Quiz #2 - Sample Solutions

"Normal" Assembler

```
gcd    cmp r0, r1      ;reached the end?
       beq stop
       blt less        ;if r0 > r1
       sub r0, r0, r1  ;subtract r1 from r0
       bal gcd
less   sub r1, r1, r0  ;subtract r0 from r1
       bal gcd
stop
```

ARM Conditional Assembler

```
gcd    cmp   r0, r1      ;if r0 > r1
       subgt r0, r0, r1  ;subtract r1 from r0
       sublt r1, r1, r0  ;else subtract r0 from r1
       bne   gcd         ;reached the end?
```

The Barrel Shifter

* The ARM doesn't have actual shift instructions.

* Instead it has a barrel shifter which provides a mechanism to carry out shifts as part of other instructions.

* So what operations does the barrel shifter support?

Barrel Shifter - Left Shift

* **Shifts left by the specified amount (multiplies by powers of two) e.g.**

  LSL #5 = multiply by 32

Logical Shift Left (LSL)

Barrel Shifter - Right Shifts

Logical Shift Right

- Shifts right by the specified amount (divides by powers of two) e.g.

  LSR #5 = divide by 32

Logical Shift Right

Arithmetic Shift Right

- Shifts right (divides by powers of two) and preserves the sign bit, for 2's complement operations. e.g.

  ASR #5 = divide by 32

Arithmetic Shift Right

Sign bit shifted in

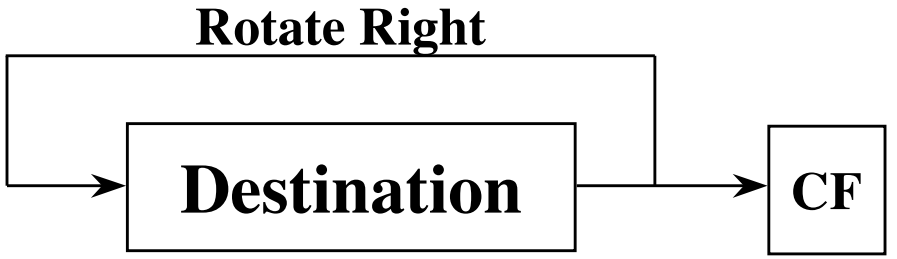
Barrel Shifter - Rotations

Rotate Right (ROR)

- Similar to an ASR but the bits wrap around as they leave the LSB and appear as the MSB.

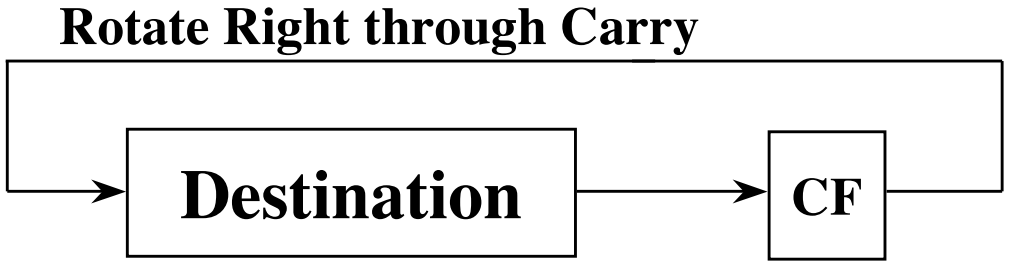
  e.g. ROR #5

- Note the last bit rotated is also used as the Carry Out.

