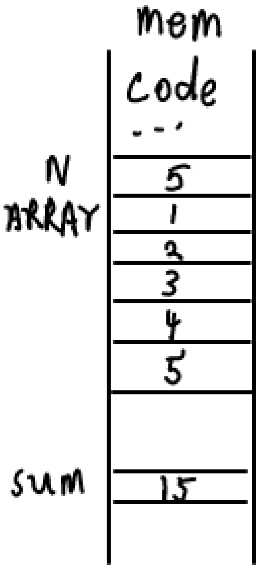


5.3) Example

- pc-relative: LDR{size}{cond} Rt, label
  - label is translated into an offset from an instruction.
  - data is loaded from <address> = pc+4+offset
  - offset ∈ [-4095, +4096]

e.g. LDR r1, DATA1; DATA1 is 32 bytes after LDR

becomes

LDR r1, [pc, #28] ← pc incremented by 4 during fetch.

//r1 ← [[pc]+28]

- load address into register: ADR{cond} Rd, label

e.g. ADR r1, DATA1

//r1 ← [pc]+28

5.4) Pseudo-Instructions
- don't match an existing machine instruction
- translated by the assembler into an opposite instruction(s)

e.g. LDR{cond} Rt, =<expr>

↳ label or numeric instruction

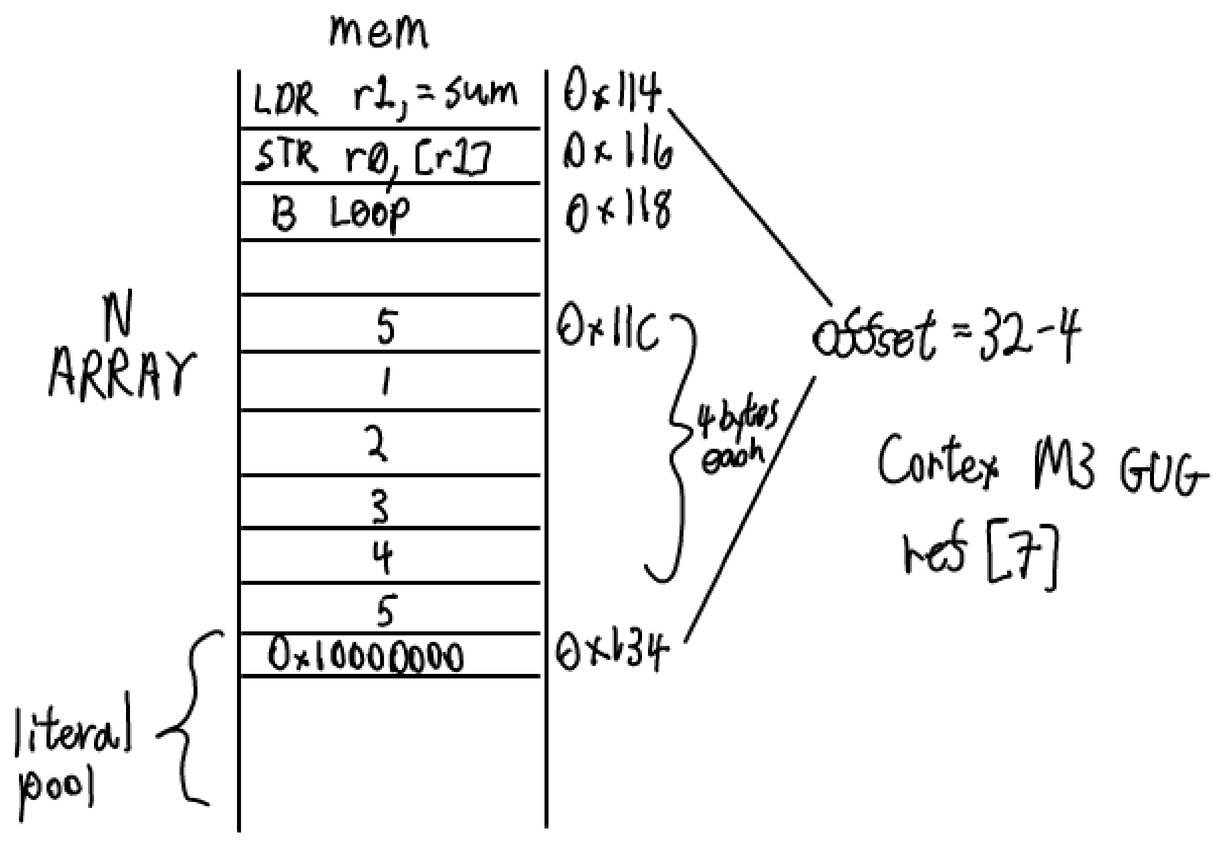
- converted into:
  a) MOV or MVN if possible, or
  b) adds the value to the literal pool (↑ program constants) and generates a LDR instr. with pc-relative addressing.

from demo0.s

LDR r1, =SUM - couldn't use ADR cause the distance from LDR (0x114) to SUM (0x10000000) exceeds ±4095.

