Each instruction influences the data path in a unique way. Our processor by default sets all control signals to 0 (except for PCen, and enRAM/enROM), and therefore, as the control signals are explained, only signals that will need to change state from 0 will be listed for each instruction.

addi:

Addi is an I-Type instruction with an 18-bit, signed immediate value. Because the immediate value is interpreted as a two’s complement number, it can be used to add both positive and negative integer values. When the instruction is decoded, the Operation Code that goes to the Logic Controller is 4'b0101. The destination register is decoded from bits [27:23] of the instruction, and the source register, Rs, is decoded from bits [23:18] of the instruction. The immediate value is decoded from bits [17:0], but is fed into a sign extender (to preserve the two’s complement sign of the number) to be used in computation as a 32-bit number. From the Logic Controller, aluSrcb is set to a 0, which allows the immediate value to pass through the multiplexer determining the value of the second argument into the ALU. The Logic Controller also sets aluop to 3’b000 to perform addition. Thus the ALU utilizes the 32-bit data stored in Rs and the sign extended 32-bit immediate value to perform the add computation. The result of the ALU is passed through a mux controlled by memSrc, which is set to 1 to allow the ALU data to pass through. The data then passes through both the wbSrc and wbPSR muxes till it gets to the RaWriteMUX, where it also passes through. The Logic Controller sets regWriteEn high at the beginning of the instruction, and thus computation can be stored into the destination register. This instruction will have an effect on the C and F flags of the Program Status Register depending on the resulting ALU outcome. Refer to the diagram of the processor and the outlined control signals below for visualization.

*Control Signals:*

CFwrite = 1;

memSrc = 1;

wbSrc = 1;

regWriteEn = 1;

sub:

Sub is an R-Type instruction. Because this is an R-Type instruction, the two arguments are registers that contain data to be subtracted. The Operation Code that goes to the Logic Controller is 4'b0000. The associated decoded Function Code comes from bits [3:0] of the instruction and are 4'b1001 for subtraction. The resulting control signals from the Logic Controller set the ALU to perform subtraction on two signed numbers. Three operands are considered for this instruction: Rdest, bits [27:23] of the instruction; Rs, bits [22:18] of the instruction; and Rt, bits [17:13] of the instruction. The Register File reads out the 32-bit data from Rs, called RsData, and the 32-bit data from Rt, called RtData to be used in the ALU to perform subtraction. The output of the ALU then contains the subtracted value of Rs minus Rt. This value is to be written back into the destination register, Rdest, so it bypasses the data RAM, and goes through a series of multiplexors until the data is ultimately passed into the port see at the Register File for writing data. This instruction will have an effect on the C and F flags of the Program Status Register depending on the resulting ALU outcome. Refer to the diagram of the processor and the outlined control signals below for visualization.

*Control Signals:*

CFWrite = 1'b1;

wbSrc = 1'b1;

memSrc = 1'b1;

aluSrcb = 1'b1;

regWriteEn = 1'b1;

aluop = 3'b001;

multi:

Multi is an I-Type instruction with an 18-bit signed immediate value. The two arguments for the desired operation are contained in the source register field, Rs, and the immediate field. The Operation Code that goes to the Logic Controller is 4'b1110. Because this is an I-Type instruction, there is no associated Function Code. The resulting control signals from the Logic Controller set the ALU to perform multiplication on two signed numbers. The data for the arguments comes from a 32-bit immediate value which is the sign extended immediate value from the instruction, and the 32-bit RsData coming out of the Register File. The two argument data go into the ALU where the resulting output of the ALU contains the multiplied data. The value is to be written back into the destination register specified in the instruction, Rdest, so it bypasses the data RAM, goes through a series of multiplexors and ultimately passes into the port at the Register File for writing data. This instruction has no effect on the Program Status Register. Refer to the diagram of the processor and the outlined control signals below for visualization.

