Lab session Datapath

Group A: November 9, 2009
Group B: November 10, 2009

Work in the given groups of two. Submit your solutions to the respective assignment on Blackboard. The file name is:
`a05_s0XXXXX_s0XXXXX.tar.gz`
One of the group members commits your solution. Keep an eye on the deadline (see Blackboard)!

1 Project

1. Build a circuit that implements a 16-bit program counter (PC) that selects an instruction in a RAM element of 16-bit words. By default, the PC is increased each clock cycle, and the next instruction is read from memory. The special case of branching/jumping must also be implemented. In this case, the PC must go back or forward according to the branch offset. You should have te following inputs and outputs:

name	in/out	width	meaning
branch?	I	1 bits	selects whether we want to branch or simple increase the PC
branch offset	I	16 bits	the branch offset w.r.t. the PC
instruction	O	16 bits	the selected instruction from memory

2. Use four 16-bit registers, a RAM element (16-bit addresses, 16-bit words) and your own ALU to implement a partial datapath. The circuit has the following inputs (no outputs):

name	in/out	width	meaning
operation	I	4 bits	denotes which operation is executed (cfr. ALU operations)
rs	I	2 bits	source register index for the current operation
rt	I	2 bits	2nd source register index for the current operation
rd	I	2 bits	destination register index for the current operation
offset	I	8 bits	memory index offset for the sw/lw operations

- The datapath must be able to perform so-called register operations. These are the 14 operations you implemented in your ALU. This time, operands are read from, and the result is stored into registers. The right registers are selected by specifying the rs, rt and rd index inputs. For binary operations (e.g. add, eq, ...), the registers are used as follows:
  $rd = $rs *operation* $rt
  For unary operations (e.g. not, sl, ...), the registers are used as follows ($rt is unused):
  $rd = *operation* $rs
- The datapath must be able to perform the load word (lw – reading from RAM) and store word (sw – writing to RAM) operations. These are immediate instructions, and similar to the MIPS lw/sw instructions, a constant can be used to denote an offset. The meaning of these instructions is as follows:
  lw: $rt = MEM[$rs + offset]
  sw: MEM[$rs + offset] = $rt

Examples:

To add the values of register1 and register3, and put them in register0, the following inputs are given:

operation	*your opcode for addition*
rs	01
rt	11
rd	00
offset	*ignored*

To store the value of register0 in memory, two words beyond the address stored in register2:

operation	*your opcode for store word*
rs	10
rt	00
rd	*ignored*
offset	4 *(remember that words are 2 bytes long in our datapath)*