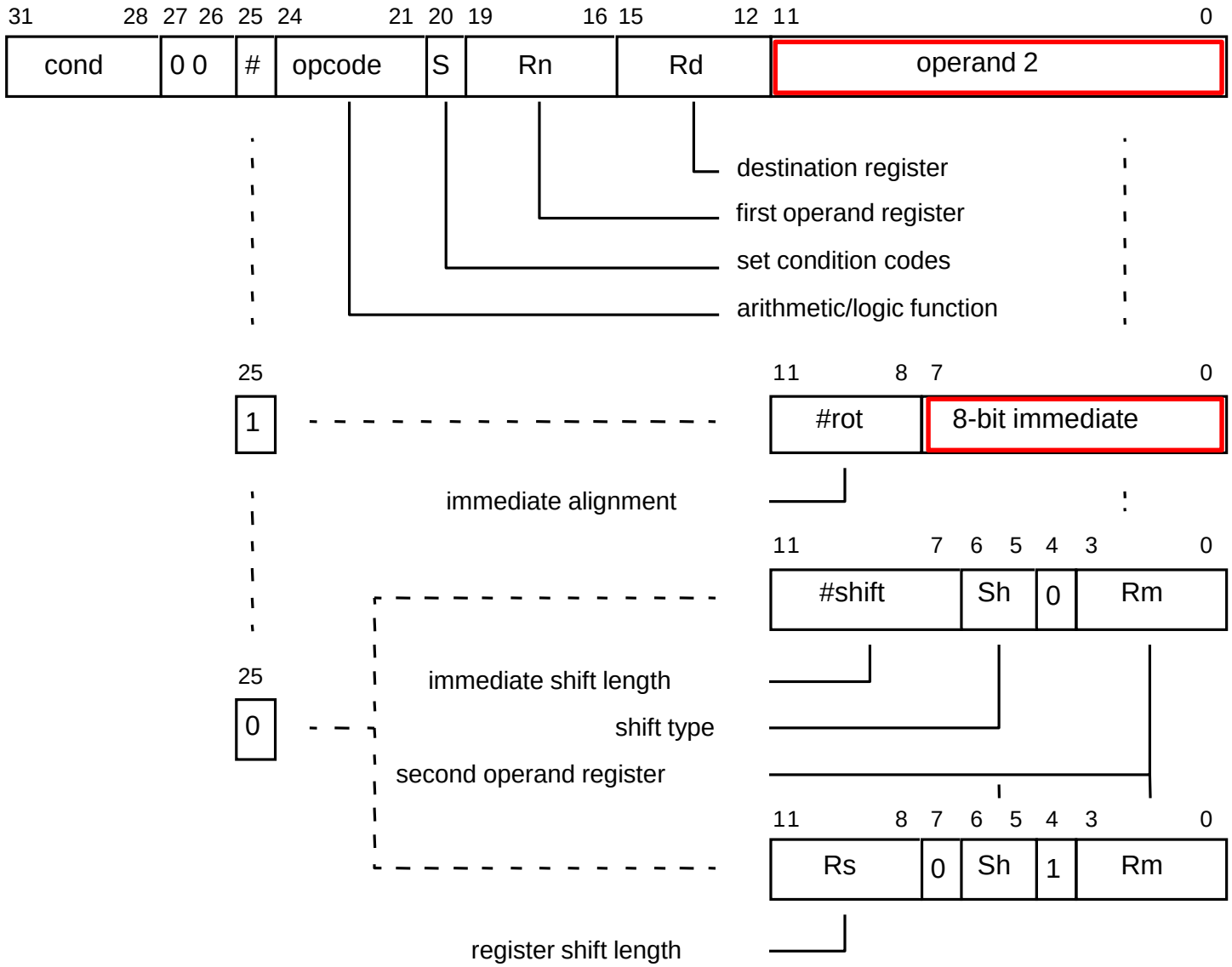
```
ldr     r0, [pc, #offset]
...
.word   0x12345678
```

Load an Address into Register

- ADR is a **pseudo** instruction
- Assembler will transfer pseudo instruction into a sequence of appropriate normal instructions
- Assembler will transfer ADR into a single ADD, or SUB instruction to load the address into a register.
- 只能使用於獲取同一個file與section內之label.

