Adrian Ruiz

CPE 404

Assignment 3

# Time Spent

10 Hours

# Verilog Code

Mipspart.v:

//------------------------------------------------

// mipsparts.v

// David\_Harris@hmc.edu 23 October 2005

// Components used in MIPS processor

//------------------------------------------------

**module** regfile**(input** clk**,**

**input** we3**,**

**input** **[**4**:**0**]** ra1**,** ra2**,** wa3**,**

**input** **[**31**:**0**]** wd3**,**

**output** **[**31**:**0**]** rd1**,** rd2**);**

**reg** **[**31**:**0**]** rf**[**31**:**0**];**

// three ported register file

// read two ports combinationally

// write third port on rising edge of clock

// register 0 hardwired to 0

**always** **@(posedge** clk**)**

**if** **(**we3**)** rf**[**wa3**]** **<=** wd3**;**

**assign** rd1 **=** **(**ra1 **!=** 0**)** **?** rf**[**ra1**]** **:** 0**;**

**assign** rd2 **=** **(**ra2 **!=** 0**)** **?** rf**[**ra2**]** **:** 0**;**

**endmodule**

**module** adder**(input** **[**31**:**0**]** a**,** b**,**

**output** **[**31**:**0**]** y**);**

**assign** y **=** a **+** b**;**

**endmodule**

**module** sl2**(input** **[**31**:**0**]** a**,**

**output** **[**31**:**0**]** y**);**

// shift left by 2

**assign** y **=** **{**a**[**29**:**0**],** 2'b00**};**

**endmodule**

**module** signext**(input** **[**15**:**0**]** a**,**

**output** **[**31**:**0**]** y**);**

**assign** y **=** **{{**16**{**a**[**15**]}},** a**};**

**endmodule**

**module** flopr **#(parameter** WIDTH **=** 8**)**

**(input** clk**,** reset**,**

**input** **[**WIDTH**-**1**:**0**]** d**,**

**output** **reg** **[**WIDTH**-**1**:**0**]** q**);**

**always** **@(posedge** clk**,** **posedge** reset**)**

**if** **(**reset**)** q **<=** 0**;**

**else** q **<=** d**;**

**endmodule**

**module** flopenr **#(parameter** WIDTH **=** 8**)**

**(input** clk**,** reset**,**

**input** en**,**

**input** **[**WIDTH**-**1**:**0**]** d**,**

**output** **reg** **[**WIDTH**-**1**:**0**]** q**);**

**always** **@(posedge** clk**,** **posedge** reset**)**

**if** **(**reset**)** q **<=** 0**;**

**else** **if** **(**en**)** q **<=** d**;**

**endmodule**

**module** mux2 **#(parameter** WIDTH **=** 8**)**

**(input** **[**WIDTH**-**1**:**0**]** d0**,** d1**,**

**input** s**,**

**output** **[**WIDTH**-**1**:**0**]** y**);**

**assign** y **=** s **?** d1 **:** d0**;**

**endmodule**

// upimm module needed for LUI

**module** upimm**(input** **[**15**:**0**]** a**,**

**output** **[**31**:**0**]** y**);**

**assign** y **=** **{**a**,** 16'b0**};**

**endmodule**

// mux3 needed for LUI

**module** mux3 **#(parameter** WIDTH **=** 8**)**

**(input** **[**WIDTH**-**1**:**0**]** d0**,** d1**,** d2**,**

**input** **[**1**:**0**]** s**,**

**output** **[**WIDTH**-**1**:**0**]** y**);**

**assign** **#**1 y **=** s**[**1**]** **?** d2 **:** **(**s**[**0**]** **?** d1 **:** d0**);**

**endmodule**

//4 to 1 Mux

**module** mux4 **#(parameter** WIDTH **=** 8**)**

**(input** **[**WIDTH**-**1**:**0**]** d0**,** d1**,** d2**,** d3**,**

**input** **[**1**:**0**]** s**,**

**output** **[**WIDTH**-**1**:**0**]** y**);**

**assign** **#**1 y **=** s**[**1**]** **?** **(**s**[**0**]** **?** d3 **:** d2**)** **:** **(**s**[**0**]** **?** d1**:** d0**)** **;**

**endmodule**

// Zero Extend Module needed for ANDI

**module** zeroext **(input** **[**15**:**0**]** a**,**

**output** **[**31**:**0**]** y**);**

**assign** y **=** **{**16'b0**,** a**};**

**endmodule**

**module** signext32**(input** **[**7**:**0**]** a**,**

**output** **[**31**:**0**]** y**);**

**assign** y **=** **{{**24**{**a**[**3**]}},** a**};**

**endmodule**

I added a four to one multiplexor, a zero extender, and a sign extender for an 8-bit value. The multiplexor is used to determine which source is going into the ALU (RT, Sign Extended Immediate, LUI, or Zero Extended Immediate) and which byte is going to loading into a register. The new sign extender is used for load byte.

