SPRING 2016
CMPE 364

Microprocessor Based Design

Dr. Ryan Riley
(Slides adapted from Dr. Mohamed Al-Meer)

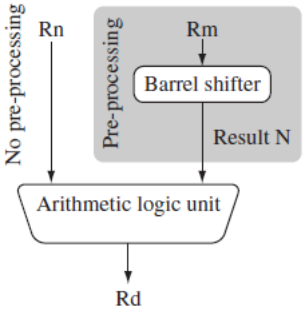
Barrel Shifter

- A unique and powerful feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before it enters the ALU.
- The argument could be a constant or a register that cab be processed by a shifter before it can affected by any instruction (MOV).
- See next Figure to see how it works.
- **Pre-processing** or **shift** occurs within the cycle time of the instruction
- useful for **loading constants** into a register and achieving **fast multiplies** or **division** by a power of **2**

Barrel Shifter

• Example 3

Assume R5 = 5, R7 = 8. what is the value R7 after the execution of next instruction?



MOV R7, R5, LSL #2 ; let R7 = R5*4 = (R5 << 2)

R5 = 5

R7 = 20

Barrel Shifter

• The five different shift operations that you can use within the barrel shifter are summarized in next Table.

Table 3.2 Barrel shifter operations.

Mnemonic	Description	Shift	Result	Shift amount y
LSL	logical shift left	xLSL y	$x \ll y$	#0–31 or *Rs*
LSR	logical shift right	xLSR y	(unsigned)$x \gg y$	#1–32 or *Rs*
ASR	arithmetic right shift	xASR y	(signed)$x \gg y$	#1–32 or *Rs*
ROR	rotate right	xROR y	((unsigned)$x \gg y$) | ($x \ll (32 - y)$)	#1–31 or *Rs*
RRX	rotate right extended	xRRX	(c flag $\ll 31$) | ((unsigned)$x \gg 1$)	none

Note: x represents the register being shifted and y represents the shift amount.

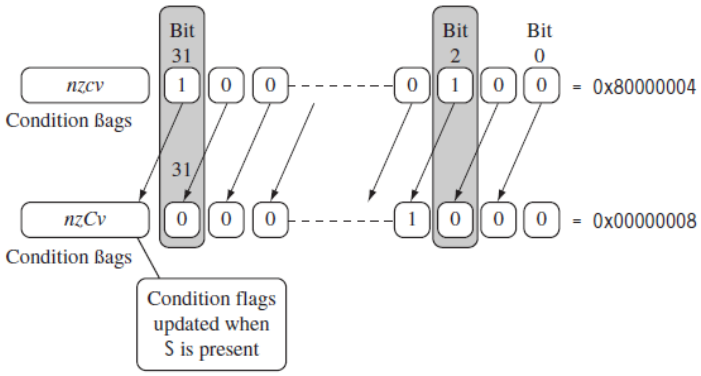
Shift and Rotate with ARM processor

N shift operations	Syntax
Immediate	`#immediate`
Register	`Rm`
Logical shift left by immediate	`Rm, LSL #shift_imm`
Logical shift left by register	`Rm, LSL Rs`
Logical shift right by immediate	`Rm, LSR #shift_imm`
Logical shift right with register	`Rm, LSR Rs`
Arithmetic shift right by immediate	`Rm, ASR #shift_imm`
Arithmetic shift right by register	`Rm, ASR Rs`
Rotate right by immediate	`Rm, ROR #shift_imm`
Rotate right by register	`Rm, ROR Rs`
Rotate right with extend	`Rm, RRX`

Barrel shift Operations for Data Processing Instructions

Barrel Shifter

- Next Figure illustrates a logical shift left by one.
- For example, the contents of bit 0 are shifted to bit 1. Bit 0 is cleared. The *C* flag is updated with the last bit shifted out of the register



Barrel Shifter, Another Example with Status

This example of a MOVS instruction shifts register *r1* left by one bit. This multiplies register *r1* by a value 2^1. As you can see, the *C* flag is updated in the *cpsr* because the S suffix is present in the instruction mnemonic.

```
PRE     cpsr = nzcvqiFt_USER
        r0 = 0x00000000
        r1 = 0x80000004

        MOVS    r0, r1, LSL #1

POST    cpsr = nzCvqiFt_USER
        r0 = 0x00000008
        r1 = 0x80000004
```

Using the Barrel Shifter with Arithmetic Instructions

- As known The wide range of second operand shifts is very powerful feature of the ARM instruction set.
- The next example is a use of the inline barrel shifter with an arithmetic instruction.
- All conditions regarding barrel shifter implemented on MOV instruction applies here as well.