*Control Signals:*

wbSrc <= 1'b1;

memSrc <= 1'b1;

regWriteEn <= 1'b1;

aluop <= 3'b110;

cmpi :

Cmpi is an I-Type instruction with an 18-bit signed immediate value. The two arguments for the desired operation are contained in the source register field, Rs, and the immediate field. The Operation Code that goes into the Logic Controller is 4'b1011. Because this is an I-Type instruction, there is no associated Function Code. The resulting control signals from the Logic Controller set the ALU to perform a comparison on two signed numbers. The data for the arguments comes from the 32-bit sign extended immediate value of the instruction and the 32-bit RsData coming out of the Register File. This comparison is done by performing a subtraction. If the difference of the two values is positive, then Rs is larger than the inputted immediate value. If they are equal, then the Z Flag in the Program Status Register is set high. If the difference is negative, then the N Flag in the Program Status Register is set high. It is important to note that though this instruction is an I-Type instruction, it is undesirable to write back the result into a register; therefore, regWriteEn remains low. The purpose of this instruction is to set the Program Status Register to use for conditional branches, jumps, ect. Refer to the diagram of the processor and the outlined control signals below for visualization.

*Control Signals:*

LZNWrite <= 1'b1;

wbSrc <= 1'b1;

memSrc <= 1'b1;

aluop <= 3'b111;

andi :

andi is an I-Type instruction with an 18-bit immediate value. The two arguments for the desired operation are contained in the source register field, Rs, and the immediate field. The Operation Code that goes into the Logic Controller is 4'b0001. Because this is an I-Type instruction, there is no associated Function Code. The resulting control signals from the Logic controller set the ALU to perform a bitwise AND on the two numbers, therefore, treating them as signed numbers has no effect on this operation. The data for the arguments comes from the 32-bit sign extended immediate value of the instruction and the 32-bit RsData coming out of the Register File. The two argument data go into the ALU where the resulting output of the ALU contains the bitwise ANDed data. The value is to be written back into the destination register specified in the instruction, Rdest, so it bypasses the data RAM, geos through a series of multiplexors and ultimately passes into the port at the Register File for writing data. This instruction has no effect on the Program Status Register. Refer to the diagram of the processor and the outlined control signals below for visualization.

*Control Signals:*

wbSrc <= 1'b1;

memSrc <= 1'b1;

regWriteEn <= 1'b1;

aluop <= 3'b011;

ori :

ori is an I-Type instruction with an 18-bit immediate value. The two arguments for the desired operation are contained in the source register field, Rs, and the immediate field. The Operation Code that goes into the Logic Controller is 4'b0010. Because this is an I-Type instruction, there is no associated Function Code. The resulting control signals from the Logic controller set the ALU to perform a bitwise OR on the two numbers, therefore, treating them as signed numbers has no effect on this operation. The data for the arguments comes from the 32-bit sign extended immediate value of the instruction and the 32-bit RsData coming out of the Register File. The two argument data go into the ALU where the resulting output of the ALU contains the bitwise ORed data. The value is to be written back into the destination register specified in the instruction, Rdest, so it bypasses the data RAM, geos through a series of multiplexors and ultimately passes into the port at the Register File for writing data. This instruction has no effect on the Program Status Register. Refer to the diagram of the processor and the outlined control signals below for visualization.

*Control Signals:*

wbSrc <= 1'b1;

memSrc <= 1'b1;

regWriteEn <= 1'b1;

aluop <= 3'b010;

xori:

xori is an I-Type instruction with an 18-bit immediate value. The two arguments for the desired operation are contained in the source register field, Rs, and the immediate field. The Operation Code that goes into the Logic Controller is 4'b0011. Because this is an I-Type instruction, there is no associated Function Code. The resulting control signals from the Logic controller set the ALU to perform a bitwise XOR on the two numbers, therefore, treating them as signed numbers has no effect on this operation. The data for the arguments comes from the 32-bit sign extended immediate value of the instruction and the 32-bit RsData coming out of the Register File. The two argument data go into the ALU where the resulting output of the ALU contains the bitwise XORed data. The value is to be written back into the destination register specified in the instruction, Rdest, so it bypasses the data RAM, geos through a series of multiplexors and ultimately passes into the port at the Register File for writing data. This instruction has no effect on the Program Status Register. Refer to the diagram of the processor and the outlined control signals below for visualization.

*Control Signals:*

wbSrc <= 1'b1;

memSrc <= 1'b1;

regWriteEn <= 1'b1;

aluop <= 3'b100;