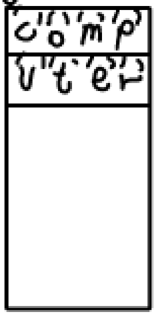
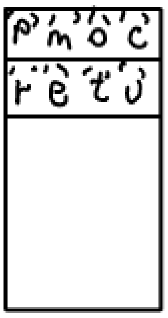
Big Endian



Little Endian



✱ Aside

ADD R2, R0, $\underline{\#1}$

ADD $\underset{Rd}{R2}$, $\underset{R_n}{R0}$, $\underbrace{R_1, LSL \#2}_{Op2}$

ARM

5.1) D.P. Instructions

...

- data movement

MOV{S}{cond} Rd, Op2

MOV{cond} Rd, #imm16 ← 0-extended to 32 bits.

MVN{S}{cond} Rd, Op2 move not

MVT{cond} Rd, #imm16 writes #imm16 to Rd[31:16] and doesn't change Rd[15:0]

e.g. move 0xabcd1234 into r0

MOV r0, #0x1234 r0 [0 0 0 0 | 1 2 3 4]

↓

MOVT r0, #0xabcd [a b c d | 1 2 3 4]

e.g. MVN r0, #1 ← 0000....0001

1111 ---- 1110

r0 ← NOT 1

= r0 ← 0xfffffffe

5.1) D.P. Instructions (cont.)

- shift

  format <op>{S}{cond} Rd, Rm, <Rs|#n>

  ASR, LSL, LSR, ROR, RRX

  e.g. LSL r1, r0, r2

  contain n

  // r1 ← [r0] << n

  [r2]

5.2) Memory Access Instructions (lab manual ref [7])

- format:

  <op>{size}{cond} Rd, <address>

  register indirect addressing

  destination reg (loads) source reg (stores)

  only executes if true

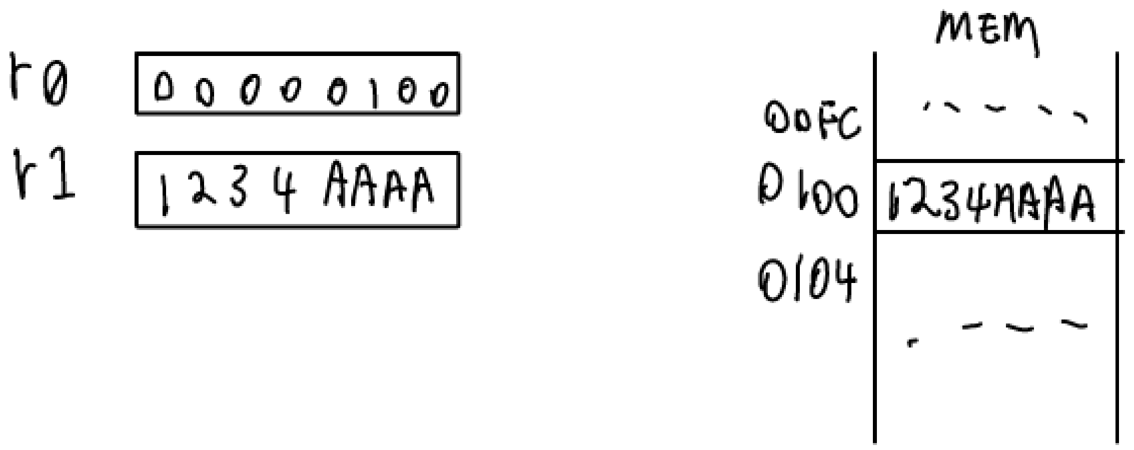
  - B byte } zero-extended to
  - H halfword } 32-bits on load
  - SB signed byte } sign
  - SH signed halfword } extended

  LDR (load register) or STR (store register)

- with immediate offset: <address> = [Rn {, #offset}]

  e.g. LDR r1, [r0]

  // r1 ← [[r0] + 0] ← from memory



e.g. LDR r1, [r0, #4]

// r1 ← [[r1]+4]

