Computer organization and architecture

Lesson 6

More instructions

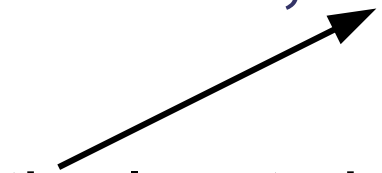
Equating a symbol to a numeric constant

Use the EQU directive

```
MY_VAR EQU 0xA     ; assigns A to the symbol MY_VAR
  MOV R6, #MY_VAR
```

Multiply-accumulate instructions

```
MLA r7, r8, r9, r3      ; r7 = r8 * r9 + r3
```



the least significant 32 bits of the result

```
SMLAL r4, r8, r2, r3    ; {r8,r4} = r2*r3 + {r8,r4}
UMLAL r5, r8, r0, r1    ; {r8,r5} = r0*r1 + {r8,r5}
```

MS32bits LS32bits

These instructions can boost the math performance in applications such as matrix multiplication and signal processing consisting of repeated multiplies and adds.

Reverse subtract

```
RSB Rd, Rn, Src2      ; Rd ← Src2 - Rn
```

Add with carry

```
ADC Rd, Rn, Src2      ; Rd ← Rn+Src2+C
```

The ADC instruction adds the values in Rn and Src2, together with the carry flag.

Subtract with carry

```
SBC Rd, Rn, Src2      ; Rd ← Rn - Src2 - !C
```

If the carry flag is clear, the result is reduced by one.

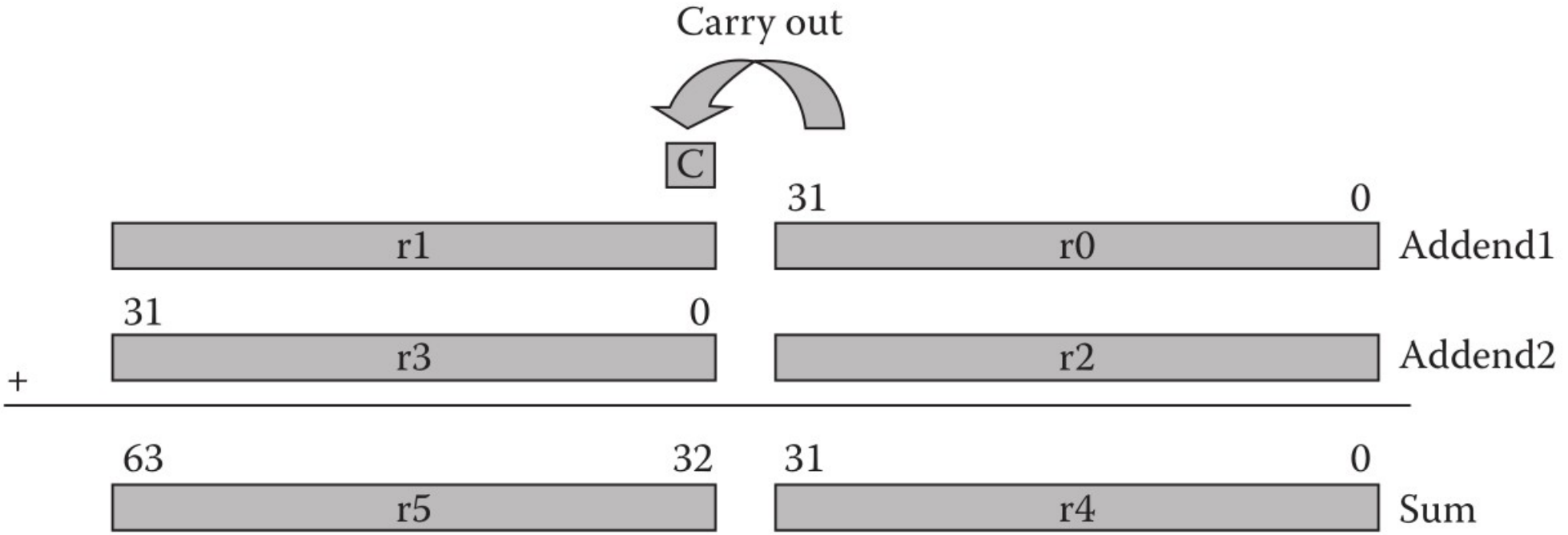
Reverse subtract with carry

```
RSC Rd, Rn, Src2      ; Rd ← Src2 - Rn - !C
```

Multiword arithmetics

Add a 64-bit integer contained in R2 and R3 to another 64-bit integer contained in R0 and R1, and place the result in R4 and R5.

```
ADDS R4, R0, R2  ; add the least significant words
ADC  R5, R1, R3  ; add the most significant words
                 ; with carry
```



Multiword values do not have to use consecutive registers.

96-bit subtraction:

```
SUBS R6, R6, R9  ; sub the least significant words
SBCS R9, R2, R1  ; sub the middle words with carry
SBC  R2, R8, R11 ; sub the most significant words
                 ; with carry
```

Consider the following operation:

```
SUB r0, r2, r3, LSL #2 ; r0 = r2 - r3*4
```

Suppose we want modify (shift) register r2 before the subtraction instead of register r3.

This is done using the reverse subtract operation

```
RSB r0, r3, r2, LSL #2 ; r0 = r2*4 - r3
```

Example: Write an assembly program to perform the function of absolute value.

Try to use only two instructions.

Solution:

```
MOVS  r1, r0
RSBLT r1, r1, #0
```

The program first sets the status flags to see if anything needs to be done.

If the argument is zero, the result is zero.

