Tuan Nguen

Question 1:

a)

b) cRand2 = RegB means that the second operand is a register. The instructions that would work correctly with this value of cRand2 are: adds r, subs r, ands r, movs r. If cRand2 = Imm8, then the second operand is an 8-bit number. The instructions that would correctly are: adds i8, subs i8, movs i8, because we are not accessing registers with the second operand.

c) cMemRd = F means that we will not read from the memory. The instructions that would work correctly are: adds r, subs r, ands r, movs i8, str r. If cMemRd = T, then we will read from the memory. The only instruction that would work correctly in this case is ldr r, because we are loading contents from a certain place in the memory. The other instructions might not work correctly, because if cMemRd = T, then we would be unnecessarily reading from the memory, which might corrupt the stored data.

d) If cWReg = N, this is essentially cRegWrite = F. The only instruction that would word correctly in this condition is the str instruction, because this is the only instruction(apart from branch), that does not write the registers. If cWReg = T, this is as if cRegWrite = T. In this case, all the arithmetic instruction and the ldr instruction.

e) If cWFlags = F, the the NZVC flags would not be updated. The only time, that that is the correct behavior is when we use the ldr , str and branch instructions. If cWFlags was stuck in T, then all the arithmetic and logical instructions would work correctly.

Question 2:

The behavior, that the first half of the bl instruction is sign extended is correct, because lr can decrease as well as increase. On the other hand, the pc can only increase.

Question 3:

a) The moveq instruction is executed when the N or the Z flag is set. Therefore, we can compute the maximum of two numbers with this code:

```
cmp r0, r1
moveq r0, r1   // r0-r1 < 0, move r1 to r0
```

Executing this code requires three less clock cycles, because if we were to have a branching instruction, it would take 1 cycles for cmp, 3 to change the pipeline, and one to execute the move. In this case we only need 1 clock cycles for the cmp instruction and 1 for the moveq instruction.

Suppose we need to move r1 to r0(calling convention) as r1 > r0. Then the conditional move instruction takes only one clock cycle. Whereas if we had a branching instruction, it would take 1 cycle for cmp, 3 cycles for the branching instruction, and 1 more to execute the move. In this case, we are saving 4 clock cycles.

Now suppose we did not have to move r1 to r0. This means r1 <= r0. Then the conditional move takes only one clock cycle. For the branching code, it would take 1 cycle for cmp and 1 for the move instruction. In this case, we would save 1 clock cycle.

b) We add a new signal to the table of existing instructions. Suppose it is called cBranch, which will have 2 states – F and T. It will take directly the state of enable, which is true, when the condition is satisfied, and false otherwise. The cBranch signal will go to ALU, which will indicate it whether it should move or not.

Question 4:

a) This addressing mode is very useful, because the elements of an array can be easily accessed. r2 might contain the base address of the array and r3 will be the index. Each element is a 4-byte word, so that is why we would left shift r3 by 2.

b)

ldr

cRegSelA – Rn
cRegSelB – Rm
cRegSelC – Rd
cRand2 – Imm8
cShiftOp – Lsl
cShiftAmt – Sh2
cAluSel – Add
cMemRd – T
cMemWr – F
cRegWrite – T

str

cRegSelA – Rn
cRegSelB – Rm
cRegSelC – Rd
cRand2 – Imm8
cShiftOp – Lsl
cShiftAm – Sh2
cAluSel – Add
cMemRd – F
cMemWr – T
cRedWrite – F