- Stores 0x10000000 at address 0x134 and replaces it with

LDR r1, [pc, #28]



Cortex M3 GUG ref [7]

5.5) Branch Instruction

- changes control flow by adding an offset to the PC
- format: B{cond} label

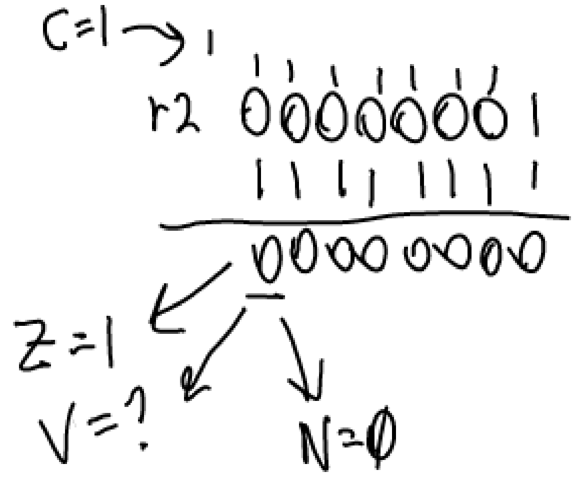
label ← Assembler replaces with pc-relative offset.

{cond} ← only execute if condition true.

- condition code suffixes:

			signed		unsigned
EQ	equal (to zero)	GT	greater than	⟷	HI
NE	not equal	GE	greater or equal	⟷	HS
VS	overflow set	LT	less than	⟷	LO
VC	overflow clear	LE	less or equal	⟷	LS
AL	always (default)	PL	plus (≥0) } ignores overflow		
		MI	minus (<0) } ignores overflow		

```
e.g. SUBS  r2, r2, #1   // r2 ← [r2] - 1
     BGT   LOOP                ⇓
                         N, Z, C, V flags
```



ARM

5.6) Pre- and Post- indexed addressing
- applies to LDR & STR
- pre-indexed: `<op>{size}{cond} Rt, [Rn, #offset]` $\boxed{!}$
  - the offset is added to the address in Rn, then the memory access is performed and Rn is updated

```
e.g. LDR r1, [r0, #4]!
     // r1 ← [[r0]+4], r0 ← [r0]+4
```

- post-indexed: `<op>{size}{cond} Rt, [Rn], #offset`
  - the memory access is performed with the address in Rn, then Rn is updated by adding the offset.

```
e.g. LDR r1, [r0], #4
     // r1 ← [[r0]], r0 ← [r0]+4
```

5.7) Compare Instructions

- compare two operands and sets the condition flags (N, Z, C, V) but does not save to a destination register.

- CMP{cond} Rn, Operand2

  e.g. CMP r1, #1

  // N, Z, C, V $\Leftarrow$ [r1] − 1

  if r1 = 1, N ← 0, Z ← 1, C ← 1, V ← 0

  but r1 is not changed.

- CMN{cond} Rn, Operand2 //compare negative
  - compares [Rn] and −Operand2
- Test: TST{cond} Rn, Operand2
  - performs bit-wise AND and updates flags N, Z, C, V

  e.g. TST r1, #0x00008000 ⇐ mask

  // N, Z, C, V ⇐ [r1]·2_0000 0000 0000 0000 1000 0000 0000 0000

- tst equal TEQ{cond} Rn, Operand2
  - performs bit-wise EOR and updates flags

- compare and branch: ⟨op⟩ Rn, label

  ⟨op⟩: CBZ compare equal to zero

  CBNZ compare and not equal to zero

- compares [Rn] with zero and decides on branch.

CBZ Rn, label = $\begin{cases} \text{CMP Rn, \#0} \\ \text{BEQ label} \end{cases}$

CBNZ Rn, label = $\begin{cases} \text{CMP Rn, \#0} \\ \text{BNE label} \end{cases}$

- does not set the condition flags N, Z, C, V

5.8) If - Else

```
e.g. if(X==0)   //cond
        Y++;    // stmt 1
     else
        Y--;    // stmt 2
     X in r0, Y is r1
```

with branch:

```
        CBZ  R0, ADD_       //cond
        SUB  R1, R1, #1     //stmt 2
          B  END_
ADD_    ADD  R1, R1, #1     //stmt 1
END_    -----
```

without branch:

```
CMP r0, #0
ADDEQ r1, r1, #1
SUBNE r1, r1, #1
```