STR: store/write to memory instructions

STR - word STR - byte

STR R1,[R1,#4] [R1+4] ← R1

example loop in arm:

```
for(i=6; i!=0; i--);        MOV  R2, #6     R2 ← 6
                      LOOP: SUBS R2,#1*     R2 ← R2-1
                            BNE  LOOP
```

* immediate constant are limited in how big they can be (varies)

* SUB instruction does not affect the Z or any othe flags

SUBS ..., ... does affect ... and other flags

It's true for all arithmetic logic and MOV operator

ADDS, SUBS, ANDS, MOVS all cause the condition code flags to be set. But ADD, SUB, AND & MOV do not

Flags:

Z=1 if result =0

C=1 if result generated carry-out

N=1 ... ... is negative (2's compliment) (= bit 31 for a word → sign bit in 2's comp)

V=1 ... ... overflowed (too positive or too negative)

These condition code bits give rise to lots of branch conditions

BEQ	Z	
BNE	$\bar{Z}$	
BGT	$\bar{Z}$ + (NV + $\bar{N}\bar{V}$)	Branch if result is >0
BLT	N$\bar{V}$ + $\bar{N}$V	..., ..., ... <0
BGE	NV + $\bar{N}\bar{V}$	..., ..., ... ≥0
BLE	Z + (N$\bar{V}$ + $\bar{N}$V)	..., ..., ... ≤0

CMP: A compare instruction does a subtraction operation without changing a register, but it does change the condition codes

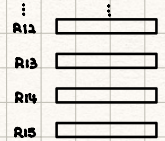
```
CMP  R0,R1   // computes R0-R1 & sets the condition codes (cc) appropriately
CMP  R0,#4   // R0-4 & sets on cc
```

example:

```
while (t!=-1){              topwhile: CMP  R0,#1    // Assume t=R0 computes t0-1 sets cc
    A=3;                          BNE  Done
    ⋮                             MOV  r1,#3
    t=x;                           ⋮
}                                  B   topwhile
                              done: ...
```

Review ARM Processor Registers

New one: Current Processor Status Register (CPSR)



16 x 32 bit registers

general purpose registers

SP Stack Pointer Register

LR Link Register

PC Program Counter

LR is used to know where to return to after a subroutine/function/procedure/method is done

example:

```
main(){                          // Assume 10 is Y
    int x,y,z;              0  MAIN: MOV R0, #10    // R0 ← 10
    y = 10;                 4        BL  MY_SUB     // LR: rm ← PC=NEXT=8
    x=my_sub(y);            8  NEXT: MOV ...           PC: rm ← MY_SUB
    ⋮                                ⋮
    z=my_sub(10);                    BL  MY_SUB     // LR: rm←PC=20   PC ← MY_SUB
}                           20  FRED: ...
                          MY_SUB: MOV R1,R0      // computes Y+Y
```

```
my_sub(int p){
    return  sp+p;
}
```

```
ADD  R1, R0
MOV  PC, LR      // PC ← LR + R1 + R0
```