Tutorial 5

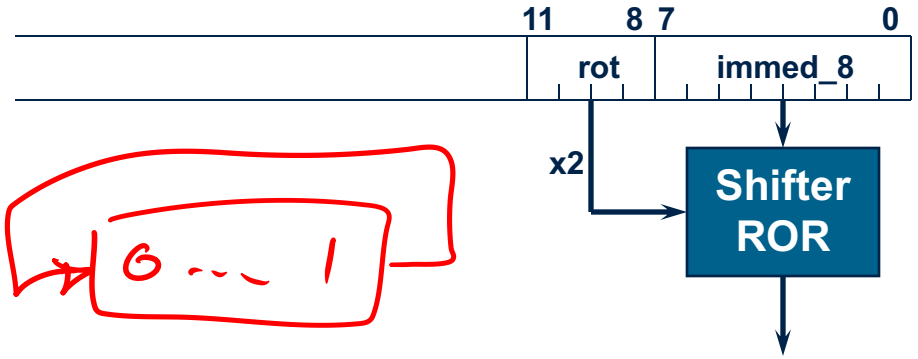
1. What constant would be loaded into register r7 by the following instructions?

   a) MOV r7, #0x8C, 4

   b) MVN r7, #2

Immediate constants (1)

- **No ARM instruction can contain a 32 bit immediate constant**
  - All ARM instructions are fixed as 32 bits long
- **The data processing instruction format has 12 bits available for operand2**



- **4 bit rotate value (0-15) is multiplied by two to give range 0-30 in steps of 2**
- **Rule to remember is "8-bits shifted by an even number of bit positions".**

Loading 32 bit constants

- **To allow larger constants to be loaded, the assembler offers a pseudo-instruction:**
  - `LDR rd, =const`
- **This will either:**
  - Produce a **`MOV`** or **`MVN`** instruction to generate the value (if possible).

  or
  - Generate a **`LDR`** instruction with a PC-relative address to read the constant from a *literal pool* (Constant data area embedded in the code).
- **For example**
  - `LDR r0,=0xFF` => `MOV r0,#0xFF`
  - `LDR r0,=0x55555555` => `LDR r0,[PC,#Imm12]`

```
...
...
DCD 0x55555555
```

- **This is the recommended way of loading constants into a register**

1. It may be easier to do the rotation in binary if the shift is not multiples of 4.

   a) MOV r7, #0x8C, 4

   8 C

   0x8C = 1000 1100B (B : Binary)

   Rotate Right by 4 bits becomes

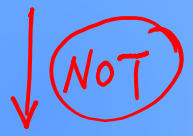
   1100 0000 0000 0000 0000 0000 0000 **1000**B

   r7 = 0xC0000008

b). MVN r7, #2

2 = 10B

= 0000 0000 0000 0000 0000 0000 0000 0010



Negate 10B =

1111 1111 1111 1111 1111 1111 1111 1101B

r7 = 0xFFFFFFFD

2. Whenever possible, use the byte rotation scheme described in the lecture notes for the MOV instruction to generate an instruction needed to load the following constants into register r2. If it is not possible to use a MOV instruction, provide an alternative instruction that does the job.

   a) 0xA400

   b) 0x7D8

2. Answer may not be unique !!!

a) 0xA400

By inspection, it is clear that **0xA4** can fit into a byte and the number 0xA400 constructed by shifting **0xA4** by 8 bits to the left.

(or 32-8 =24 bits to the right.)

Answer: **MOV r2, #0xA4, 24**

Alternative solution

0xA400 = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 B

The number can also be expressed as 1 0 1 0 0 1 B (ie. 0x29) shifted 10 bits to the left,

(or 22 bits to the right.)

Answer: **MOV r2, #0x29, 22**

b) 0x7D8

0x7D8 = 1 1 1 1 1 1 0 1 1 0 0 0 B

Note that the underlined portion 1 1 1 1 1 0 1 1 can fit into 8 bits but this requires a **shift left of 3 bits.**

The amount of shift must be an even number. Hence it is not possible to represent it by the MOV instruction.

Instead must use **LDR r2, =0x7D8**

3. Without using the MUL instruction, give instructions that multiply register r4 by :

   a) 135

   b) 255

and place your result in register r0.
Your solution for each part should only have two or less instructions.

3. Try to find the multipliers closest to the numbers that are to the powers of 2

a) Multiply by 135 = 128 + (8 - 1)

The closest is multiply by 128 or 2^7 then add 7 times of it

Multiply by 7 can be constructed from multiply by 8 then subtract 1.

r4 * 8

```
RSB     r1, r4, r4, LSL #3   ; r1 = r4*8 – r4 = 7*r4
ADD     r0, r1, r4, LSL #7   ; r0 = r1 + r4*128
```

r4 * 128

SUB r1, r2, r3 ; r1 ← r2 - r3

RSB r1, r2, r3 ; r1 ← r3 - r2

255 = 256 - 1

b) Multiply by 255

Multiply by 255 can be perform by multiplying by 256 then subtract 1

RSB r0, r4, r4, LSL #8 ; r0 = r4*256 – r4 = 255*r4

r4 * 256