ESET 349 – Microcontroller Architecture

Logic and Arithmetic

Dr. Muhammad Faeyz Karim

FLAGS (Revision)

- As mentioned earlier, flags are located in the most significant 4 bits (N Z C V) in CPSR.
- Instructions can be made to set flags by appending an “S” to the mnemonic, e.g. using ADDS instead of ADD.

N Flag

- Checks for a negative result.
- Negative if most significant bit is 1.
- In both of the following cases, the N Flag is set

```
   FFFFFFFF          7B000000
 + FFFFFFFE         +30000000
---------------     ----------------
   FFFFFFFD          AB000000   Negative?
```

- Programmer has to decide whether to use the N flag

V Flag

- The V flag indicates a sign overflow
- V flag is calculated by doing an exclusive OR of the carry bit going into bit 31 of the result with the carry bit coming out of bit 31.
- In the second example in previous slide, the V flag will be set and the programmer will ignore the N flag.

Z flag

- Z flag tell us the result is All Zeroes
- All 32 bits must be 0.

C Flag

- The C flag is set if

– result of an addition is greater than or equal to 2^{32}

– Result of an inline barrel shifter operation in a move or logical instruction.

COMPARISON INSTRUCTIONS

- 4 instructions do nothing except set the condition codes or test for a particular bit
- CMP – Compare. CMP subtracts a register or an immediate value from a register value and updates the condition codes. You can use CMP to quickly check the contents of a register for a particular value, such as at the beginning or end of a loop.
- E.g.

```
CMP r8, #0   ;r8==0
BEQ  routine ;yes, then go to my routine
```

CMN

- CMN – Compare negative. CMN adds a register or an immediate value to another register and updates the condition codes.
- It is the inverse of CMP
- If you typed CMP r0, #-20 , the assembler will generate CMN r0, #0x14 (note 0x14 = 20)

Test Instructions

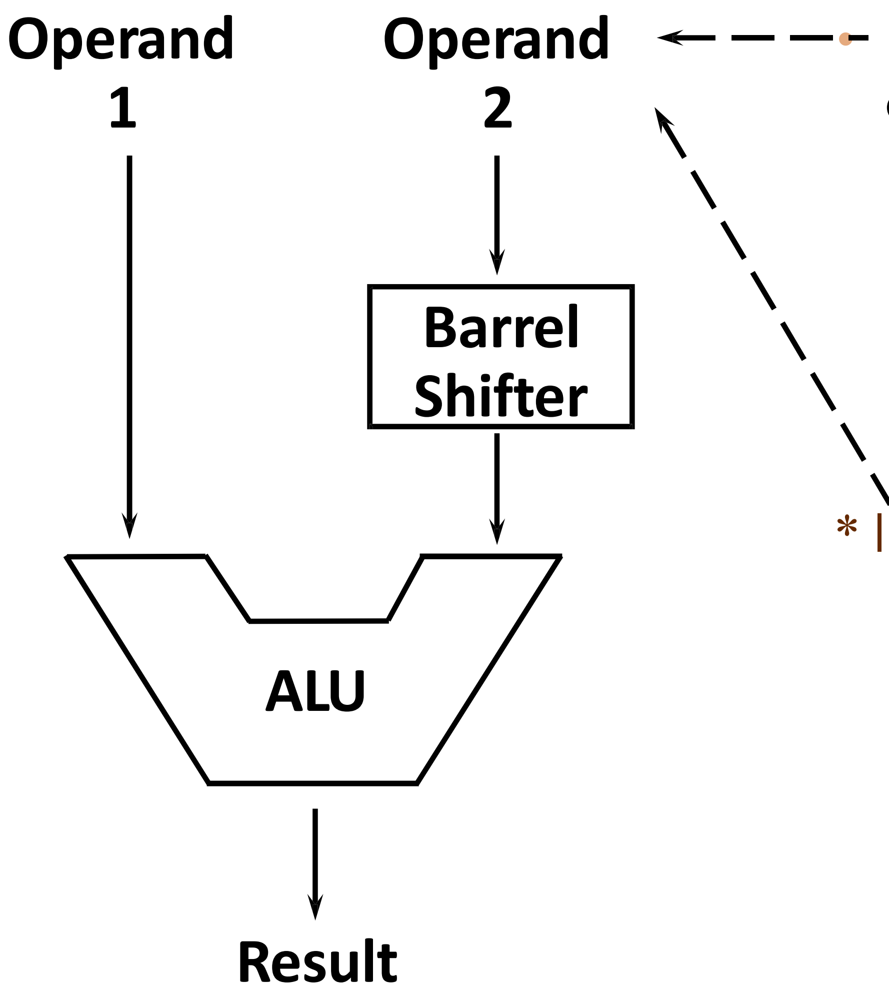
- TST – Test. TST logically ANDs an arithmetic value with a register value and updates the condition codes without affecting the V flag.
- TEQ – Test equivalence. TEQ logically exclusive Ors an arithmetic value with a register value and updates the condition codes without affecting the V flag. You can use TEQ to determine if two values are the same. E.g.

```
TEQ    r9, r4, LSL #3
```

