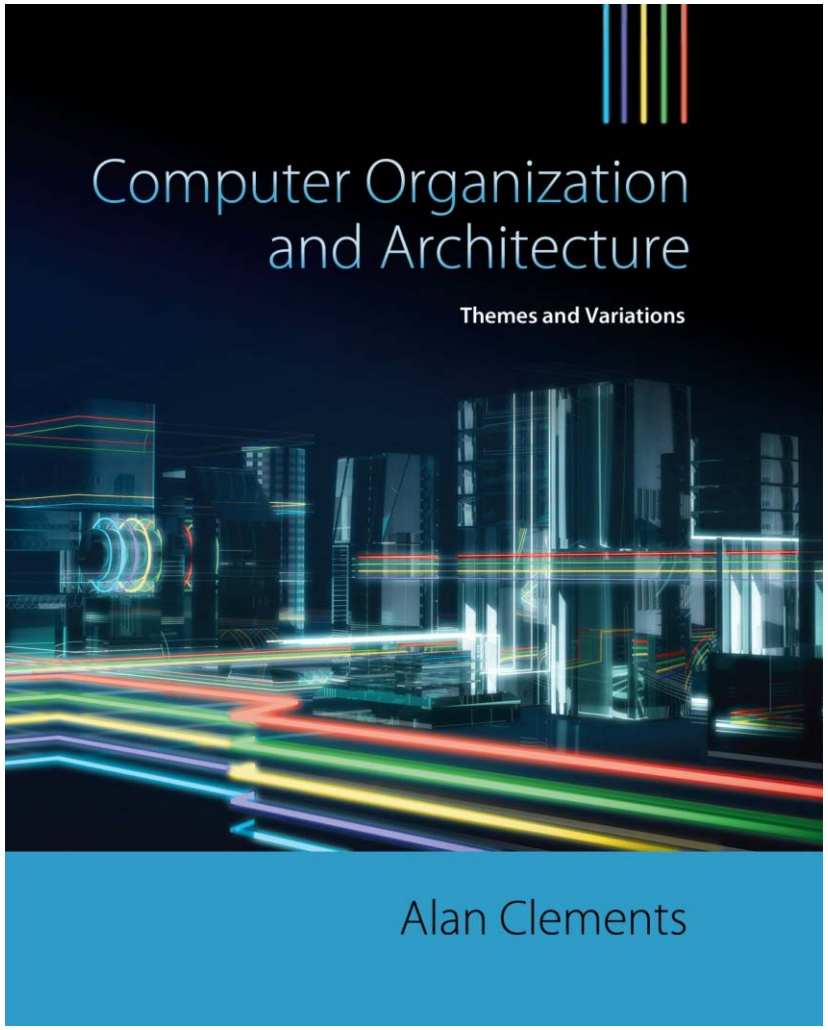
Part 0x4

Chapter 3

Architecture and Organization



CENGAGE Learning

These slides are being provided with permission from the copyright for in-class (CS2208B) use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides and/or lecture recordings are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.

Music: *"Corporate Success"* by *Scott Holmes*, used under *Attribution-NonCommercial License*

Pseudo instructions

- A *pseudo instruction* is an operation that the programmer can use when writing code.
  - The actual instruction ***does not*** have a ***direct*** machine language equivalent.
    - For example, you ***can't*** write MOV **r0**,#0x12345678 to load register r0 with the 32-bit value 0x12345678 because the instruction is only 32 bits long in total.
    - Instead, you can use LDR **r0**,=0x12345678 *pseudo instruction,*
      Yes, it is = not #

      It is NOT MOV r0, = 0x12345678
      - the assembler will generate suitable code to carry out the same action.
        - *store the constant* 12345678_{16} in a so-called *literal pool* or *constant pool* somewhere in memory ***after the program***
        - *generates suitable code* to load the stored constant 12345678_{16} to r0

This slide is modified from the original slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

Pseudo instructions

- Another *pseudo instruction* is ADR **r0**, label, which loads the *32-bit address* of the line 'label' into register r0, using the appropriate code generated by the assembler
- The following fragment demonstrates the use of the ADR *pseudo instruction*.

