Lec 11
Oct 14, 2022

ARM7 Instruction set Part II

(1) Data Xsfers instruction

- MOV R0, R1

* The values in R1 goes to R0. (It is not cut paste)
(It is copy paste method)
all 32 bit value in R1 goes to R0.

- MOV R0, #25H
- MOV R0, #0x25

} 8 bit immediate values

* It is a two way of writing of the same instruction.
* in imed immediate addressing mode we can only use 8-bit data.

- MOVS R0, R1
- MOVEQ R0, R1
- MOVEQS R0, R1

* MOV R0, R1
the data is transferred but flags are not affected.

MOVS R0, R1.
the data is transferred also affects the flags.

In other processors the flags are affected by the arithmatic and logic instructions.

But in ARM we can affect flags by any type of instruction.

MOVS R0, R1

if the value is negative in R1.
after this operation negative flag will be 1.

MOVEQ R0, R1.

if the result of the perious instruction is equal then it will operate this instruction, otherwise this instruction will be skiped.

Pipelining fails when jump instr. is executed.
when we use MOVEQ. we do not need to use jump. so it saves the failure of pipelining.

MOVCS R0, R1

if the previous instr. produces carry flg is 1. then this instr. will be occured. otherwise the instr. will be skiped.

MOVEQS R0, R1.

if the result of previous instr is equal the data will be transfered R1 to R0 and also affects the flags.

*) MVN R0, R1

MVN → move not. means it moves the complement of the current value.

Like CPL in 8051.

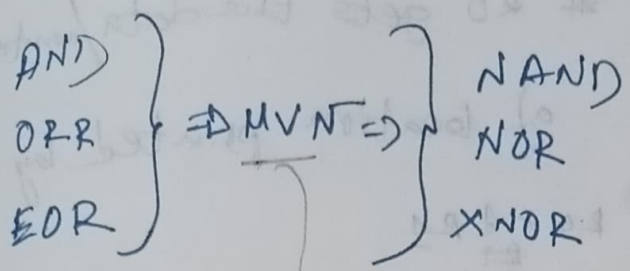
MVN R0, R1.

R0 ← R1.

we can also complement any specific register value.

MVN R0, R0

R0 ← $\overline{R0}$.



MVNS R0, R1.

the complement value of R1 goes to R0 and also affect the flags.

MVNHI R0, R1.

if the result of the previous instr is higher then pass the complement value to R0.

MVNHIS R0, R1

if the result of the previous instr. is higher than the pass the complement value (R1) to R0 and also affects the flag.

MVN can also be used as NOT gate.

(1) Load and Store Instruction

- LDR R0, [R1] → 32 bit

R0 gets the data/content of location pointed by R1.

R0 or R1 or R4

* R0 is 32 bit register

1 location 8 bit data

4 location 32 bit data.

R0 ← [R1] (1 to 8 bit)
[R1+1] (9 to 16 bit)
[R1+2] (17 to 24 bit)
[R1+3] (25 to 32 bit)

- LDRB R0, [R1] → 8 bit
- LDRH R0, [R1] → 16 bit

} unsigned

'B' stands for byte.
'H' stands for ~~word~~ half word.
word ⇒ 32 bit
half word ⇒ 16 bit

LDRB R0, [R1]

R0 ← [R1] (only one location)

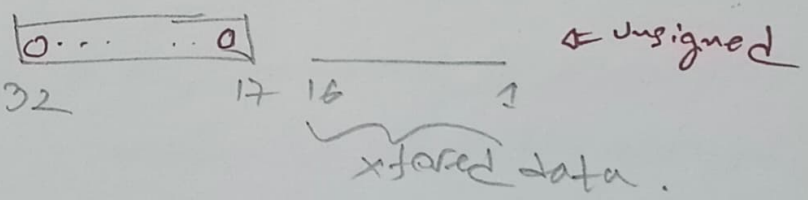
LDRH R0, [R1]

R0 ← [R1]

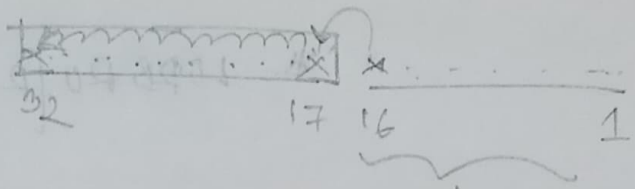
[R1+1] (two corresponding locations)

Sometimes we use 32 bit
" we " 16 bit
" " we 8 bit.

the R0 register is 32 bit.
wit we use 8/16 bit.
what will be the value in remaining bits in R0.



Ans: when we use unsigned data the remaining bits in R0 will be filled with zero.



=> signed number.

if it is signed number the remaining empty bits will be filled with MSB of the transferred data.

- LDRSB R0,[R1] → 8 bit
- LDRSH R0,[R1] → 16 bit

} signed

'SB' stands for sign byte.

'SH' stands for sign half words

- LDRT R0,[R1] → low privilege

'T' indicates the protection.

when data transfer occurs in user system level the transfering is allowed. But when the programm tries to access system level the transfering

operation is discarded.
and goes to next instruction.
So it is safe to use LDRT
rather than simple LDRT

- STR R0, [R1]
- STRB R0, [R1] } unsigned
- STRH R0, [R1] }
- STRSB R0, [R1] } signed
- STRSH R0, [R1] }

Store do the job in same way, but in reverse manner.
the value in R0 is stored to the location pointed by R1.