Boolean Operations

Instruction	Comment
AND	Logically ANDs two operands
ORR	Logically ORs two operands
EOR	Exclusive OR of two operands
MVN	Move Negative – logically NOT's all bits

Using the Barrel Shifter: The Second Operand



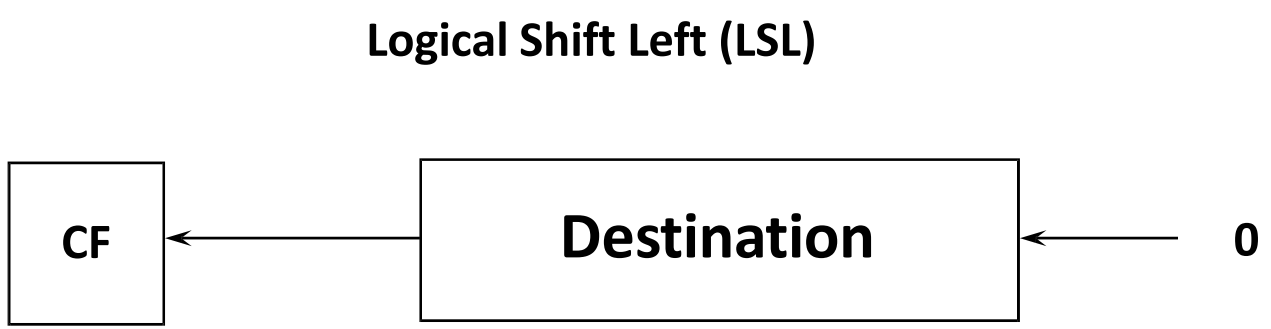
Register, optionally with shift operation applied.

* Immediate value

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

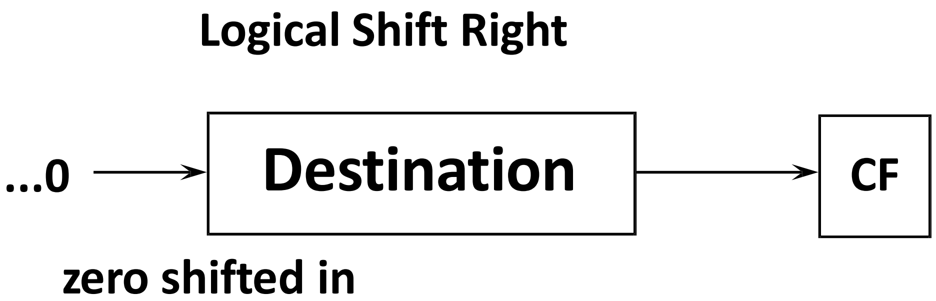
Barrel Shifter - Left Shift

- Shifts left by the specified amount (multiplies by powers of two) e.g.
- LSL #5 => multiply by 32

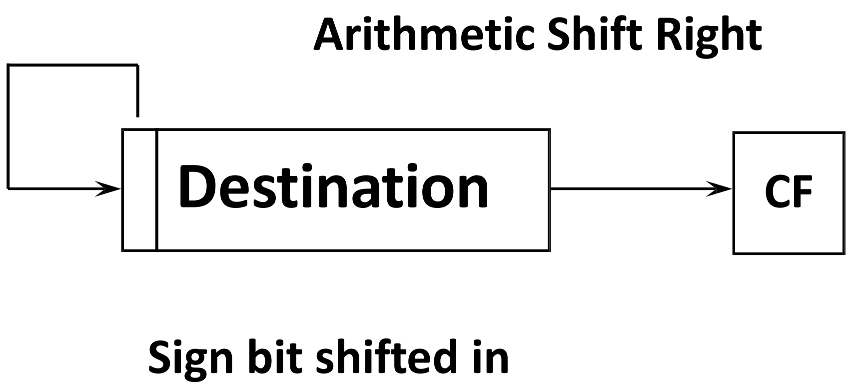


Barrel Shifter - Right Shifts

<u>Logical Shift Right (LSR)</u>
Shifts right by the specified amount (divides by powers of two) e.g. **LSR #5 = divide by 32**



