FINAL PROJECT: HELLO MIPS

REPORT

FERHAT ŞİRİN

161044080

Introduction: My processor works as a single cycle processor.It has a single cycle datapath and control unit. When instruction is taken from mips_instr_mem unit at positive edge of clock signal . It is breaked into parts as Opcode, function, rs, rt, rd, 16BitImmed, 26BitJumpAddress by Mips_core unit.Then ControlUnit takes Opcode and function and finds out which instruction it is. Then Sets the necessary control signals like ALUctr, MEMCtr, ShiftCtr etc. to execute the instruction properly. ALU unit takes the ALUctr signal and execute the process like add, sub, slt etc. corresponding to the ALUctr.Shifter unit takes Shiftctr and execute the process like sll (shift left logic), srl, sra corresponding to the Shiftctr. Same goes on for mips_data_mem unit. Mips_data_mem unit takes MemCtr and execute the process like sw, lw, lbu, sb etc. corresponding to the Memctr. The data R[rs] and R[rt] is taken from mips_registers unit at positive edge of clock. When the result data is produced by ALU or Shifter or taken by mips_data_mem unit. If the data is written to the register R[rt] or R[rd] It is written in the unit mips_registers at negative edge of clock.If the data R[rt] is written to the Data Memory then it is handled by mips_data_mem unit like mips_registers at negative edge of clock. Instructions like beq, bne, j, jal etc. are handled by mips_core unit at negative edge of clock.

Instructions this processor can execute : add, addi, addu, addiu, sub, subu, and, andi, beq, bne, j, jal, jr, lbu, lhu, lw, lui, nor, or, ori, slt, slti, sltu, sltiu, sll, srl, sra, sw, sb, sh

Method:

mips_core(clock) : This module connects all other modules to each other. Supply of necessary signals to the other modules. When positive edge of clock comes from mips_testbench. Sends the clock signal to mips_instr_mem unit to take the instruction and breaks into parts.Sends the rs and rt signals to mips_registers to read data from registers.Sends Opcode and funct. code to the controlUnit to take necessary signals to execute instruction properly.After signals taken from ControlUnit sends signals to ALU, Shifter, mips_data_mem to take the result. When negative egde of clock comes from mips_testbench. Send the result data to mips_register to write R[rd] register and mips_data_mem to write memory. It depends on control signals. After that increase ProgramCounter at the negative edge of clock. Instruction like beq, j, jal is handled at this negative edge of clock in mips core unit.

mips_registers (
  output reg [31:0] read_data_1, read_data_2,
  input [31:0] write_data,
  input [4:0] read_reg_1, read_reg_2, write_reg,
  input signal_reg_write, clk
) : This module reads from register and writes to it at negative edge of clock. When instruction fetches from mips_instr_mem unit at the positive edge clock. Mips_core breaks it into parts. Then sends the rs and rt address to mips_registers to read those registers. When intruction execution is done then negative edge of clocks comes and data is written to write_reg register if signal_reg_write is 1.

mips_instr_mem(instruction,program_counter,clk) : This module reads the instruction from instruction memory at the positive egde of clock. Program_counter holds the instruction address to be read.

mips_data_mem (read_data, mem_address, write_data, sig_mem_read, sig_mem_write,MemCtr,clk) : This module reads the data at the positive edge of clock. Instruciton lbu, lhu ans lw is executed at this stage when clock is positive. Data is written to memory at the negative edge of clock . Address is recieved from mem_address . 8 bits at lbu, 16 bits at lhu 16 and 32 bits at lw is taken from memory when sig_mem_read is 1 and completes missing part with zeroes. Instruction sw, sb and sh is executed at this stage when clock is negative. 8 bits at sb, 16 bits at sh and 32 bits at sw is written to data memory when clock is negative and sig_mem_write is 1. MemCtr tells the module what process to do this module.

ALU(result,carry,overFlow,Zerobit,data1,data2,ALUCtr) : This module makes arithmetic operations ALUCtr says. Add, sub, and etc. are executed at this stage. This module is combinational. Produce result, carry, overFlow and ZeroBit. Carry is 1 when unsigned numbers added or subtracted (borrow) and produce extra 1 bit . When operation is unsigned overFlow is not taken in consideration. When signed operation is executed overFlow is 1 when pos + pos = neg or neg + neg = pos at adding opearation or pos – neg =neg neg – pos =pos at subtraction. When operation is signed carry is not taken in consideration. When result is 0 Zerobit is 1 otherwise 0.

shifter(result,reg_data,shamt,shiftCtr) : This module execute shifting operations shiftCtr says. Sll, srl and sra are executed at this stage. This module is combinational. Shamt coming from instruciton specifies the amount of shifting. shifCtr coming from controlUnit specifies the operation. At sll shifts left adds zeroes, at srl shifts right adds zeroes at sra shifts right adds most significant bits of that number.

selectWriteData(writeData,fromALU,fromShifter,fromMem,PC,regSrc,clk) : This module selects the data to be written to the register. Datas coming from ALU, shifter, memory and PC (program_counter) is chosen by regSrc at the positive edge of clock. RegSrc coming from controlUnit decides who is written to register. PC is written to register when jal is executed.