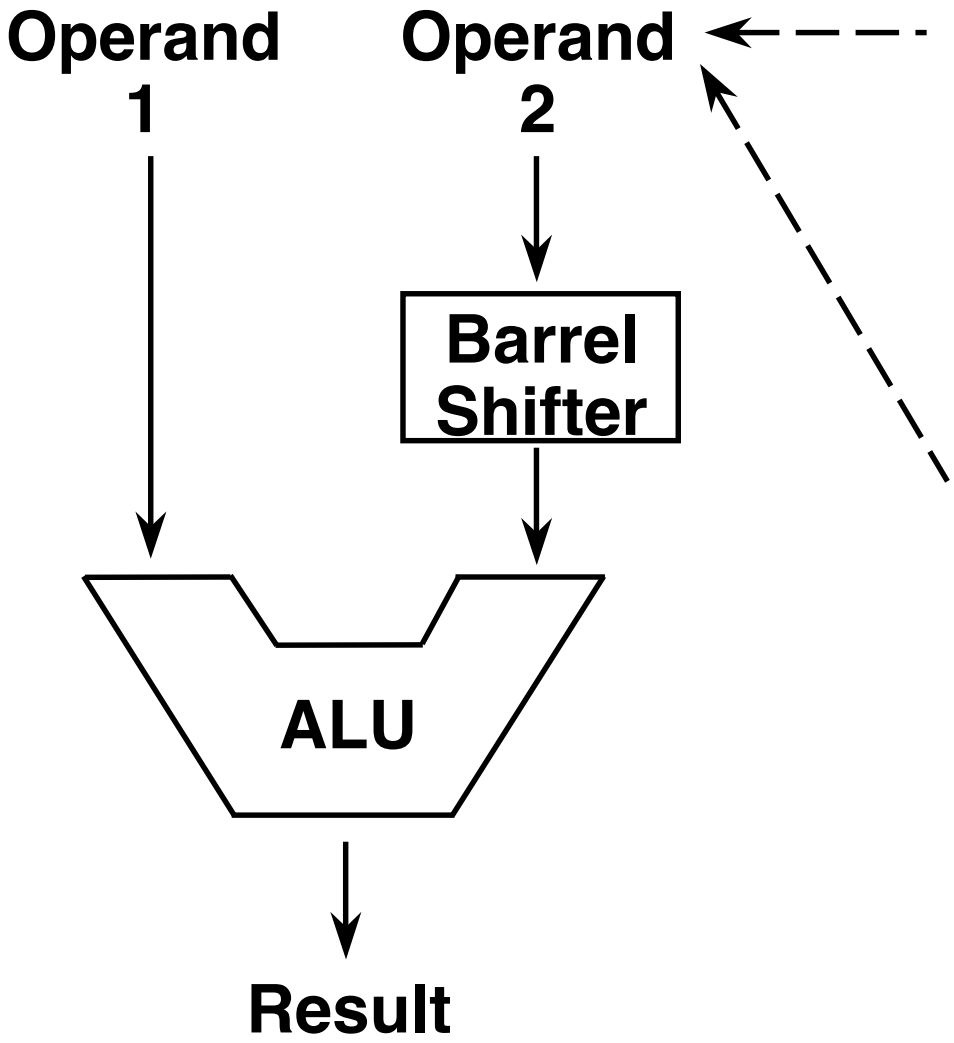


Rotate Right Extended (RRX)

- This operation uses the CPSR C flag as a 33rd bit.
- Rotates right by 1 bit. Encoded as ROR #0.



Using the Barrel Shifter: The Second Operand



* **Register, optionally with shift operation applied.**
* **Shift value can be either be:**
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.

* **Immediate value**
  - 8 bit number
  - Can be rotated right through an even number of positions.
  - Assembler will calculate rotate for you from constant.

Second Operand : Shifted Register

* **The amount by which the register is to be shifted is contained in either:**
  - the immediate 5-bit field in the instruction
    - *NO OVERHEAD*
    - Shift is done for free - executes in single cycle.
  - the bottom byte of a register (not PC)
    - Then takes extra cycle to execute
    - ARM doesn't have enough read ports to read 3 registers at once.
    - Then same as on other processors where shift is separate instruction.
* **If no shift is specified then a default shift is applied: LSL #0**
  - i.e. barrel shifter has no effect on value in register.

Second Operand : Using a Shifted Register

* **Using a multiplication instruction to multiply by a constant means first loading the constant into a register and then waiting a number of internal cycles for the instruction to complete.**
* **A more optimum solution can often be found by using some combination of MOVs, ADDs, SUBs and RSBs with shifts.**
  - Multiplications by a constant equal to a ((power of 2) ± 1) can be done in one cycle.
* **Example: r0 = r1 * 5**
  Example: r0 = r1 + (r1 * 4)

```
ï  ADD r0, r1, r1, LSL #2
```

* **Example: r2 = r3 * 105**
  Example: r2 = r3 * 15 * 7
  Example: r2 = r3 * (16 - 1) * (8 - 1)

```
ï  RSB r2, r3, r3, LSL #4     ; r2 = r3 * 15
ï  RSB r2, r2, r2, LSL #3     ; r2 = r2 * 7
```

Second Operand : Immediate Value (1)

* **There is no single instruction which will load a 32 bit immediate constant into a register without performing a data load from memory.**
  - All ARM instructions are 32 bits long
  - ARM instructions do not use the instruction stream as data.
* **The data processing instruction format has 12 bits available for operand2**
  - If used directly this would only give a range of 4096.
* **Instead it is used to store 8 bit constants, giving a range of 0 - 255.**
* **These 8 bits can then be rotated right through an even number of positions (ie RORs by 0, 2, 4,..30).**
  - This gives a much larger range of constants that can be directly loaded, though some constants will still need to be loaded from memory.

Second Operand : Immediate Value (2)

* **This gives us:**
  - 0 - 255 [0 - 0xff]
  - 256,260,264,..,1020 [0x100-0x3fc, step 4, 0x40-0xff ror 30]
  - 1024,1040,1056,..,4080 [0x400-0xff0, step 16, 0x40-0xff ror 28]
  - 4096,4160, 4224,..,16320 [0x1000-0x3fc0, step 64, 0x40-0xff ror 26]
* **These can be loaded using, for example:**
  - MOV r0, #0x40, 26 ; => MOV r0, #0x1000 (ie 4096)
* **To make this easier, the assembler will convert to this form for us if simply given the required constant:**
  - MOV r0, #4096 ; => MOV r0, #0x1000 (ie 0x40 ror 26)
* **The bitwise complements can also be formed using MVN:**
  - MOV r0, #0xFFFFFFFF ; assembles to MVN r0, #0
* **If the required constant cannot be generated, an error will be reported.**