22C:160/55:132 High Performance Computer Architecture Homework 2

Assigned Feb 15, 06
Due Feb 23, 06
Total Points = 50
There are three questions.

Question 1 (15 points) Consider the (non-pipelined) datapath of the multicycle MIPS in Fig. 5.28. Consider implementing the following instructions on this machine:

addi rt, rs, constant	(Add immediate)	
jr rs	(Jump register)	
lui constant	(Load upper immediate)	(See Fig 2.23, p.95)

Check the semantics of each instruction from the green page inside the book as well as from the various sections in Chapter 2.

To answer the question, use a copy of Fig 5.28, and add additional components to it only if it is necessary. Show the implementations by writing down the register transfer expressions for each instruction.

Question 2. (25 points) We briefly discussed about a processor that has only one instruction in its instruction set, and yet can be programmed to solve any problem. The instruction is (i. j. k), where i, j, k are memory addresses of size 16-bit each. The meaning of the instruction is as follows:

M[j] := M[i] – M[j]
If the result is negative, then jump to address k
else execute the next instruction in sequence

(a) Draw the datapath of a processor that executes the instruction (M1, M2, M3). Feel free to use a memory with two read ports and one write port, but at any time you can perform either read or write operation, and not both. Use register transfer notations (as in Sec 5.5, pp. 325-329) to show the implementation of the instruction.

(b) Design the hardware control unit of the above processor, and explain your design. You need to come up with a diagram like Fig 5.38 in p.339 of your textbook. Also see Appendix C on the CD. Mark the states of the state machine as s0, s1, s2, etc, and write equations for each control signal in terms of s0, s1, s2, … A good documentation is essential.

Question 3. (10 points)
(a) Draw a timing diagram (by inserting bubbles, assume no extra data forwarding paths have been added) to show the execution of the following MIPS program on a pipelined version of the processor (as in Fig. 6.22):

```
add     r3, r4, r2
sub     r5, r3, r1
lw      r6, 200(r3)
add     r7, r3, r6
```

(b) How will an optimizing compiler reorder the instructions of the above program to maximize performance enhancement?