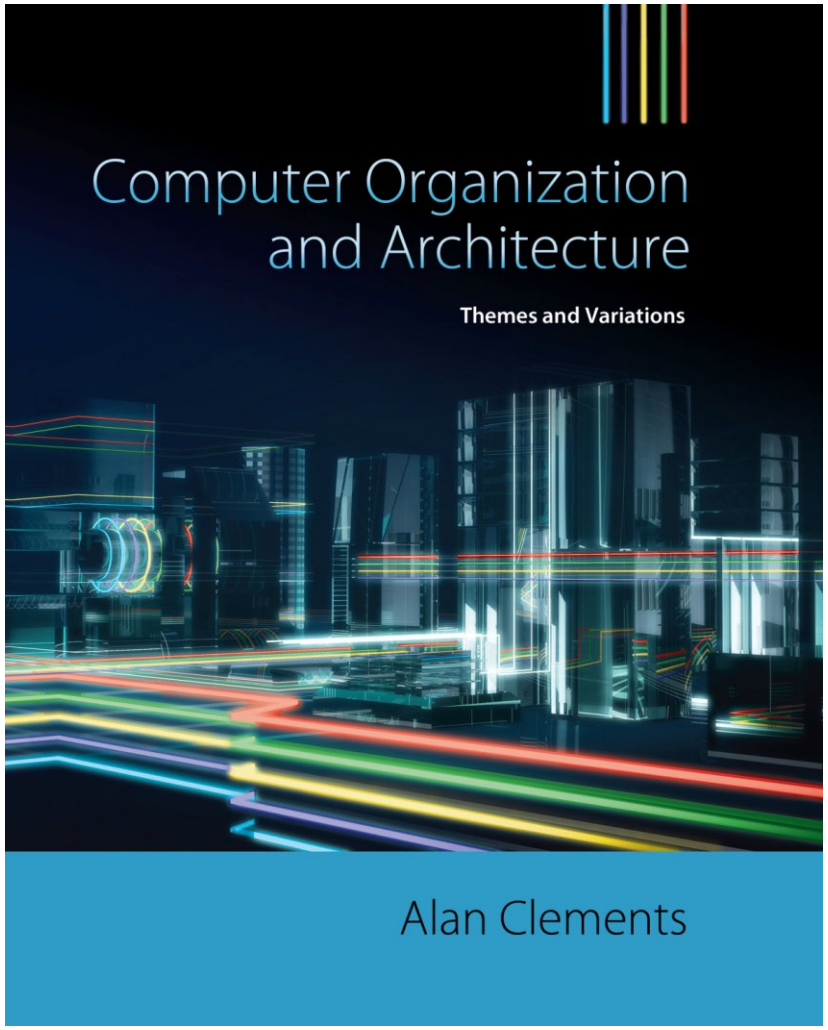
Part 4

Chapter 3

Architecture and Organization



CENGAGE Learning

MOV本身就是一条32bit的指令. #0x12345678这个数字本身长度为32. 总长过长.

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 #*** (#之即数 常量前缀)
      - 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

MOV used for only moving between register

It is NOT MOV r0, = 0x12345678 e.g. MOV r1, r2

之即数: 指令后的数据.

汇编: MOV AL R0 ← 之即数值.

机器: 0xE3004.

高字节0xE3 低字节0x04.

(操作码) (之即数值)

LDR r0, 0x12345678: 将地值对应值存入r0.

LDR r0, =0x12345678: 伪指令. 将这个地址本身存入r0.

MOV只能在寄存器间移动数据或将之即数移入寄存器.

LDR 伪指令类似 MOV. 但 MOV 限制立即数长度只能为 8位(2个数字), LDR 伪指令无限制. (但是条件允许情况下应被优先编译为 MOV

LDR R2, =0xFF0　　LDR R2, =0x12

MOV R2, #0xFF0

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

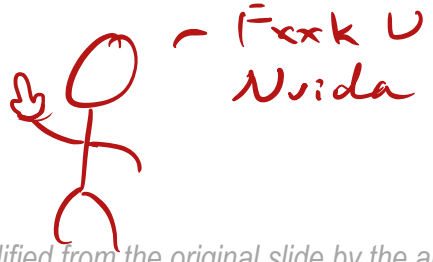
```
ADR   r1,MyArray  ;set up r1 to point to MyArray
                  ; loads register r1 with the 32-bit address of MyArray
...
LDR   r3,[r1]     ;read an element using the pointer

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*

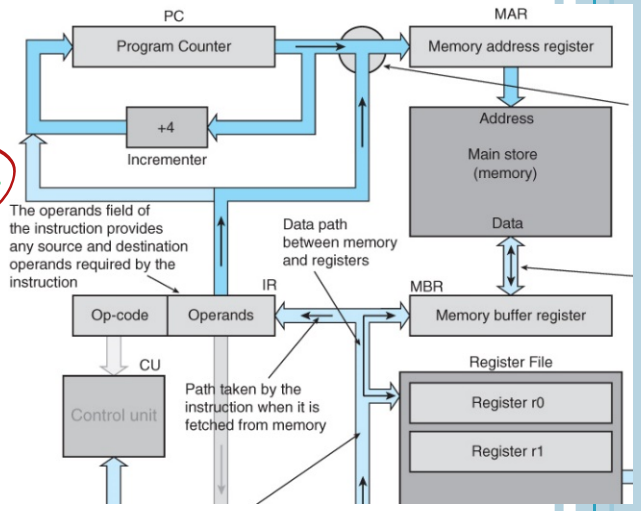


Program Counter Relative Addressing

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

- `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.

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



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

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

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

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

The ***LDR*** will load the value of P3 in r4. directly.
The ***ADR*** will utilize ADD/SUB to load the address of P3 in r4.

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

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 copy the ***ADDRESS*** of P3 at the literal pool and then load the value of this address in R4.

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

The 1st one will copy ***0x1234*** at the literal pool and then load the value of this address in R4.
The 2nd one will copy 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
```

Address.

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).

=> ADR utilise ADD/SUB.