This LDR instruction here is ***NOT*** a pseudo instruction

```
ADR   r1,MyArray ;set up r1 to point to MyArray
                 ; loads register r1 with the 32-bit address of MyArray
...
LDR   r3,[r1]    ;read an element using the pointer

MyArray DCD  0x12345678 ;the address of this data will be loaded to r1
```

- The programmer does not have to know how the assembler generates suitable code to implement such *pseudo instructions*
- All this is done automatically.

But as a student, you need to know it!!

- This can be realized by utilizing the *program counter relative addressing*

This slide is modified from the original slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

Program Counter Relative Addressing

- Register *indirect relative addressing allows* us to
  - *specify the location of an operand with respect to a register value.*

get the value of r1, use it as a pointer to the memory, get the value in the memory and store it in r0

- `LDR r0,[r1]` specifies that the operand address is in r1
- `LDR r0,[r1, #16]` specifies that the operand is 16 bytes onward from r1.

r1 → 16 bytes

Array [ax+15]

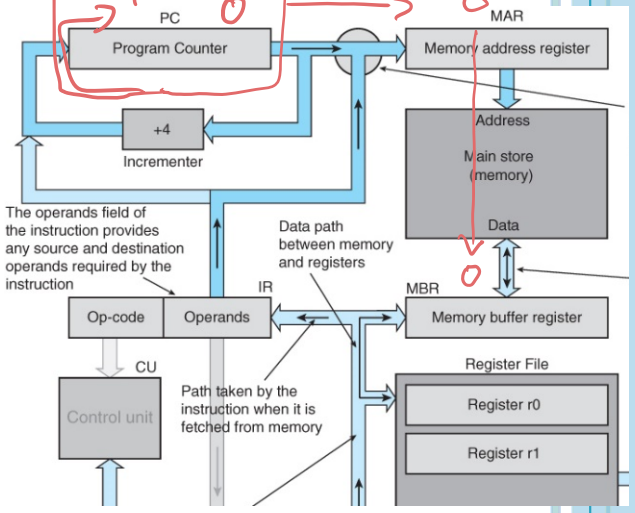
- Suppose that we use *r15*, i.e., *the PC*, to generate an address by writing `LDR r0,[PC,#16]`.

position offset

current position + offset + 8

  - The operand is 16 bytes onward from the PC
  - i.e., 8 + 16 = 24 bytes from the current instruction.
    - *The ARM's PC **in most of the cases** is 8 bytes from the current instruction to be executed,* due to *pipelining* (automatically fetches the next instruction before the current one has been executed).

initialized to be 0 once the program is executed, it becomes 8



- If the program and its data are relocated elsewhere in memory, the *relative offset* does not change.

This slide is modified from the original slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

This is the original page.....

Pseudo instructions

FIGURE 3.20 Code using pseudoinstructions

5 instructions.

3 DCD literal pool

```
C:\Keil\ARM\Examples\test.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Disassembly
    4:                  LDR   r0,=0x12345678
0x00000000  E59F0018  LDR     R0,[PC,#0x0018]
    5:                  ADR   r1,Table
0x00000004  E28F1008  ADD     R1,PC,#0x00000008
    6:                  ADR   r2,Table1
0x00000008  E28F2008  ADD     R2,PC,#0x00000008
    7:                  LDR   r3, = 0xAAAAAAAA
0x0000000C  E59F3010  LDR     R3,[PC,#0x0010]
    8:                  LDR   r4,P3
0x00000010  E59F4004  LDR     R4,[PC,#0x0004]
0x00000014  ABCDDCBA  BLGE    0xFF377304
0x00000018  FFFFFFFF  (???)
0x0000001C  22222222  EORCS   R2,R2,#0x20000002
0x00000020  12345678  EORNES  R5,R4,#0x07800000
0x00000024  AAAAAAAA  BGE     0xFEAAAAD4
0x00000028  00000000  ANDEQ   R0,R0,R0
PseudoInst.asm
01          AREA ConstPool, CODE, READONLY
02          ENTRY
03
04          LDR   r0,=0x12345678
05          ADR   r1,Table
06          ADR   r2,Table1
07          LDR   r3, = 0xAAAAAAAA
08          LDR   r4,P3
09
10  Table   DCD   0xABCDDCBA
11  Table1  DCD   0xFFFFFFFF
12  P3      DCD   0x22222222
```

pesudo instruction

This slide is modified from the ARM® Software Mahmoud R. El-Sakka.

