WEEK BY WEEK

Week 01 (Thursday, Sept. 8 -- Saturday, Sept. 10)

Expectations, the start and the Introduction

Week 02 (Sunday, Sept. 11 -- Saturday, Sept. 17)

Number systems, sign representations, and two's complement

Week 03 (Sunday, Sept. 18 -- Saturday, Sept. 24)

Floating-point representation

Week 04 (Sunday, Sept. 25 -- Saturday, Oct. 1)

Digital logic

Week 05 (Sunday, Oct. 2 -- Saturday, Oct. 8)

ISA & introduction to assembly language

Week 06 (Sunday, Oct. 9 -- Saturday, Oct. 15)

First midterm test

Week 07 (Sunday, Oct. 16 -- Saturday, Oct. 22)

ARM directives, pseudo, data-processing, and shift instructions

Week 08 (Sunday, Oct. 23 -- Saturday, Oct. 29)

Branching/looping & instruction encoding/decoding, and addressing modes

Week 09 (Sunday, Oct. 30 -- Saturday, Nov. 5)

Reading week

Week 10 (Sunday, Nov. 6 -- Saturday, Nov. 12)

Second midterm test

Week 11 (Sunday, Nov. 13 -- Saturday, Nov. 19)

LDR/STR encoding/decoding, Examples, and Stacks

Week 12 (Sunday, Nov. 20 -- Saturday, Nov. 26)

Block move, Block move encoding/decoding, Subroutine call/return, and summary

Week 13 (Sunday, Nov. 27 -- Saturday, Dec. 3)

Stack frame & passing parameters

Week 14 (Sunday, Dec. 4 -- Thursday, Dec. 8)

Last week!!

ARM Load and Store Encoding

- Encode the following ARM instructions.

```
LDR R1,[R2]
LDR R1,[R2],#0
LDR R1,[R2,#0]
LDR R1,[R2,#0]!
STR R1,[R2]
STR R1,[R2],#0
STR R1,[R2,#0]
STR R1,[R2,#0]!
```

- Is there any *effective* difference between the 4 LDR instructions?
- Is there any *effective* difference between the 4 STR instructions?

stack suffix -> STR suffix

LDR otherwise.

R11: FP : base of stack

R14: LR : Address of return

R13: SP : TOS

R15: PC : next instruction.

* due to pipelining, PC is 8 bytes from current instruction.

e.g. LDR R0, [PC, #0x0018] ; 0x00+0x18+0x8 = 0x20.

Besides instruction ending with S, CMP, CMN, TEQ, TST update flags as well.

SUB. TEQ XOR TST Anding

Static shift

LSL: 补0, #0-31

LSR: 补0, #1-32.

ASR: 补符号, #1-32. , ASL同LSL

ROR: 出补in, #1-31, ROL用ROR替代实现

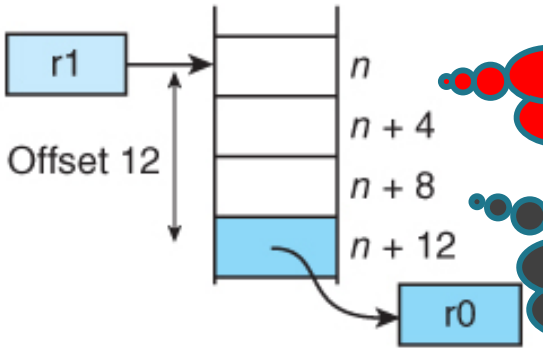
RRX: rotate shift right, one-bit only

```
while:  CMP R0, #0 ; statement
        BNE Exit
        code ....
        B while
Exit    code .....
```

```
        MOV R0, #10
For     code ...
        SUBS R0, R0, #1
        BNZ Loop
        post-loop codes ...
```

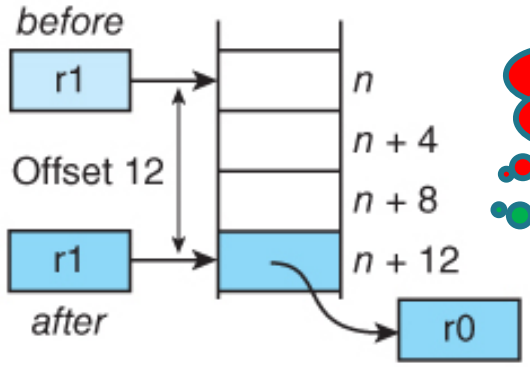
handling literals

Register Indirect Addressing with Offset



Adjust the pointer then use the adjusted pointer.

Do NOT write-back (do not update) the adjusted pointer.



Adjust the pointer then use the adjusted pointer.

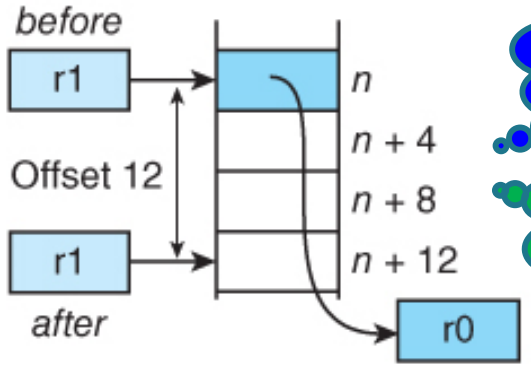
Write-back (update) the adjusted pointer.

(a) `LDR r0,[r1,#12]`
Offset added to base register to generate effective address. Operand accessed at effective address. Base register remains unchanged.

(b) `LDR r0,[r1,#12]!`
Offset added to base register to generate effective address. Operand accessed at effective address. Base register updated after access.

the pointer would never updated and the program would reapeatly load the first elem.

Why do not we have "Use the original base pointer then adjust the pointer" with "Do NOT write-back (do not update) the adjusted pointer"?



Use the original base pointer then adjust the pointer.

Write-back (update) the adjusted pointer.

LDR r0, [r1, #12]: post index, not write-back
LDR r0, [r1, #12]!: pre index, write back
LDR r0, [r1], #12: post index, write back.

(c) `LDR r0,[r1], #12`
Effective address specified by base register. Operand accessed at effective address. Offset added to base register after the access.

FD

STMFD SP! {FP}

MOV FP, SP

SUB SP, #8 ← size of the stack.

collapse

mov SP, FP

LDMFD SP!, FP