Lec 09
Oct 14, 2022

Addressing Modes

There are 7 addressing modes available in ARM7.

i) Immediate ii) Register
iii) Direct iv) Indirect
v) Register Relative vi) Pre Index
vii) Base with scaled Index.

Addressing mode means the manner in which an operand is given in an instruction.
An operand is a value we want to operate.

① **Immediate**

MOV R0, #25H
ADD R0, R1, #25H

The data is given in the instruction.

MOV R0, #25H.
R0 register will get the data '25H'.
The number with "#" indicates data. The number without "#" indicates address.

In immediate addressing we can only put 8bit data in the instruction. If we want to use more bigger then 8bit number we have put the number in memory then use the number in the instruction.

Why we can not use ~~directly~~ bigger than 8bit number directly in the instruction?

Ans: Every instructions in ARM is 32 bit. In my instruction there are places for opcode, register, and data. If the data is 32 bit where is the place for opcode and register. Hence the operand size in immediate addressing mode is 8bit.

ADD R0, R1, #25H.

R0 register gets the value in R1 register addition with 25H.

② Registers

MOV R0, R1

ADD R0, R1, R2

The data (operand) will be given through a register.

③ Direct

12 bit offset

LDR R0, Amount

STR R0, Amount

In addressing mode address is given in the instruction.

The only two operations in memory that can happen in ARM7 is the "Load and Store".

~~With~~ Out of these two no operations occurs on memory.

~~The addr~~

LDR → Load operation (data comes memory to register)

STR → Store operation (data goes register to memory)

The address is ~~de~~ given in any variable. ex: (Amount)

LDR R0, Amount.

means the ~~gi~~ data comes from the location given in Amount, to R0 register.

The address is 32 bit.

But in instruction Amount gives 12 bit of ~~data~~ offset address.

Because the total instruction size 32 bit. So Amount can give the 32-bit address to the instruction.

$$2^{12} = 2^{2} \cdot 2^{10}$$
$$= 4KB$$

(short jump, medium and long jump)

So the maximum range of the address from the current location of the instruction is 4KB.

④ Indirect

LDR R0, [R1]
STR R0, [R1]

This is also called registers indirect.

The address of the memory is given in the registers.

LDR R0, [R1]

R0 registers will get the content from ~~fat~~ the location pointed by R1 registers.

As registers is 32 bit, so we can use full range of memory. which is 32 bit.

~~STR~~ STR R0, [R1]

The data in R0 will be stored at the location pointed by R1.

If we want to access the series of locations we use indirect addressing mode. we can increment the value in R1. So ~~the~~ then it will indicate the next location.

Using ~~Di~~ Indirect addressing mode we can access one or ~~two~~ location.

Using direct addressing mode we can ~~a~~ access a series of locations.

⑤ Reg Relative

Normal

LDR R0, [R1, #04H]

R0 registers will get the data from the location given in R1 plus displacement.

suppose location is 2000H.

the displacement is 04H.

so, ~~2~~ R1 is 2000H.

MOV R0, [R1, #04H]

Now R0 will get the data from the location 2004H.

R1 remains 2000H. after the operation. we will change the displacement.

Pre Index

LDR R0, [R1, #04H]!

suppose R1 contains 2000H.

By using this instruction first R1 is incremented by 04H and becomes 2004H.

Then R0 will get the data from the location. 2004H.

As, R1 is incremented first than pass the result to R0.

So it is called "pre index.

! → in programming this sign is called ~~bank~~ bank.

Anything in the '[ ]' square bracket is the address.

Post index

LDR R0, [R1], #04H

Suppose the location in R1 is 2000H. and the displacement is 04H.

R0 will get the data from the location 2000H.

first the value passes to R0 then R1 is incremented. Then R1 gets incremented to 2004H. means R1 becomes 2004H.

As R1 is incremented later so it is called post index.

In normal: R0 ← R1 + displacement

R1 remains same after the operation.

In pre index: ~~R1 ←~~ R1 + displacement

~~R0 ←~~ R0 ← R1.

* R1 gets changed ~~after the operation~~ before passing the value to R0.

In post index:

R0 ← R1

R1 ← R1 + displacement.

R1 gets changed after passing the value to R0.

0x4 are indicates 4 locations.

4 1 location 1 Bytes. = 8 bit

4 location 4 Bytes = 32 bit.

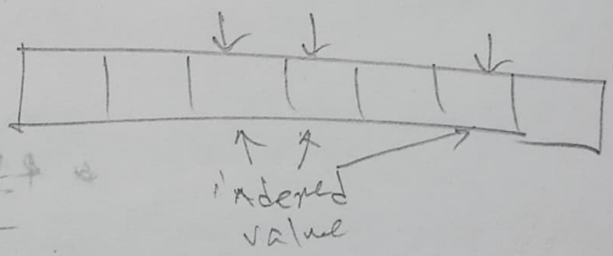
⑥ Base Indexed

Normal:

LDR R0, [R1, R2]

The address is given using the sum of two registers.

one register indicates the base, the other are indicates the index.



In array we can access several locations without changing the base value.

LDR R0, [R1, R2]

R1 contains base value.
R2 contains index value.

None of the values in R0 R1 an R2 will get changed.

<u>Pre-index</u>

LDR R0, [R1, R2]! # first R1 ← R1 + R2
second R0 ← R1.

<u>Post-index</u>

LDR R0, [R1], R2 # first R0 ← R1.
second R1 ← R1 + R2.

⑦ Base with Scaled Index

Normal

~~LDR, R~~

LDR R0, [R1, R2, LSL #4]

LSL ⇒ logically shift left.

$11 \xrightarrow{LS} 110$

$(3) \rightarrow (6)$

$1100 \xrightarrow{RS} 110$

$(12) \rightarrow (6)$

left shifting by 1 bit = multiplied by 2.

Right shifting by 1 bit = division by 2.

left shifting by 2 bit = multiplied by 2^2 (4)

LS by 3 bit = multiplied by 2^3 (8).

LS by 4 bit = ... " 2^4 (16).

scaled means upshifting.

Up scaling → left shifting

down scaling → right shifting

~~LDR R0, R1, R2, LSL #4H~~

LDR R0, [R1, R2, LSL #4H]

first the value in R1 will be shifted by 4 byte (16 bit)

second shifted value plus with value in R2.

~~R0~~ Now R0 will get the data from the new generated address (shifting + summation).

Suppose we want to access from location 1C59H.

MOV R1, # 1C59H

MOV R0, R1

This is an invalid instruction.

Because in immediate addressing mode Data can not be more than 8 bit.

Now what to do?

R1 = 1C
R2 = 59

```
MOV  R1, #1CH
MOV  R2, #59H.
LDR  R0, [R1, R2, #2]
```

LS by 2

first (R1) 1C → 1C00 → left shifting
(R2) (59) → 59

1C59

Now R0 will get the data from the location 1C59H.

R1 and R2 will not change after the operation.

Pre-index

LDR R0, [R1, R2, LSL #4]!

first R1 ← R1 (~~LS~~) + R2 (LS by 4)
second R0 ← R1.

Post index

LDR R0, [R1], LSL #4

first R0 ← R1
second R1 ← R1 (~~LS~~ with) + R2 (LS by 4)