Digital Systems: Problem sheet 6

Mike Spivey, Trinity Term, 2019

1 [Hennessy&Patterson exx. 5.1–2, translated] A common fault in chip manufacture is that signals become stuck at logic 0 or logic 1. For each of the following stuck-at faults in the processor design shown in the handout, describe the effect on the function of the processor. Which instructions would still work correctly?

(a) *cRegSelC* = *Rd* or *Rt*.

(b) *cRand2* = *RegB* or *Imm8*.

(c) *cMemRd* = *F* or *T*.

(d) *cWReg* = *N* or *Y*.

(e) *cWFlags* = *F* or *T*.

2 When the `bl` instruction is split into two halves, the decoding table reveals that the displacement in the first half is sign extended, but that in the second half is not. Why is this correct?

3 In native ARM code, not just branches, but almost any operation can be made conditional on the flags. One of the most useful such operations is a conditional move: the sequence

```
cmp r0, #0
moveq r0, r1
```

checks to see if `r0` contains zero, and if so, replaces its contents with the contents of `r1`. Similar instructions exist for all 14 conditions that are supported for conditional branches, and they all execute in one clock cycle, whether the move happens or not.

(a) Show how a conditional move instruction can be used to compute the maximum of two values in registers without using any branches. Compared with the branching code, how much time would be saved in the Cortex-M0 implementation?

(b) Devise an encoding for conditional moves as Thumb code, and write an entry for the decoding table that implements them in the single-cycle architecture. (If you want to add the instructions to the simulator, you can use the opcodes 24 and 25 that are so far unimplemented.)

(c) Leaving aside the issue of finding suitable encodings for the instructions, what other operations could be made conditional and implemented with the existing single-cycle datapath? What operations could not be made conditional without changing the datapath, and why?

4 Native ARM code provides an addressing mode where the values of two registers are added to form the address, but the value of one of the registers is first shifted left. For example, the instruction

```
ldr r0, [r2, r3, LSL #2]
```

loads `r0` with the 4-byte value stored at address `r2 + r3*4`. Native code may shift the register by any amount, multiplying by any power of two, but we consider here only scaling by 4 as shown.

(a) Explain why this addressing mode is particularly useful in programs that contain a lot of array indexing.

(b) Write decoding rules for versions of the `ldr` and `str` instructions that implement this addressing mode. (If you want to add them to the simulator, you could use opcodes 14 and 15.)