References:

Textbook Chapter 4, Chapter 9.1 – 9.2

“ARM Cortex-M Users Manual”, Chapter 3
CPU instruction types

- **Data movement operations (Chapter 5)**
  - memory-to-register and register-to-memory
    - includes different memory “addressing” options
    - “memory” includes peripheral function registers
  - register-to-register
  - constant-to-register (or to memory in some CPUs)

- **Arithmetic operations (Text – Chapter 4.1 – 4.5, Chapter 9.1-9.2)**
  - add/subtract/multiply/divide
  - multi-precision operations (more than 32 bits)

- **Logical operations (Text – Chapter 4.4 – 4.6)**
  - and/or/exclusive-or/complement (between operand bits)
  - shift/rotate
  - bit test/set/reset

- **Flow control operations (Text – Chapter 6)**
  - branch to a location (conditionally or unconditionally)
  - branch to a subroutine/function
  - return from a subroutine/function
ARM arithmetic instructions

- ADD\{S\}: $[Rd] \leftarrow Op1 + Op2$
- SUB\{S\}: $[Rd] \leftarrow Op1 - Op2$
- RSB\{S\} (reverse subtract): $[Rd] \leftarrow Op2 - Op1$

Why would we need RSB if we have SUB? (Op2 options?)

- ADD/SUB/RSB performed only on 32-bit operands
- ADDS/SUBS/RSBS also set Z/N/C/V flags
  - What if we have 8-bit or 16-bit data? 
    (Flags would not reflect 8 or 16-bit results)
- CPU cannot distinguish between signed and unsigned data
  - One 32-bit binary adder circuit in the ALU
  - SUB/RSB performed via 2’s complement arithmetic (whether data are signed or unsigned)
Addition Summary

Let the 32-bit result $R$ be the result of the 32-bit addition $X+M$.

- **N bit** is set
  - if unsigned result is above $2^{31}-1$ or
  - if signed result is negative.
  - $N = R_{31}$
- **Z bit** is set if result is zero
- **V bit** is set after a signed addition if result is incorrect
  - if signed result $< -2^{31}$ or signed result $> 2^{31}-1$
  - $V = X_{31} \& M_{31} \& R_{31}$  \vert  \unfill M_{31} \& R_{31} \& X_{31}$
- **C bit** is set after an unsigned addition if result is incorrect
  - if unsigned result is above $2^{32}-1$
  - $C = X_{31} \& M_{31}$  \vert  \unfill M_{31} \& R_{31}$  \vert  \unfill R_{31} \& X_{31}$
Subtraction Summary

Let the 32-bit result $R$ be the result of the 32-bit subtraction $X-M$

- **N bit** is set
  - if unsigned result is above $2^{31}-1$ or
  - if signed result is negative.
  - $N = R_{31}$
- **Z bit** is set if result is zero
- **V bit** is set after a signed subtraction *if result is incorrect (overflow)*
  - Signed result $< -2^{31}$ or signed result $> 2^{31}-1$
  - $V = X_{31} \& M_{31} \& R_{31}$ \or $X_{31} \& M_{31} \& R_{31}$
- **C bit** is clear after an unsigned subtraction *if result is incorrect (overflow)*
  - if unsigned result $< 0$ (unsigned $X <$ unsigned $M \Rightarrow$ “borrow” condition)
  - $C = X_{31} \& M_{31}$ \or $M_{31} \& R_{31}$ \or $R_{31} \& X_{31}$
Checking for overflow

- **Signed** operands:
  
  ```
  ADDS r3,r2,r1 ; r3 = r2 + r1  
  BVS Error ; branch if V flag set (overflow)
  SUBS r3,r2,r1 ; r3 = r2 + r1  
  BVS Error ; branch if V flag set (overflow)
  ```

- **Unsigned** operands:
  
  ```
  ADDS r3,r2,r1 ; r3 = r2 + r1  
  BCS Error ; branch if C flag set (carry = overflow)
  SUBS r3,r2,r1 ; r3 = r2 + r1  
  BCC Error ; branch if C flag clear (borrow = overflow)
  ```
ARM multiply instructions

- Product of 32-bit operands can be up to 64 bits long
  - Worst case unsigned product (product of max 32-bit values)
    \((2^{32} - 1) \times (2^{32} - 1) = 2^{64} - 2^{33} + 1\)
    \(0xFFFFFFFF \times 0xFFFFFFFF = 0xFFFFFFFFF00000001\)
  - Worst case signed products:
    - Positive \(\times\) Positive
      \((2^{31} - 1) \times (2^{31} - 1) = +2^{62} - 2^{32} + 1\)
      \(0x7FFFFFFF \times 0x7FFFFFFF = 0x3FFFFFFF00000001\)
    - Negative \(\times\) Negative
      \((-2^{31}) \times (-2^{31}) = +2^{62}\)
      \(0x80000000 \times 0x80000000 = 0x4000000000000000\)
    - Positive \(\times\) Negative
      \((2^{31} - 1) \times (-2^{31}) = -2^{62} + 2^{31}\)
      \(0x7FFFFFFF \times 0x80000000 = 0xC000000080000000\)