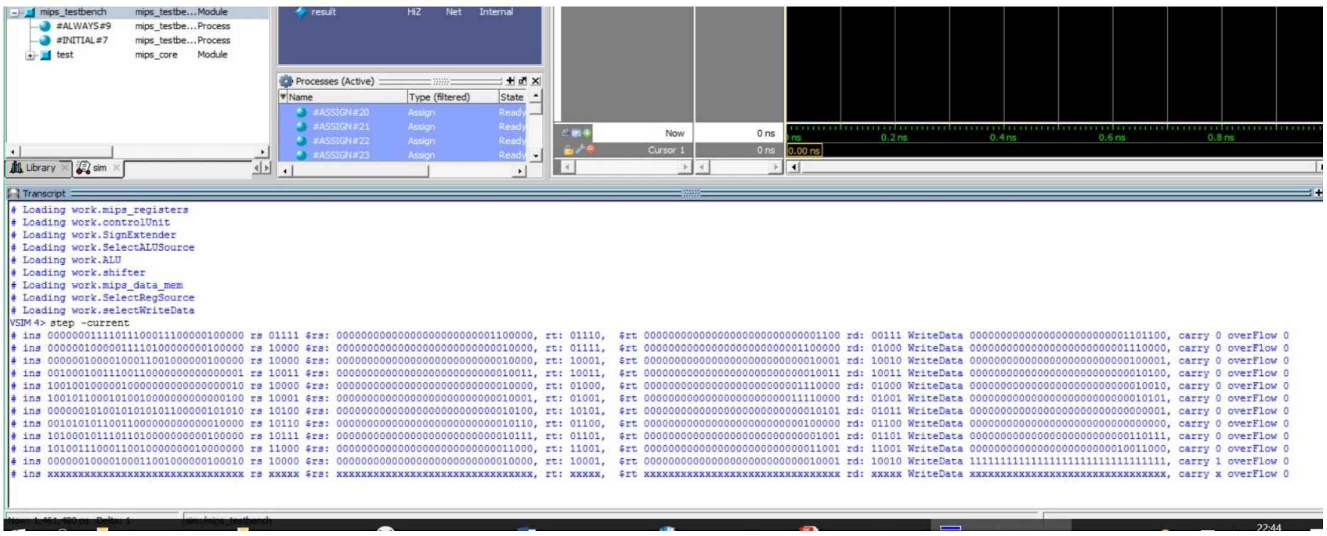
SignExtender(extended,unextended) : This module extends the 16 bits data coming from instruction when instruction is I type. Extends the data according to the most significant bit of data. This data is used when instruction is lw,lbu,lhu, sb, sh, sw for memory address and immidiate type operations.

SelectRegSource(write_reg,Regrt,Regrd,RegDst) : This module selects the register to be written in. When instruction is R type then Regrd is chosen when it is I type then Regrt is chosen. RegDst coming from controlUnit decides whom is chosen.

SelectALUSource(sourceData,fromRt,SignExt,ALUsrc) : This module select the data to be send to ALU. When instruction is R type it is fromRT but when instruction I type it is SignExt coming from SignExtender. ALUsrc coming from controlUnit decides who will go to ALU.

Result:

mips_testbench () : Testbench produces clock signal. Signal changes its status in every 10 ps and sends it to controlUnit. It works in always block therefore it doesn't have a break statement. It will stop when it starts reading instruction like xxxxxxxx.



```
# Loading work.mips_registers
# Loading work.controlUnit
# Loading work.SignExtender
# Loading work.SelectALUSource
# Loading work.ALU
# Loading work.shifter
# Loading work.mips_data_mem
# Loading work.SelectRegSource
# Loading work.selectWriteData
VSIM 4> step -current
# ins 00000001111011100011100000100000 rs 01111 $rs: 00000000000000000000000001100000, rt: 01110,  $rt 00000000000000000000000000001100 rd: 00111 WriteData 00000000000000000000000001101100, carry 0 overFlow 0
# ins 00000010000011110100000000100000 rs 10000 $rs: 00000000000000000000000000010000, rt: 01111,  $rt 00000000000000000000000001100000 rd: 01000 WriteData 00000000000000000000000001110000, carry 0 overFlow 0
# ins 00000010000100011001000000100000 rs 10000 $rs: 00000000000000000000000000010000, rt: 10001,  $rt 00000000000000000000000000010001 rd: 10010 WriteData 00000000000000000000000000100001, carry 0 overFlow 0
# ins 00100010011100110000000000000001 rs 10011 $rs: 00000000000000000000000000010011, rt: 10011,  $rt 00000000000000000000000000010011 rd: 10011 WriteData 00000000000000000000000000010100, carry 0 overFlow 0
# ins 10010010000010000000000000000010 rs 10000 $rs: 00000000000000000000000000010000, rt: 01000,  $rt 00000000000000000000000001110000 rd: 01000 WriteData 00000000000000000000000000010010, carry 0 overFlow 0
# ins 10010110001010010000000000000100 rs 10001 $rs: 00000000000000000000000000010001, rt: 01001,  $rt 00000000000000000000000011110000 rd: 01001 WriteData 00000000000000000000000000010101, carry 0 overFlow 0
# ins 00000010100101010101100000101010 rs 10100 $rs: 00000000000000000000000000010100, rt: 10101,  $rt 00000000000000000000000000010101 rd: 01011 WriteData 00000000000000000000000000000001, carry 0 overFlow 0
# ins 00101010110011000000000000010000 rs 10110 $rs: 00000000000000000000000000010110, rt: 01100,  $rt 00000000000000000000000000100000 rd: 01100 WriteData 00000000000000000000000000000000, carry 0 overFlow 0
# ins 10100010111011010000000000100000 rs 10111 $rs: 00000000000000000000000000010111, rt: 01101,  $rt 00000000000000000000000000001001 rd: 01101 WriteData 00000000000000000000000000110111, carry 0 overFlow 0
# ins 10100111000110010000000010000000 rs 11000 $rs: 00000000000000000000000000011000, rt: 11001,  $rt 00000000000000000000000000011001 rd: 11001 WriteData 00000000000000000000000010011000, carry 0 overFlow 0
# ins 00000010000100011001000000100010 rs 10000 $rs: 00000000000000000000000000010000, rt: 10001,  $rt 00000000000000000000000000010001 rd: 10010 WriteData 11111111111111111111111111111111, carry 1 overFlow 0
# ins xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx rs xxxxx $rs: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, rt: xxxxx,  $rt xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx rd: xxxxx WriteData xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, carry x overFlow 0
```

(Picture is in the file)

1) 00000001111011100011100000100000 R type add
Rs =01111 Rt = 01110 Rd =00111
R[00111] =R[01111] + R[01110]
2) 00000010000011110100000000100000 R type add
Rs = 10000 Rt = 01111 Rd =01000
3) 00000010000100011001000000100000 R type add
Rs =10000 Rt =10001 Rd =10010
4) 00100010011100110000000000000001 I type addi
Rs =10011 Rt =10011 Immed = 0000000000000001
R[10011] =Rs +SignExtendImmed
5) 10010010000010000000000000000010 I type lbu
Rs =10000 Rt =01000 Immed =0000000000000010
R[rt] =Mem[Rs+SignExtendImmed][7:0];
6) 10010110001010010000000000000100 I type lhu
Rs =10001 Rt =01001 Immed =0000000000000100
R[rt] =Mem[Rs+SignExtendImmed][15:0];
7) 00000010100101010101100000101010 R type slt
Rs =10100 Rt =10101 Rd =01011
İf (R[rs] < R[rt]) R[rd] =1 otherwise R[rd] =0
8) 00101010110011000000000000010000 I type slti
Rs =10110 Rt =01100 Immed =0000000000010000
İf(R[rs] < SignExtendedImmed) R[rt] =1 otherwise R[rt] =0
9) 10100010111011010000000000100000 I type sb
Rs =10111 Rt =01101 Immed =0000000000100000
Mem[Rs+SignExtendImmed][7:0] =R[rt][7:0]
10) 10100111000110010000000010000000 I type sh
Rs =11000 Rt =11001 Immed =000000010000000
Mem[Rs + SignExtendImmed][15:0] =R[rt][15:0]
11) 00000010000100011001000000100010 R type sub
Rs =10000 Rt =10001 Rd =10010
R[rd] =R[rs] –R[rt]