Digital Systems: Problem sheet 1

Mike Spivey, Hilary Term, 2022

Questions on this sheet probe details of ARM Cortex-M0 assembly language, and also of its encoding as binary machine code. It can't be emphasised enough that, though a lot of this detail is needed for the experience of understanding how a particular computer works (which every Computer Science student should enjoy at some stage), it isn't detail that is worth memorising in the medium term. Id est, *it won't be on the exam.*

1 Find as many single Thumb instructions as you can that have the effect of setting register `r0` to zero, and as many as you can that copy register `r1` to register `r0`. In each case, the instruction may or may not set the condition codes: say which.

To answer this exercise, use the ARM Architecture Reference Manual (linked from the micro:bit page on the wiki) to look up the details of some of the instructions shown in red, orange, yellow or green in tables [A] and [B] of the Rainbow Chart.

2 The following listing shows a disassembly of the simple multiplication routine from the lecture.

```
           func:
c0: 2200       movs    r2, #0
           loop:
c2: 2900       cmp     r1, #0
c4: d002       beq     cc <done>
c6: 3901       subs    r1, #1
c8: 1812       adds    r2, r2, r0
ca: e7fa       b       c2 <loop>
           done:
cc: 0010       movs    r0, r2
ce: 4770       bx      lr
```

(I've edited it a bit to remove distracting details.) Decode some of the instructions by hand, and in particular explain the displacements that appear in the `beq` and `b` instructions.

Again look up the instructions in the ARM Architecture Reference Manual, and use the information about encodings given there to decypher some of the hexadecimal numbers shown in the listing.

3 As it stands, the multiplication routine shown in Exercise 2 has a loop that contains 5 instructions, and takes 7 cycles for each iteration but the last: three cycles for the branch back to the start, and one cycle for each of the other instructions, including the untaken conditional branch. Try to rewrite the loop so that it contains fewer instructions. *A good solution has 3 instructions in the loop, taking 5 cycles per iteration, but it is possible to go further using 'loop unrolling' and make a still faster routine.*

4 Note that conditional branch instructions in Thumb code, such as `beq`, have a displacement field of limited size, but the unconditional branch `b` has a bigger displacement field. Show how a two-instruction sequence can be used to simulate conditional branches with a bigger range than can be achieved with a single `beq`. What is the penalty in execution time for doing this? The branch-and-link instruction `bl` has a still larger displacement field. Can it be used in a similar way to simulate even longer conditional branches?

5 Show with examples why different instructions `blt` and `blo` are needed following comparison of signed and unsigned numbers. Find out the conditions under which each of these branches is taken, and explain why the result is correct in each case.

6 Write (in some high-level language – C, Scala, or Python if you must) a routine for unsigned integer division, using the naïve algorithm based on repeated subtraction. Code the result in assembly language.

To answer this problem well, you must: formulate precisely the problem to be solved; give a clear algorithm for the problem and justify its correctness; code the algorithm in commented assembly language in an efficient way, considering the possibility of overflow; comment on errors that should be detected.

7 Repeat the previous exercise using a better algorithm. Fancier versions of the ARM have an instruction `clz r1, r2` that sets `r1` to the number of leading zeroes in the binary number in `r2`; what use is this instruction in writing a division routine?