<u>Arithmetic Shift Right (ASR)</u>
Shifts right (divides by powers of two) and preserves the sign bit, for 2's complement operations. e.g. **ASR #5 = divide by 32**



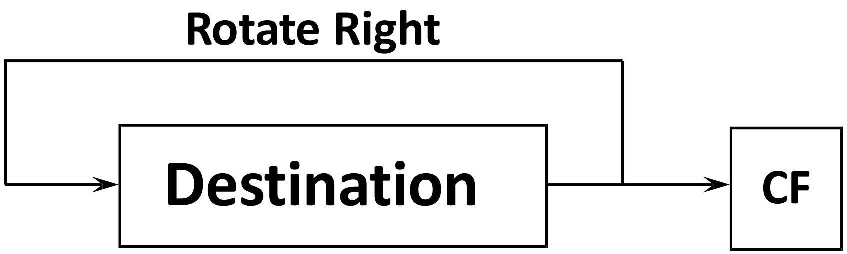
Barrel Shifter - Rotations

Rotate Right (ROR)

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

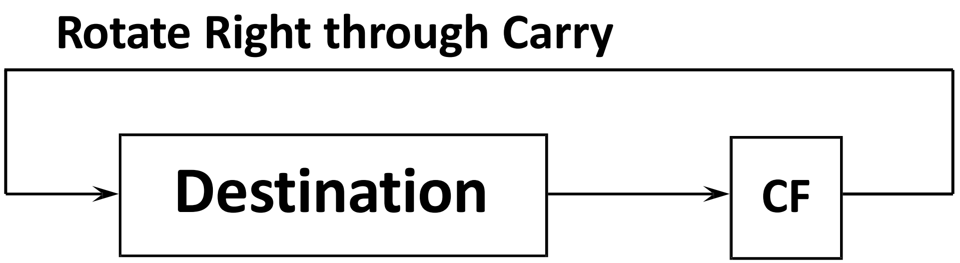
e.g. ROR #5

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.



Summary Of Shifts And Rotates

Shift/Rotate	Operation	Use
LSL	Logical shift left by n bits	Multiplication by 2^n
LSR	Logical shift right by n bits	Unsigned division by 2^n
ASR	Arithmetic shift right by n bits	Signed division by 2^n
ROR	Rotate right by n bits	32-bit rotate
RRX	Rotate right extended by one bit	33-bit rotate, 33^{rd} bit is Carry flag

Shift and rotate examples

- MOV   r4, r6, LSL #4 ; r4 = r6 << 4 bits
- MOV   r4, r6, LSL r3 ; r4 = r6 << # specified in r3
- MOV   r4, r6, ROR #12 ; r4 = r6 rotated right 12 bits
- All shift operations take one clock cycle to execute, except register-specified shifts which take another clock cycle.
- Shifts and rotates are a very fast way to perform certain multiplications and divisions.

Example

- The shift and logical operations can also be used to move data from one byte to another. Suppose we need to move the uppermost byte from register r2 and put it at the bottom of register r3. The contents of register r3 are shifted left by 8 bits first.

```
MOV  r0, r2, LSR #24 ;extract top byte from r2 into r0

ORR  r3, r0, r3, LSL #8 ;shift up r3 and insert r0
```