LDR R0, [PC, #0x0018] : use PC as a pointer, offset it 18 in hex, and copy this location in memory to R0

Once the instruction is executed, the PC is pointing 8, so the actual address is $0x00+0x08+0x18 = 0x20_{16}$

The offset is determined at where the memory location is empty. right after the program $20_{16} - 8_{16} = 18_{16}$, so 18 is the offset

PC + offset + implement

① Store the address 14 in R1 ② Calculate the position $4+8+8 = 14_{16}$

Address of the table is 14 offset = 14 - 4 - 8 = 8

$18_{16} = 8_{16} + 8_{16} + 8_{16}$., Storing address 18 in R_2.

offset = 18 - 8 - 8 = 8

① Find the first empty memory location at 24

② offset = 24 - C - 8 = 10 / C + 8 + 10 = 24.

Copy the value of P3 to R4.

first empty location : 28. offset = 28 - 10 - 8 = 10

Having ADR r4, P3 at line 08 will utilize the ADD instruction and the PC value to load the address of P3 in R4. To be translated to: `ADD r4, PC, #0x4`

Pseudo instructions

FIGURE 3.20 Code using pseudoinstructions

To understand all these questions, try to put each pair of instructions in an assembly program and analyze the disassembly result.

1. What is the difference between LDR r4, P3 and ADR r4, P3 ?

2. What will be the generated code if you replaced LDR r4, P3 by ADR r4, P3 ?

3. What is the difference between ADR r4, P3 and LDR r4, = P3 ?

Note that there is a difference between LDR r4, P3 and LDR r4, = P3

The 1st one will load the *VLAUE* of P3 in R4.

The 2nd one will store the *ADDRESS* of P3 at the literal pool and then load the value of this address in R4.

```
4:                  LDR   r0,=0x12345678
0x00000000  E59F0018  LDR   R0,[PC,#0x0018]   0x00 + 0x08 + 0x18 = 0x20
5:                  ADR   r1,Table
0x00000004  E28F1008  ADD   R1,PC,#0x00000008  0x04 + 0x08 + 0x08 = 0x14
6:                  ADR   r2,Table1
0x00000008  E28F2008  ADD   R2,PC,#0x00000008  0x08 + 0x08 + 0x08 = 0x18
7:                  LDR   r3, = 0xAAAAAAAA
0x0000000C  E59F3010  LDR   R3,[PC,#0x0010]   0x0C + 0x08 + 0x10 = 0x24
8:                  LDR   r4,P3
0x00000010  E59F4004  LDR   R4,[PC,#0x0004]   0x10 + 0x08 + 0x04 = 0x1C
0x00000014  ABCDDCBA  BLGE    0xFF377304
0x00000018  FFFFFFFF  (???)
0x0000001C  22222222  EORCS   R2,R2,#0x20000002
0x00000020  12345678  EORNES  R5,R4,#0x07800000
0x00000024  AAAAAAAA  BGE     0xFEAAAAD4
0x00000028  00000000  ANDEQ   R0,R0,R0
```

PseudoInst.asm

```
01          AREA ConstPool, CODE, READONLY
02          ENTRY
03
04          LDR   r0,=0x12345678
05          ADR   r1,Table
06          ADR   r2,Table1
07          LDR   r3, = 0xAAAAAAAA
08          LDR   r4,P3
09
10  Table   DCD   0xABCDDCBA
11  Table1  DCD   0xFFFFFFFF
12  P3      DCD   0x22222222
```

Note that there is a difference between LDR r4, = 0x1234 and LDR r4, = P3

The 1st one will save *0x1234* at the literal pool and then load the value of this address in R4.

The 2nd one will save the *ADDRESS* of P3 at the literal pool and then load the value of this address in R4.

In "ADR r4, P3", the distance between P3 and the ADR instruction MUST be represented as 0-255 and a rotation!!, while in "LDR r4, = P3", the distance Must be < 4096 (i.e., < 4K).

(1)
LDR r4, P3: copy the value in P3 and store it in r4
ADR r4, P3: address

(2)

(3) the result will be the same.
the difference will be in the way of implementation
ADR: make an addition by subtracting the address of P3 from (the address of current instruction - 8) and add it to the PC
LDR: get the address of P3 and put it in literal pool, and calculate the distence between current instruction and (the address of literal pool - 8)