Alternative Code

```
_start :   mov   R1, #LIST
1st element → LDR   R2, [R1]
2nd element → LDR   R3, [R1, #4]
              ADD   R2, R3

3rd  → LDR   R3, [R1, #8]
       ADD   R2, R3

4th  → LDR   R3, [R1, #12]
       ADD   R2, R3   //R2 ← sum.

END :  B     END

LIST : .word 10,20,30,40
```

Address	Machine Code	Assembly Code
0:	e3a01024	mov r1, #36 ; 0x24
4:	e5912000	ldr r2, [r1]
8:	e5913004	ldr r3, [r1, #4]
c:	e0822003	add r2, r2, r3
10:	e5913008	ldr r3, [r1, #8]
14:	e0822003	add r2, r2, r3
18:	e591300c	ldr r3, [r1, #12]
1c:	e0822003	add r2, r2, r3
20:	eafffffe	b 20 <END>
24:	0000000a	.word 0x0000000a
28:	00000014	.word 0x00000014
2c:	0000001e	.word 0x0000001e
30:	00000028	.word 0x00000028

- B means Branch. If no condition is given, then it defaultes to always. Equivalent to `mov pc, #END`

Alternative Code (using byte-size data)

```
_start :   mov   R1, #LIST
1st element → LDRB  R2, [R1]
2nd element → LDRB  R3, [R1, #1]
              ADD   R2, R3

3rd  → LDRB  R3, [R1, #2]
       ADD   R2, R3

4th  → LDRB  R3, [R1, #3]
       ADD   R2, R3   //R2 ← sum
```

Address	Machine Code	Assembly Code
0:	e3a01024	mov r1, #0x24
4:	e5d12000	ldrb r2, [r1]
8:	e5d13001	ldrb r3, [r1, #1]
c:	e0822003	add r2, r2, r3
10:	e5d13002	ldrb r3, [r1, #2]
14:	e0822003	add r2, r2, r3
18:	e5d13003	ldrb r3, [r1, #3]
1c:	e0822003	add r2, r2, r3
20:	eafffffe	b 20 <END>
24:	281e140a	.word 0x281e140a

4 bytes

```
END:  B    END

.byte  10,20,30,40
```

Store Instruction

STR word

STRB byte

STR R2, [R1, #4] – store R2 into memory @ addr. given by R1 + 4

* Addressing mode.

The same addressing modes available for LDR can also be used with STR

Example of a loop:

```
      ⋮
      MOV   R2, #6
LOOP: SUBS* R2, #1*
      BNE*  LOOP
END:  B     END
```

* BNE means Branch if NOT EQUAL TO ZERO. This is exactly same as MVNZ in your processor.

* for ARM, any data processing instruction can use a #D constant argument. The size of D is restricted.

* SUB is the subtraction instruction. The S suffix means set the condition code flags according to the result.

Condition code flags.

Z : result is zero

C : result produce a carry out

N : result is negative (bit 31 = 1 "MSB")

V : result is overflowed. for signed number.

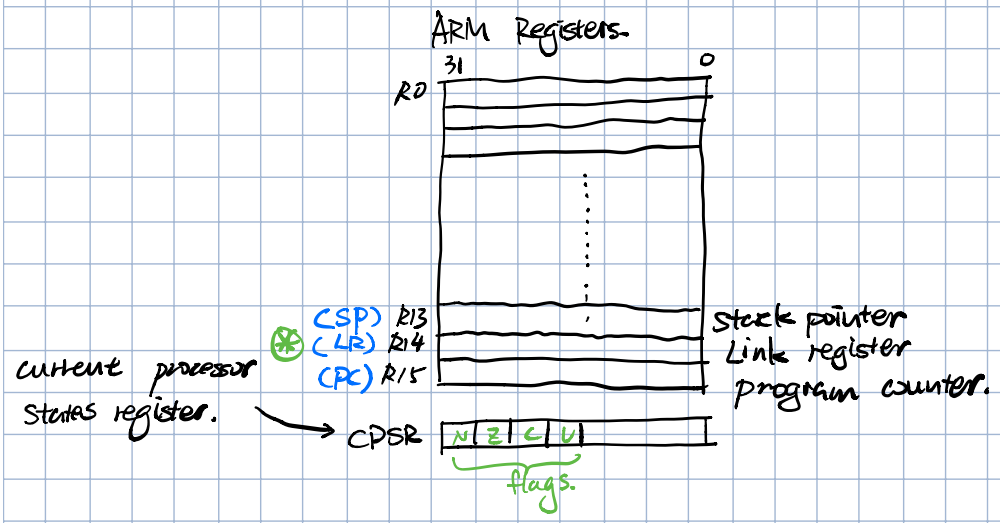
- these flags accommodate many conditions. For example, using branch

BNE	$\bar{Z}$	cond. = 0001 (in machine code)
BEQ	Z	cond = 0000
BGT	$\bar{Z}$ +	cond = 1100
BLT	$N \oplus V$	cond = 1011
BGE	$N \odot V$	cond = 1010
BLE	$Z + (N\bar{V} + \bar{N}V)$	cond = 1101

- you can set the condition code flags using any data processing instruction with S appended ADDS, SUBS, MOVS, ANDS, ORRS, ...
There are also some special comparison instruction whose only result is to affect the flags.

```
cmp  R0, R1   //performs R0-R1 to set flags
cmp  R5, #2   //performs R5 - 2 to set.
```

More about ARM Registers.



* R14 (aka LR) is used to return from a subroutine

```
main(){
    int x, y;
    y= 10;
    x= my_sub(y);
    ⋮
mysub (int x) {
    return (x+x);
}
```

```
MAIN:   mov   R0, #10  //y
        BL*   my_sub
NEXT:   - - - - -
        ⋮
//parameter is in R0
//Result is returned in R0
my_sub: ADD   R0, R0
        mov   PC, LR
```

* Branch & link instruction copies the addr. of the next instruction into R14 (LR), and then set PC (R15) to addr. of subroutine.