Lu Maggioncalda
27 March 2019

ECE 250 Homework #4: Processor Core Design

Subcircuit name	Inputs	Outputs	Intended behavior
Register	Input - 16-bit input to write into FFs Clk - clock, 0 or 1, connected to main circuit's clock Write Enable - determines where or not we should write the input bits to the FFs Reset - connected to main circuit, sets all FFs to 0	Out - 16-bit output that describes what bits are held in all 16 FFs	Allows you to write a 16-bit value (input) into a register and then see (out) what value is stored in the register. Write enable controls whether or not the input value should be written into the flip flops and reset resets all FFs to 0
RegisterFile	reset- connected to main circuit, sets all registers to 16 bits of 0 valToWrite - 16 bit data to write into a single register Clk - clock, 0 or 1, connected to main circuit's clock WE- writes valToWrite to the specified register if WE is 1. If WE is 0, register values don't change writeReg - 3 bit input that specifies number of register to write to Read1 - 3 bit input that specifies number of first register to read from Read2 - 3 bit input that specifies number of second register to read	readData1 - the value stored in the register specified by Read1 readData2 - the value stored in the register specified by Read2	Allows you to write a value into one register and read values from two registers
ALU	In1a - the first 16-bit number In1b - the second 16-bit number Operation - 3 bit input that specifies the operation to be performed in the ALU	Zero? - returns true if the output is 0000000000000000 (16 0s) Output - 16 bit number that is the result of performing the operation on In1a or	Allows you to take two numbers and add them, subtract them, xor them, shift the first one by the amount specified in the least significant 3 bits of the second input, or allows

	000 is add, 001 is subtract, 010 is not, 011 is xor, 100 is shift left, 101 is shift right	In1a and In2b	you to not the first input, and get the output of that operation
Full adder	Cin - 1 bit carry in A - 1 bit number B - 1 bit number	S - sum of A and B Cout - true if A and B were both 1	Adds two 1-bit numbers and tells you whether there is carry out and what the sum is
isZero	Input - 16 bit number	Out - 1 if all 16 bits were 0, 0 if else	Tells you whether a 16 bit input is all 0s or not
16bitAdder	1a - first 16 bit number 2b - second 16 bit number	Output - is the sum of the two 16 bit numbers Overflow - true if there was carry out from the most significant bit, false if not	Adds two 16 bit inputs together
subtractor	1a - first 16 bit number 2b - second 16 bit number	Out - is equal to 1a - 2b	Subtracts one 16 bit input from another 16 bit input
leftShifter	Input - 16 bit number to shift Amount - number of places to shift by	Output - the Input shifted by Amount places to the left	Performs a logical left shift on Input by Amount places
rightShifter	Input - 16 bit number to shift Amount - number of places to shift by	Output - the Input shifted by Amount places to the right while preserving sign	Performs an arithmetic right shift on Input by Amount places
insnSplitter	Insn - the 16 bit instruction from ROM	Opcode - the first four bits Immediate - the last 6 bits Shamt - the last 3 bits Rd - bits 5-3 Rt - bits 8-6 Rs - bits 11-9 Address - the last 12 bits	Splits the instructions into bits we can use for controlling selectors in the main circuit
controlBox	Opcode - the four most significant bits from the instruction	RfWe - register file write enable. If true, allows to write to the register file dataWE - data write enable. If true, allows to write to RAM	Provides the selector bits for the MUXes in our main circuits that allows us to control which operations are performed and with what inputs

		isLoad – true if the instruction corresponds to a lw useImmediate – true if we will need to use the last 6 bits of the instruction for any reason (addi, lw, sw, shifts) ALU2 – the most significant bit of the operation ALU1 – the middle bit of the operation input ALU0 – the final bit of the operation input isBranch – true if the opcode corresponds to beq or blt isJump – true if opcode corresponds to j or jal isJal – true if opcode corresponds to jal isJr – true if opcode corresponds to Jr readInput – true if opcode corresponds to input printOutput – true if opcode corresponds to output	
RIJ	opcode – the four most significant bits from the instruction	isR – true if the instruction is R type isI – true if the instruction is I type isJ – true if the instruction is J type	Tells us for a given opcode what type of instruction it is
brancher	ALUresult – 16 bit result that is the output of the ALU Opcode – the four most significant bits from the instruction	doBranch – true if blt results in a negative number or if beq results in 16 bits of 0	Tells us whether we should branch. doBranch is the selector for the MUX that chooses between the address to branch to and the regular PC +1