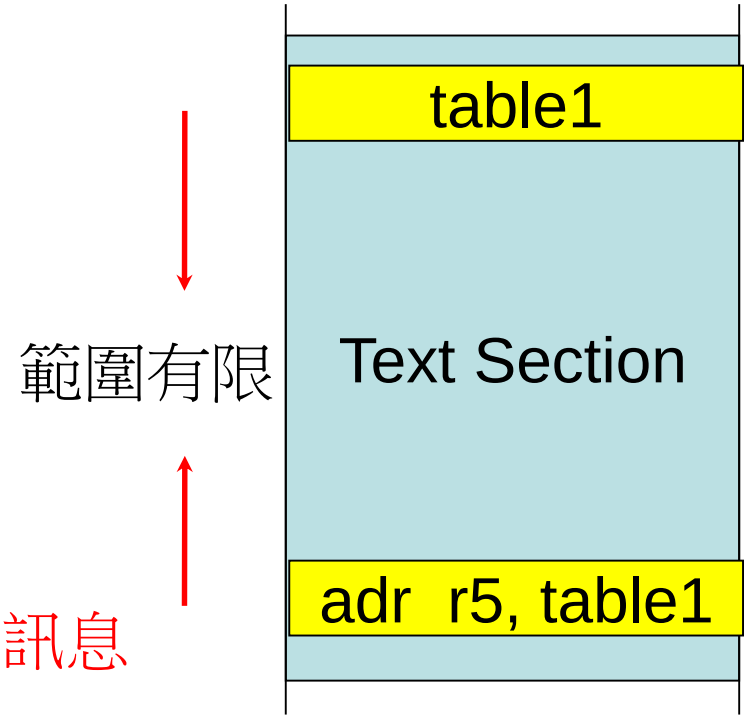

ARM Debugger - E:\books\OUP3ed\ARM\arm_test
File Edit Search View Execute Options Window Help
Execution Window - arm_test.s
1 AREA ARMtest, CODE, READONLY
2 ENTRY
3 Start MOV r0,#20
4 MOV r1,#0xFF
5 ADD r2,r0,r1
6 ADD r3,r0,r1, LSL #4
7 ADD r1,r2,#65536
8
9 ADR r5,table1
10 ADR r6,table2
11
12 Stop MOV r0,#0x18
13 LDR r1,=0x20026
14 SWI 0x123456
15
16 AREA xyz, DATA, READWRITE
17 table1 DCB "test"
18
19
20 AREA pqr, DATA, READWRITE
21 table2 DCB "test2"
22
23
24 END
Registers
r0 0x00000018
r1 0x00020026
r2 0x00000113
r3 0x00001004
r4 0x00000000
r5 0x000080b4
r6 0x000080ac
r7 0x00000000
r8 0x00000000
r9 0x00000000
r10 0x00000000
r11 0x00000000
r12 0x00000000
r13 0x00000000
r14 0x00000000
pc 0x000080a4
cpsr %NZCvift_User32
Disassembly Window: 0x8080 (1)
Start mov r0,#0x14
0x00008084 mov r1,#0xff
0x00008088 add r2,r0,r1
0x0000808c add r3,r0,r1,lsl #4
0x00008090 add r1,r2,#0x10000
0x00008094 add r5,pc,#0x18
0x00008098 add r6,pc,#0xc
Stop mov r0,#0x18
0x000080a0 ldr r1,0x000080a8 ; = #0x
0x000080a4 swi 0x123456
0x000080a8 andeq r0,r2,r6,lsr #32
table2 ldrvcbt r6,[r3],#-0x574
0x000080b0 andeq r0,r0,r2,lsr r0
xyz ldrvcbt r6,[r3],#-0x574
_edata andeq r0,r0,r0
Program terminated normally
ARMulate

Data Processing Instruction Binary Encoding



ADR

- ADR r5, table1 => ADD r5, pc, #0x18



如果超過範圍，則會產生下面的錯誤訊息

x.s: Assembler messages:

x.s:??: Error: invalid constant (xxxxx) after fixup

ADRL

- ADRL r5, table1 => 轉換成兩行ARM instructions
  - 解決8-bits範圍太小的問題
- The "adr" pseudo-instruction is always replaced by ONE real instruction.
- The "adrl" pseudo-instruction is always replaced by TWO real instructions.