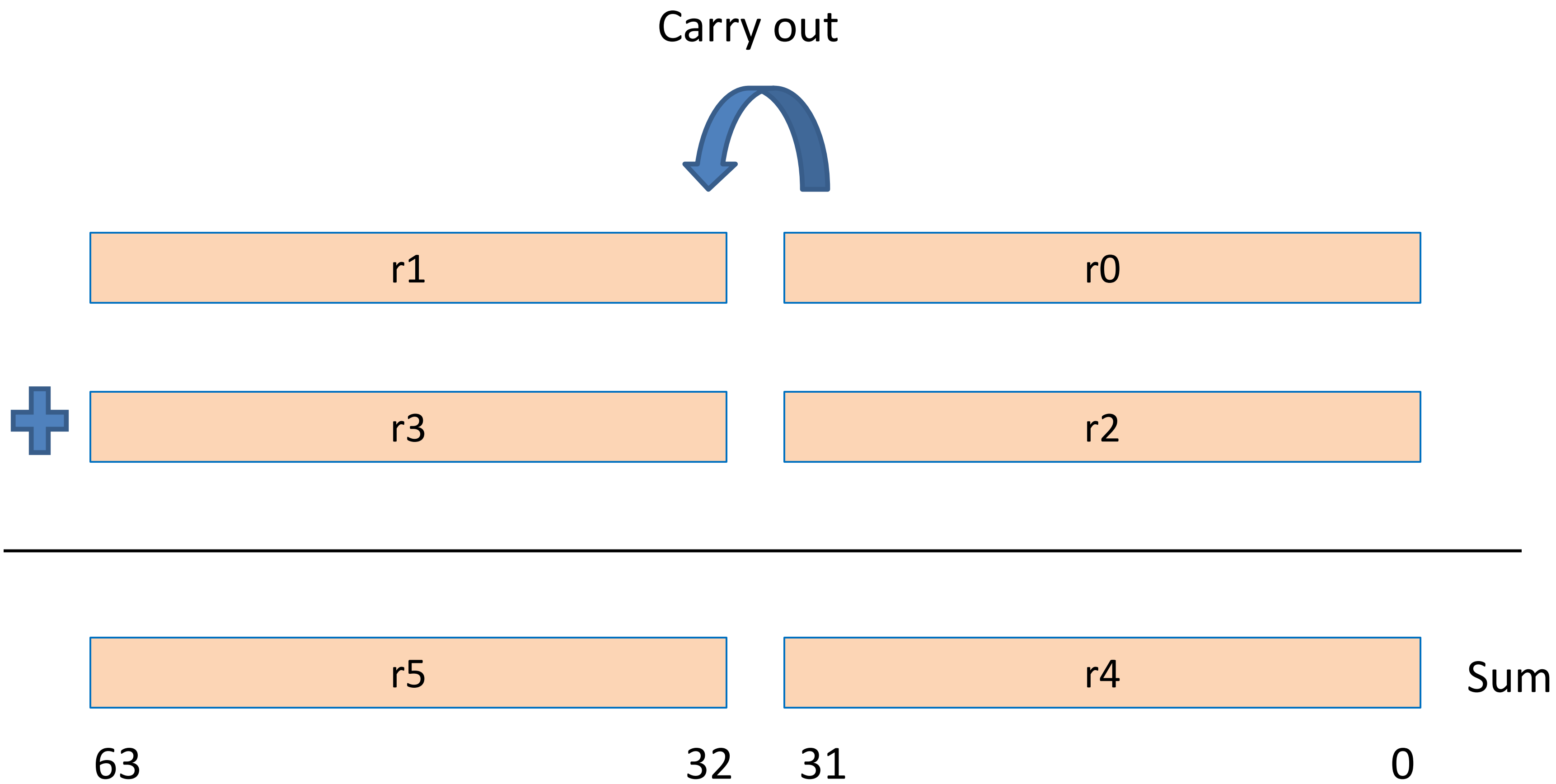
Workings

- Let r2 contains 0xD0ABCDEF
- MOV r0, r2, LSR #24 will result in r0 containing

0x000000D0 or 0xD0 (the upper most byte of r2)

- Assume r3 contains 0xABCD
- After LSL #8, r3 becomes 0xABCD00
- After the OR operation, r3 becomes 0xABCDD0!

64-bit addition



Multiplication by a constant

- It is sometimes possible to perform multiplications without using the hardware multiplier which may consumes more power and time.
- This can be done by making clever use of the barrel shifter for operand 2.
- Multiplying a number by a power of two can be easily perform using the barrel shifter

  MOV r1, r0, LSL #2; r1 = r0*4

Multiplication w/o using Multiplier

- Example, multiply by 5

  ADD r0, r1, r1, LSL #2 ; r0 = r1 + r1*4

- Example, multiply by 7

  RSB r0, r2, r2, LSL #3 ; r0 = r2*8 –r2

  ; r0 = r2*7

- Hence, it is easy to multiply a number by a power of 2, a power of 2 ± 1 without using the multiplier.

Complex Multiplication

- More complex multiplication can be constructed from simpler multiplications, e.g. *35 can be made from *7 followed by *5.
- Example: multiply by 115

```
ADD r0, r1, r1, LSL #1 ; r0 = r1*3
SUB r0, r0, r1, LSL #4 ; r0 =(r1*3)-(r1*16)
                       ;    =r1*(-13)
ADD r0, r0, r1, LSL #7 ; r0 = (r1*-13) + (r1*128)
                       ;    = r1*115
```