Example

Register *r1* is first shifted one location to the left to give the value of twice *r1*. The ADD instruction then adds the result of the barrel shift operation to register *r1*. The final result transferred into register *r0* is equal to three times the value stored in register *r1*.

```
PRE     r0 = 0x00000000
        r1 = 0x00000005

        ADD     r0, r1, r1, LSL #1

POST    r0 = 0x0000000f
        r1 = 0x00000005
```

Shift & Rotation

- Moves bits to left or right in a register

Logical Shifting

- Shifting in 0's
- Logical Left shift and Logical Right shift

- Logical Shift Left
  - All bits shifted to left
  - 0 is fed to b0
  - **B31 will be copied to C flag**.

Logical Shift Left

- Example

  Shift the 32-bit value 0x89C21E46 **5 positions to left**?

- Solution

Before 1000 1001 1100 0010 0001 1110 0100 0110

After 0011 1000 0100 0011 1100 1000 110**0 0000**

0x3843C4C0

Logical Shifting a number to left 1 position is equivalent to multiply by 2.

Logical Shift Left

- EXAMPLE: show logical left shift of 180,156 six positions is equal to multiply by 64?
- ANSWER:

180,156 = 000000000000001010111111110111100 x 64 =

= 00000000101011111110111100000000

= 11,529,984

LSL Logical Shift Left

- rm , LSL rs
- rm, LSL # immediate

- **Example**
- **MOV R8, R7, LSL R10**
  - R7 = 0x008162CA = 0000 0000 1000 0001 0110 0010 1100 1010
  - R10 = 0x16
- Solution
  - R8 = 0xB2800000 = 1011 0010 1000 0000 0000 0000 0000 0000

LSL Logical Shift Left

- **Example**
  - **ADD R0, R2, R3, LSL #6**
  - R2 = 00010049, R3 = **00000531**
- Solution
  - 0000 0000 0000 0001 0000 0000 0100 1001 +
  - **0000 0000 0000 0001 0100 1100 0100** 0000
  - ---
  - 0000 0000 0000 0010 0100 1100 1000 1001 (00024C89)

Logical Shift Right

- Shift all bits to the right.
  - All bits shifted to right
  - 0 is fed to b31
  - **b0 will be copied to C flag.**
- EXAMPLE: logically shift 32-bit the value 0x89C21E46 fourteen positions?
- SOLUTION:

  1000 1001 1100 0010 0001 1110 0100 0110

  0000 0000 0000 0010 0010 0111 0000 1000

Logical Shift Right

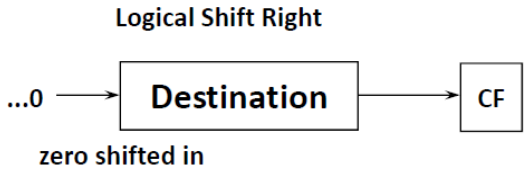
- Logical Shifting a number to right1 position is equivalent to divide by 2.
- EXAMPLE: use logical shift right to divide unsigned number (0x00008105) BY 256?
- SOLUTION:

  0000 0000 0000 0000 1000 0001 0000 0101

  0000 0000 0000 0000 0000 0000 1000 0001

LSR Logical Shift Right

- rm, LSR rs
- rm, LSR #immediat
- EXAMPLE
  - **RSB R6 R5, R9, LSR #1**
  - R5 = 00000052, R9 = 000001AC
- Solution
  - (R9 >> 1) – R5
  - (R9 >> 1) = 000000D6
  - Result = 000000D6 – 00000052 = 00000084
  - = 0000 0000 0000 0000 0000 0000 0000 1000 0100



LSR Logical Shift Right

- EXAMPLE
  - **ADD R0, R2, R3, LSR 6**
  - R2 = 00010049, R3 = 00000531
- Solution
  - 00010049 + 00000531 >> 6
  - 00010049 + 00000014 = **0001005D**
  - R0 = **0000 0000 0000 0001 0000 0000 0101 1101**.

Arithmetic Shifting

- A drawback of logical shifting is that signed numbers cannot be taken care of (results may lead to incorrect sign bit).
- Arithmetic shifting will consider the sign bit after the shifting
- **Arithmetic Right Shift preserves the sign bit**
- EXAMPLE: perform ASR (arithmetic shift right) on 32-bit number value 0x8621A93C **19 positions**?
- SOLUTION:

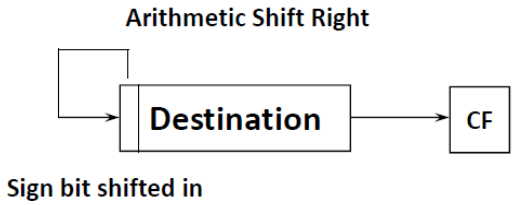
  1000 0110 0010 0001 1010 1001 0011 1100

  1111 1111 1111 1111 1111 0000 1100 0100

  = 0xFFFFF0C4

ASR Arithmetic Right Shift

- rm , ASR rs
- rm , ASR #immediate



- Example
  - **MOV R0, R6, ASR R5**
  - R5 = 00000015, R6 = 80A62150
- Solution
  - R6 >> 21 = FFFFFC05 = 1111 1111 1111 1111 1111 1100 0000 0101
  - R0 = FFFFFC05.
  - R6 and R5 unchanged.

