CSC 258 Assignment 4, Fall 2008

Due by 5:00 p.m., Friday December 5, 2008; no late assignments without written explanation.

> *This assignment can be submitted either electronically or on paper*, but not both. If you submit electronically, you must submit a single plain text file called ''a4'' which is formatted to look good in an 80-character-wide window. Please use underscores for subscripts; e.g. $R0_{out}$ would be written ''`R0_out`''. Please check that your file looks okay at the console of a CDF machine using ''cat a4''.

1. Write microcode for our simple one-bus architecture to perform the VELMA instruction ''SUB R0, 1234'', where the '1234' is in the address field of the instruction. Your answer will begin with ''0, 1, 2. standard fetch sequence'' (or you can simply begin with step 3). Use the ''Complement B'' control line along with Set Carry-In to perform the subtraction.

2.
a) Write microcode to perform the VELMA instruction ''MOV 1234, R0'' (again beginning with step 3).
b) Write microcode to perform the VELMA instruction ''MOV R0, 1234''.
c) Write microcode to perform the VELMA instruction ''EXCH R0, 1234'', assuming that we can do an exchange by turning on the Read and Write control lines in the same cycle.
d) Suppose that memory reads always take time which fits within two CPU cycles (that is, a memory read started on cycle 4 will be over by cycle 6, so that a Wait MFC on cycle 5 causes no delay), and memory writes always take time which fits within three CPU cycles. How long does each of your three above microcode routines take to execute?
e) To think about, not to submit: Given the results in part (d), why is it so unusual these days for CPUs to have an EXCH instruction?

3. Write microcode for our simple one-bus architecture to perform the VELMA instruction ''BEQ 2345''.

4. Consider an instruction set in which ''ADD'' instructions have two register fields (i.e. there's no memory address in an ADD).
a) Write microcode to perform ''ADD R0, R1'' using our simple one-bus architecture.
b) Write microcode to perform ''ADD R0, R1'' using the simple two-bus architecture presented in class and posted at http://www.dgp.toronto.edu/~ajr/258/notes/micro/two-bus.html

5. Our standard fetch sequence will still have to be three microinstructions long even with the two-bus architecture at http://www.dgp.toronto.edu/~ajr/258/notes/micro/two-bus.html . However, the three steps will be different. Note that all registers, including the MAR, the MDR, and the PC, can input only from the left bus and can output only to the right bus. Write the revised standard fetch sequence for this simple two-bus architecture. (The original standard fetch sequence is the first three steps of ''A full example'' in the web page http://www.dgp.toronto.-edu/~ajr/258/notes/micro/microcode.html .)

(over)

6. Write an interrupt service routine in VELMA to process keyboard input. Let us assume that at this point the running program is ignoring keyboard input. However, pressing ^C (which yields ASCII character code 3) should still raise a keyboard interrupt exception in the OS.

The interrupt vector is at memory locations 42 and 43. You should set the priority level to 4 while in your interrupt service routine.

You can assume that interrupts are already turned on.

When a keyboard interrupt occurs, you read the input ASCII character from location 14. If you want to raise a keyboard interrupt exception in the OS, you should do “JSR RAISE” with the number 2 in R0. (Do not push any parameters on the stack.)

You may want to look at the PDP-11 keyboard input examples in my I/O web pages, but this is not essential.

7. Just to think about, not to be submitted:
Your annoying friend claims that he has a microinstruction which will accomplish both $R1 \leftarrow [R2]$ and $R3 \leftarrow [R4]$ in a single cycle on the simple one-bus architecture. You start to show him a rigorous proof that this is impossible, but he asks you to watch and see. Sure enough, after the single cycle has passed, R1 now contains the same contents as R2; R3 now contains the same contents as R4; and the contents of R2 and R4 were not changed. You know that his microinstruction cannot possibly accomplish this in general; so how did it work in this particular case?