School of Electrical Engineering and Computing, pp. 1–5

Lab 6:
Fundamentals of Assembly Language Part 2: Flow Control and Arithmetic

ELEC1710

---

1 Introduction

This programming task demonstrates the basic use of arithmetic and flow control instructions in Thumb assembly. Conditional execution will be introduced by modifying Lab 4's `blinky` example so that the blinking can be controlled with an external push button. Arithmetic instructions will then be used to implement a basic `for` loop which will cause the LED to only flash a set number of times.

2 Assembly Reference

2.1 Arithmetic: `add` and `sub`

The arithmetic instructions `add` and `sub` can be used in several different ways. The general syntax is as follows:

```
add{s}{cond} {Rd,} Rn, Operand2
sub{s}{cond} {Rd,} Rn, Operand2
```

where:

- `s` optionally updates the condition flags in `APSR`. **NB:** This is required when using conditional execution.
- `cond` is a conditional execution suffix from Figure 1.
- `Rd` is an optional destination register. If not specified the result overwrites `Rn`.
- `Rn` is the first operand.
- `Operand2` is the second operand and can be a constant preceeded by a # symbol (eg #14), another register, or another register with bit shift.

The most basic usage is to add or subtract a constant, for example:

```
add r2, #143   // r2 = r2 + 143
sub r1, #4   //r1 = r1 - 4
```

Using this syntax the constant can be any value in the range 0-4095 (ie: it is encoded as a 12-bit unsigned binary number).

It is possible for the destination register to differ from the source, for example:

```
add r1, r2, #4   // r1 = r2 + 4
```

or for the 12-bit constant to be replaced with a register:

```
add r1, r2, r5   // r1 = r2 + r5.
```

In its most complicated form the last operand can also include a shift, for example:

```
add r2, r3, r4, LSL #2   // r2 = r3 + (r4 << 2) = r2 = r2 + 4*r4
```

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
CC or LO	C = 0	Lower, unsigned <
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned ≤
GE	N = V	Greater than or equal, signed ≥
LT	N != V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 and N != V	Less than or equal, signed ≤
AL	Can have any value	Always. This is the default when no suffix is specified.

Figure 1: Condition code suffixes, extracted from *The Cortex-M3 Instruction Set*, ST-Microelectronics document number PM0056.

2.2 Comparison: `cmp` and `cmn`

The comparison instructions set the conditions flags in `APSR` based on the sum or difference between two values. They are the equivalent to performing an `adds` or `subs` instruction except the result of the calculation is discarded. As such comparison instructions are more register efficient than arithmetic instructions.

The syntax is as follows:

```
cmp{cond} Rn, Operand2
cmn{cond} Rn, Operand2
```

Where `Rn` must be a register and `Operand2` can be a constant, register, or register with bit shift.

The `cmp` sets condition flags from the result of `Rn - Operand2` while `cmn` performs `Rn + Operand2` and sets flags.

A full list of condition suffixes and associated `APSR` flags can be found in Figure 1.

Usage example:

Listing 1: Example of the `cmp` instruction.

```
ldr r1, = 0x100
cmp r1, #0x100  // Results in Z = 1, EQ, LS and LE conditions become true
```

2.3 Branch: `b`

The `b` instruction causes program execution to jump to a location specified by an assembly label. The syntax useful for this lab is:

`b{cond} label`

Where `label` is an ASCII string followed by a ':' character in the assembly listing and `cond` is a condition suffix from Figure 1.

Example:

Listing 2: Example of the `b` instruction.

```
start:
    // Useful code goes in here

    b start     // Jump back to "start" to form an infinite loop
```

2.4 If-Then: `it`

In Thumb assembly all instructions using a conditional execution suffix must be preceded by an `it` instruction. Up to four instructions can be included in this block however only basic usage will be documented here.

The full syntax is:

`it{x{y{z}}} cond`

Where `x`,`y` and `z` specify the condition for the optional 2nd, 3rd and 4th instructions and are either `t` (then, execute if `cond` is met) or `e` (else, execute if `cond` is NOT met). The `cond` suffix is one of those listed in Figure 1.

Simple examples of the `it` instruction are shown in Listings 3 and 4. More advanced usage requiring multiple instructions should not be required for this lab.

3 Code Examples

Listing 3: `if()` statement example in Thumb assembly.

```
// if(r1 == 5)

        cmp r1, #5      // Compare r1 and the decimal constant 5, sets status bits
        it eq           // If-Then block testing equal-to status
        beq iseq        // branch to 'iseq' if status bits matched equal to
        b isnoteq       // Unconditional branch (ie: GOTO) 'isnoteq'

iseq:
        // Code executed if r1 is equal to 5

isnoteq:
```

Listing 4: `for()` loop example in Thumb assembly

```
// for(x = 0x80000; x > 0; x--) { loop_code() }

    ldr r3, =0x00080000 // This function counts down from this number
testExit:               // Label addressing start of loop
    sub r3, r3, #1      // Subtract 1 from r3 and store result back in r3
    cmp r3, #0          // Compare r3 to zero. This sets status flags
    it gt               // IT (if-then) block with greater than condition
    bgt do_loop         // If the comparison result (ie: the status flags)
    b exitLoop          // Unconditional branch to 'exitLoop',
                        // this occurs if the branch to 'do_loop' failed
do_loop:
    // write loop_code() here
    b testExit          // After doing stuff test the exit condition

exitLoop:
    // The rest of your program
```

4 Lab Task

1. Construct a single push button with pull-down resistor and connect it to `GPIOA 0` (pin `A0` on the development board).
2. Modify `main.c` so that it calls `blinky`. If your code does not compile or is otherwise corrupted you can begin this lab by re-extracting the working template from Lab 4.
3. Modify constants in `blinky.s` to increase the blink rate to approximately 2-3Hz. Maintain the 50% duty cycle.
4. Add code at the start of `blinky` which reads the status of `GPIOA` into a register. Using `cmp` and `it` instructions implement an `if` statement such that the LED only blinks when the external push button is pressed.
5. Build a `for` loop around the LED blinking code so that when the button is pushed it causes the LED to flash on/off 10 times then remains off until the button is pushed again.