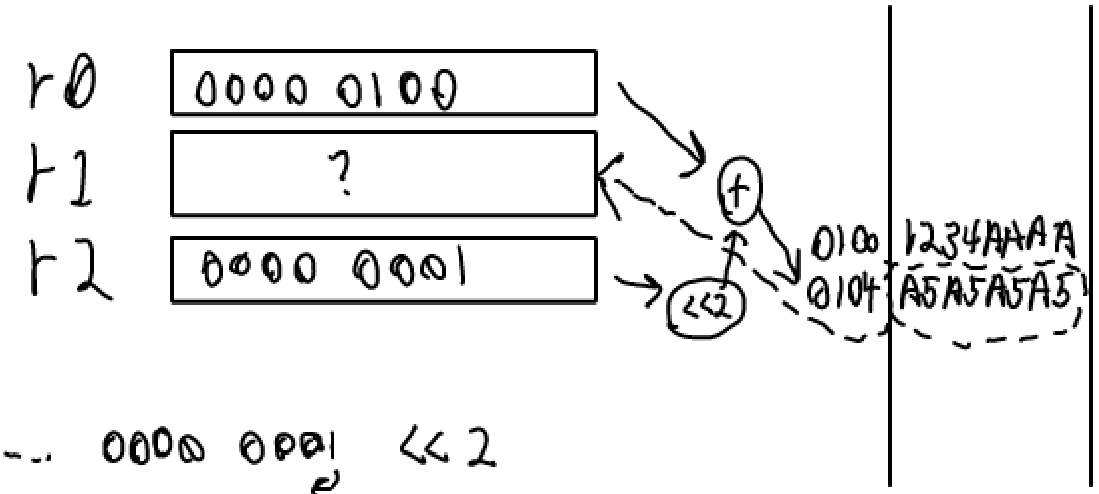
- with register offset: <address> = [Rn, Rn {, LSL #n}]

e.g. LDR r1, [r0, -r2]

// r1 ← [[r0] - [r2]]

LDR r1, [r0, r2, LSL #2]

// r1 ← [[r0] + [r2] << 2]

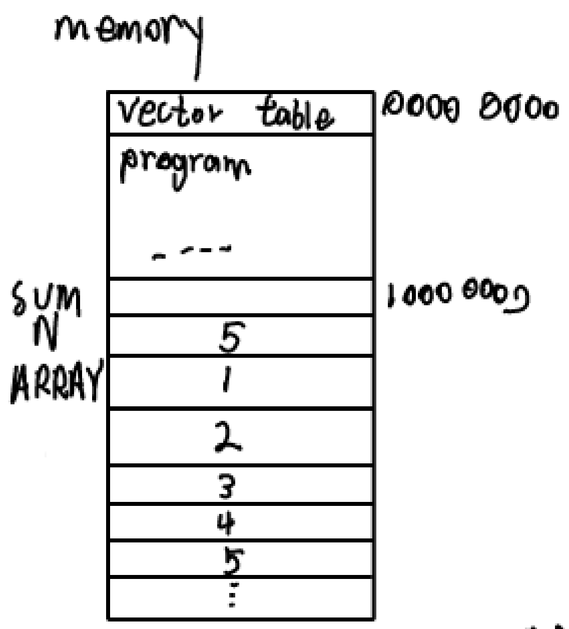


---. 0000 0001 << 2

= 0000 0100

5.3) Example : sum array

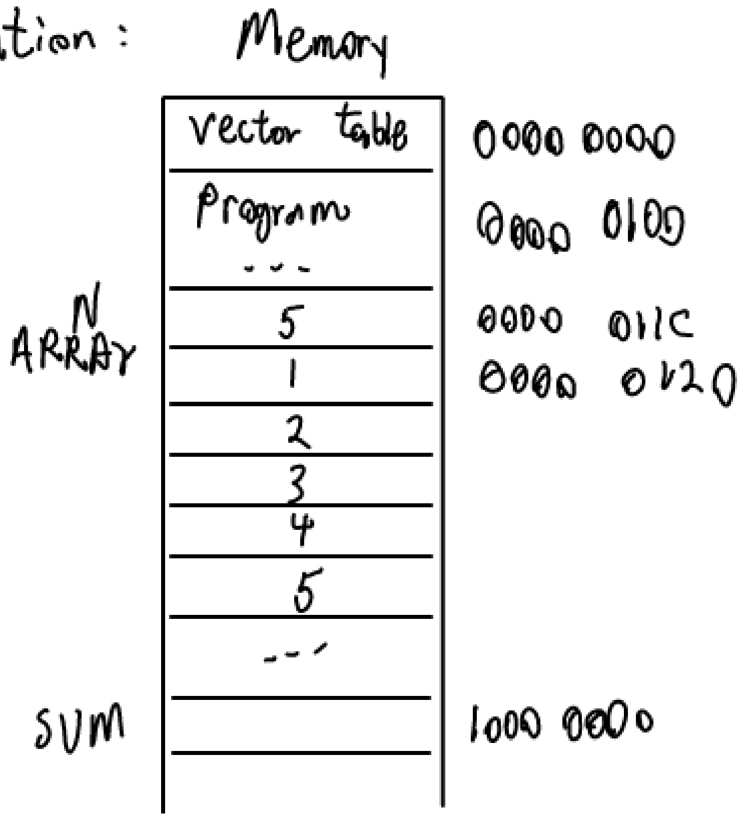
ideal picture:



Problem:
loader doesn't initialize the data segment.

N is the array length

Solution:



Memory
vector table
0000 0000
Program
0000 0100

N
5
0000 011C
ARRAY
1
0000 0120
2
3
4
5

SUM
1000 0000