If the argument is negative (LT indicates Less Than zero), the reverse subtract instruction subtracts r1 from zero, effectively changing its sign.

Examine the following instruction:

```
ADD r0, r1, r1, LSL #2 ; r0 = r1 + r1*4 = r1*5
```

Why do it this way instead of using `MUL`?

Answer: the size and power usage of a multiplier array are large.

In very low power applications, it's often necessary to play every trick in the book to save power:

- not clocking logic that is not being used
- powering down caches or the entire processor if it is not needed
- reducing voltages and frequencies
- etc

By using only the 32-bit adder and a barrel shifter, the ARM processors can actually generate multiplications by 2^n , 2^{n-1}, and 2^{n+1} in a single cycle, without having to use a multiplier array.

Example:

```
RSB r0, r2, r2, LSL #3      ; r0 = r2*8 - r2 = r2*7
```

The reverse subtract instruction was used here, since an ordinary subtraction will produce the wrong result.

By chaining together multiplications, multiplying by other constants can be done

```
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
```

Reading from and writing to Program Status Registers

`MRS` (Move PSR to general-purpose register)

– instruction to read the flags

`MSR` (Move general-purpose register to PSR)

– instruction to write the flags

```
MRS r0, CPSR ; load the contents of the CPSR into r0
MRS r1, SPSR ; load the contents of the SPSR into r1
```

From there, you can examine any flags that you like.

You cannot use register r15 as the destination register.

You must not attempt to access an SPSR in User mode, since the register does not exist.

Exercise 6.1

Try this example in Keil to learn about `MRS` and `MSR`

```
MOVS r1, #-1
MRS r0, CPSR
ADDS r3, #1
MOV r2, r0
MSR APSR, r2
```

Exercise 6.2

Write a compare routine to compare two 64-bit values, using only two instructions.

Hint: the second instruction is conditionally executed, based on the first comparison.

Exercise 6.3

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

a) 135

b) 255

c) 18

d) 16384

Exercise 6.4

Write four different instructions that clear register r7 to zero.

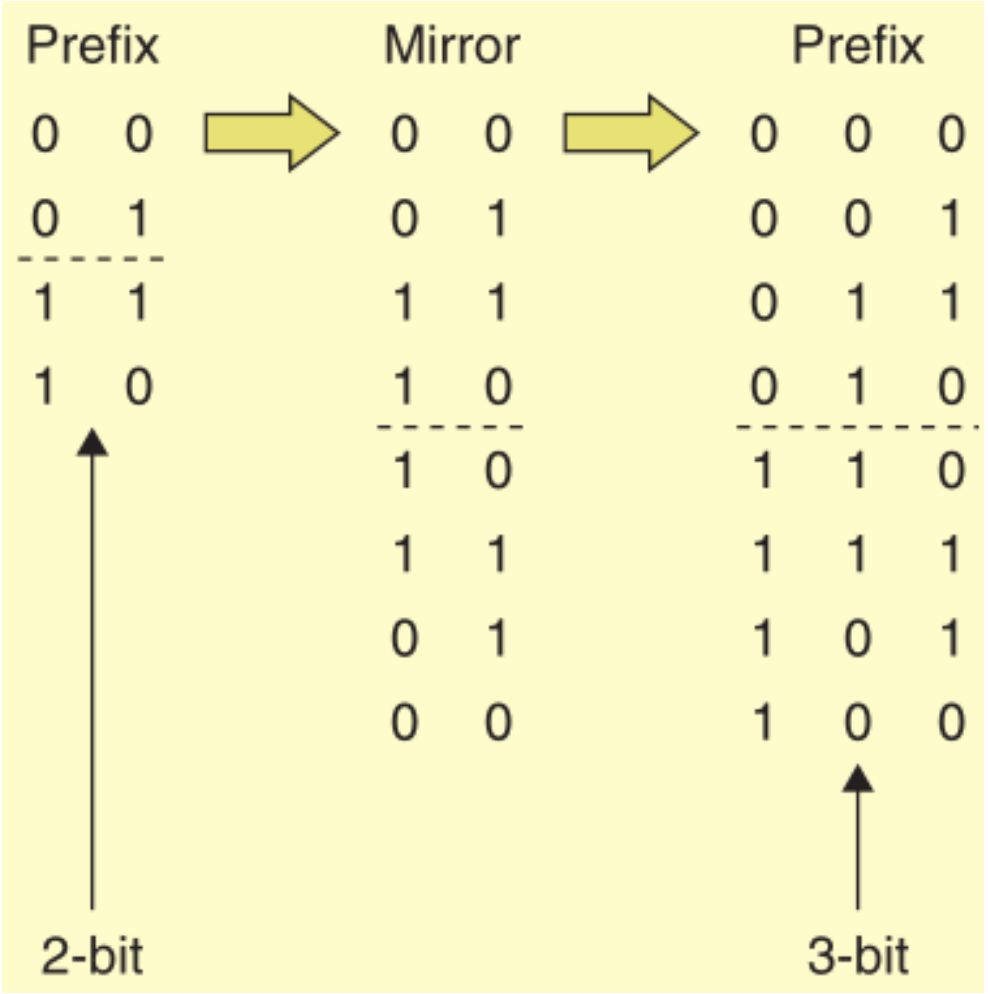
Exercise 6.5

Write ARM assembly to turn a 2-bit Gray code held in register r1 into a 3-bit Gray code in register r2.

Note:

The 2-bit Gray code occupies only bits [7:0] of register r1

The 3-bit Gray code occupies only bits [23:0] of register r2



You can ignore the leading zeros.