All results can be represented with 64 bits (no “overflows”)
ARM multiply instructions

[Rd] <= Op1 × Op2

- **MUL Rd, Rm, Rn** or **MUL Rm, Rn**
  - Saves *least-significant 32 bits* of the product in Rd
  - Valid result for both signed and unsigned operands
  - *No immediate form* for Op2

- **MULS Rm, Rs**
  - MUL updates N and Z flags (C and V are unaffected)
  - Restricted to form Rm, Rs and to registers R0-R7

- **UMULL/SMULL RdLo, RdHi, Rm, Rs**
  - Unsigned (UMULL) and Signed (SMULL) “Long Multiply”
  - 64-bit product $P_{63}-P_0$ put into two registers:
    \[ [\text{RdHi}] <= P_{63}-P_{32}, [\text{RdLo}] <= P_{31}-P_0 \]
  - No condition flags set
ARM divide instructions

- **SDIV**  \( Rd, Rn, Rm \)  (signed)
- **UDIV**  \( Rd, Rn, Rm \)  (unsigned)
  - Integer division:  \( Rd = Rn \div Rm \)  (\( = Rn/Rm \))
  - Can also use form “\( Rn, Rm \)”:\  \( Rn = Rn \div Rm \)
  - Result is truncated (rounded toward 0)
    - Result = “quotient”, with “remainder” discarded
  - Condition flags are unaffected
Example: C assignment statements

- **C**: \[ x = (a + b) - c; \]
- **Assembler**: 
  
  ```assembly
  LDR r4,=a ; get address for a
  LDR r0,[r4] ; get value of a
  LDR r4,=b ; get address for b, reusing r4
  LDR r1,[r4] ; get value of b
  ADD r3,r0,r1 ; compute a+b
  LDR r4,=c ; get address for c
  LDR r2,[r4] ; get value of c
  SUB r3,r3,r2 ; complete computation of x
  LDR r4,=x ; get address for x
  STR r3,[r4] ; store value of x
  ```
Example: C assignment

- C: \( y = a \times (b+c); \)
- Assembler:
  
  ```assembly
  LDR r4,=b ; get address for b
  LDR r0,[r4] ; get value of b
  LDR r4,=c ; get address for c
  LDR r1,[r4] ; get value of c
  ADD r2,r0,r1 ; compute partial result
  LDR r4,=a ; get address for a
  LDR r0,[r4] ; get value of a
  MUL r2,r2,r0 ; compute final value for y
  LDR r4,=y ; get address for y
  STR r2,[r4] ; store y
  ```
Multi-precision arithmetic

- **What if we need arithmetic for numbers > 32 bits?**
- **Consider addition/subtraction of decimal numbers:**
  
  \[
  \begin{align*}
  53 & \quad \text{Carry 10 from 1\textsuperscript{st} to 2\textsuperscript{nd} column} \\
  + 29 & \quad (1 \text{ added to 2\textsuperscript{nd} column}) \\
  \hline
  82 \\
  \\
  53 & \quad \text{Borrow 10 from 2\textsuperscript{nd} to 1\textsuperscript{st} column} \\
  - 29 & \quad (1 \text{ subtracted from 2\textsuperscript{nd} column}) \\
  \hline
  24 \\
  \end{align*}
  \]

- **CPU: add/subtract 32-bit parts of #s, with carry/borrow between parts**
  - **ADC (add with carry):** \[ \text{[Rd]} \leq \text{Op1} + \text{Op2} + \text{C} \]
  - **SBC (subtract with carry\(^*\)):** \[ \text{[Rd]} \leq \text{Op1} - \text{Op2} + (\text{C} - 1) \]
  - **RSC (reverse subtract with carry\(^*\)):** \[ \text{[Rd]} \leq \text{OP2} - \text{Op1} + (\text{C} - 1) \]
    - \(^*\) \text{C=0 indicates “borrow” for subtraction}

- **Examples: (in class)**
ARM multiply/accumulate instructions

- **MLA**: multiply with accumulate (32-bit result)
  \[
  \text{MLA Rd, Rm, Rs, Rn} : \ [\text{Rd}] \leq \text{Rn} + (\text{Rm} \times \text{Rs})
  \]

- **MLS**: multiply and subtract (32-bit result)
  \[
  \text{MLS Rd, Rm, Rs, Rn} : \ [\text{Rd}] \leq \text{Rn} - (\text{Rm} \times \text{Rs})
  \]

- **UMLAL (unsigned) / SMLAL (signed)**
  - Multiply with accumulate, \textbf{long} (64-bit result)
  \[
  \text{UMLAL RdLo, RdHi, Rm, Rs} : \ [\text{RdHi}:\text{RdLo}] \leq \text{RdHi}:\text{RdLo} + (\text{Rm} \times \text{Rs})
  \]

Example (in class) – DSP algorithm