ASR Arithmetic Right Shift

- Example
  - ADD R8, R1, R3, ASR R2
  - R1 = 0000AE09, R3 = FFFFF0C0, R2 = 00000005
- Solution
  - R3 >> 5 = FFFFFF86 = 1111 1111 1111 1111 1111 1111 1000 0110
  - R8 = R1 + (R3 >> 5)
  - R8 = R1 + R3/32 = **0000AD8F**
  - R8 = **0000 0000 0000 0000 1010 1000 1111**.

Rotation

- Rotation is similar to shifting bit differ in the sense **that bits shifted out of one side enters into the other end**.
- Rotate Left

  Moving all the bits to the left. It is rotated around at the ends.
- EXAMPLE: rotate **6 position to the left** for the value 0xA7250149?
- SOLUTION

  1010 0111 0010 0101 0000 0001 0100 1001

  1100 1001 0100 0000 0101 0010 0110 1001

Rotation

- EXAMPLE: rotate 0xE79212DB 12 positions to the right?
- SOLUTION:

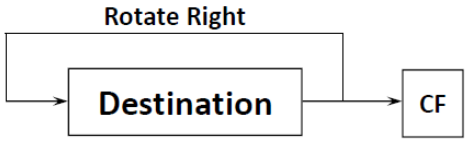
  1110 0111 1001 0010 0001 0010 1101 1011

  0010 1101 1011 1110 0111 1001 0010 0001

  = 0x2DBE7921

ROR Rotate Right

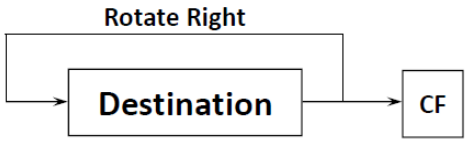
- rm , ROR rs
- rm, ROR #immediate



- EXAMPLE
  - **MOV R4, R1, ROR R5**
  - R1 = 8AE40185, R5 = 00000013
- Solution
  - R1 = 1000 1010 1110 0100 0000 0001 1000 0101 (**Before**)
  - R1 = **8030B15C**
  - R1 = **1000 0000 0011 0000 1011 0001 0101 1100**. (**After**)

ROR Rotate Right

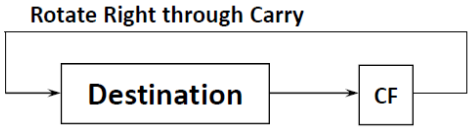
- rm , ROR rs
- rm, ROR #immediate



- EXAMPLE
  - **MOV R4, R1, ROR R5**
  - R2 = 4A08135F, R3 = FE200179, CPSR = 00000010 (CF=1)
- Solution
  - R

RRX Rotate Right Extended

- Rotate Right Extended
- carry flag is shifted into source register from left, and LSB is shifted out of right side to carry flag
- Syntax
  - rm, RRX
- EXAMPLE
  - **MOV R3, R2, RRX**
  - R2=0xFP4A08135F, CPSR = 0xA0000010 CF=1
  - R3= 0xA50409AF
  - R2, CPSR NOT changed.



Restrictions on Immediate data field

- **8-bit only to immediate data** field to be shifted or rotated number of position. And 4-bit for number of shifts.
- why? 32-bit if encoding leaves only limited bits for immediate
- Example
  - determine if **0x00029C00** is within the allowable range for immediate data in ARM instruction?
- Solution
  - 0000 0000 0000 0010 1001 1100 0000 0000 haves 8-bit constant **10100111**. so it can be shifted to left 10 positions. So it is valid immediate.

Restrictions on Immediate data field

- EXAMPLE
  - can the constant **8,352** be represented in as immediate data immediate in ARM instruction code?
- SOLUTION
  - 8,352 = 00000000000000001000001100000
  - the extracted 9 bit number is in yellow but it is 9-bit number, so it **cannot** be represented in the 8-bit place even if we apply the left shift operation.
- EXAMPLE
  - use minimal code to set bits 3, 18, and 20 of R0?
- SOLUTION
  - The mask is **0X00140008**=0000 0000 0001 0100 0000 0000 0000 1000.
  - can be done in 2 steps. first OR with **0x14** shifted to left 16 positions. Then OR with only the 8-bit **0x08**. Can you write the code?