ALU.v

//------------------------------------------------

// alu32.v

// Sarah\_Harris@hmc.edu 23 October 2005

// 32-bit ALU used by MIPS single-cycle processor

//------------------------------------------------

`timescale 1ns **/** 1ps

**module** alu**(** **input** **[**31**:**0**]** A**,** B**,**

**input** **[**3**:**0**]** F**,** **input** **[**4**:**0**]** shamt**,** // SLL

**output** **reg** **[**31**:**0**]** Y**,** **output** Zero**,**

**output** ltez**);** // BLEZ

**wire** **[**31**:**0**]** S**,** Bout**;**

**assign** Bout **=** F**[**3**]** **?** **~**B **:** B**;**

**assign** S **=** A **+** Bout **+** F**[**3**];** // SLL

**always** **@** **(** **\*** **)**

**case** **(**F**[**2**:**0**])**

3'b000**:** Y **<=** A **&** Bout**;**

3'b001**:** Y **<=** A **|** Bout**;**

3'b010**:** Y **<=** S**;**

3'b011**:** Y **<=** S**[**31**];**

3'b100**:** Y **<=** **(**Bout **<<** shamt**);** // SLL

**endcase**

**assign** Zero **=** **(**Y **==** 32'b0**);**

**assign** ltez **=** Zero **|** S**[**31**];** // BLEZ

// assign Overflow = A[31]& Bout[31] & ~Y[31] |

// ~A[31] & ~Bout[31] & Y[31];

**endmodule**

No Modifications

MipsMem.v

//------------------------------------------------

// mipsmem.v

// David\_Harris@hmc.edu 23 October 2005

// External memories used by MIPS single-cycle

// processor

//------------------------------------------------

**module** dmem**(input** clk**,** we**,**

**input** **[**31**:**0**]** a**,** wd**,**

**output** **[**31**:**0**]** rd**);**

**reg** **[**31**:**0**]** RAM**[**63**:**0**];**

**assign** rd **=** RAM**[**a**[**31**:**2**]];** // word aligned

**always** **@(posedge** clk**)**

**if** **(**we**)**

RAM**[**a**[**31**:**2**]]** **<=** wd**;**

**endmodule**

**module** imem**(input** **[**5**:**0**]** a**,**

**output** **[**31**:**0**]** rd**);**

**reg** **[**31**:**0**]** RAM**[**63**:**0**];**

**initial**

**begin**

$readmemh**(**"memfile2.dat"**,**RAM**);**

**end**

**assign** rd **=** RAM**[**a**];** // word aligned

**endmodule**

Changed the memfile

Tops.V

//-------------------------------------------------

// top.v

// Sarah\_Harris@hmc.edu 23 October 2005

// Top-level module for single-cycle MIPS processor

//-------------------------------------------------

**module** topsingle**(input** clk**,** reset**,**

**input** **[**31**:**0**]** readdata**,**

**output** **[**31**:**0**]** writedata**,** dataadr**,**

**output** memwrite**);**

**wire** **[**31**:**0**]** pc**,** instr**;**

// instantiate processor and memories

mipssingle mipssingle**(**clk**,** reset**,** pc**,** instr**,** memwrite**,** dataadr**,**

writedata**,** readdata**);**

imem imem**(**pc**[**7**:**2**],** instr**);**

dmem dmem**(**clk**,** memwrite**,** dataadr**,** writedata**,**

readdata**);**

**endmodule**

Mipssingle.v

//------------------------------------------------

// mipssingle.v

// Sarah\_Harris@hmc.edu 22 June 2007

// Single-cycle MIPS processor

//------------------------------------------------

// single-cycle MIPS processor

**module** mipssingle**(input** clk**,** reset**,**

**output** **[**31**:**0**]** pc**,**

**input** **[**31**:**0**]** instr**,**

**output** memwrite**,**

**output** **[**31**:**0**]** aluresult**,** writedata**,**

**input** **[**31**:**0**]** readdata**);**

**wire** memtoreg**;**

**wire** **[**1**:**0**]** alusrc**;** // LUI

**wire** **[**1**:**0**]** regdst**;** // JAL

**wire** regwrite**,** jump**,** pcsrc**,** zero**;**

**wire** **[**3**:**0**]** alucontrol**;** // SLL

**wire** ltez**;** // BLEZ

**wire** jal**;** // JAL

**wire** lh**;** // LH

**wire** lb**;** //LB

**wire** jr**;** //JR

controller c**(**instr**[**31**:**26**],** instr**[**5**:**0**],** zero**,**

memtoreg**,** memwrite**,** pcsrc**,**

alusrc**,** regdst**,** regwrite**,** jump**,**

alucontrol**,**

ltez**,** // BLEZ

jal**,** // JAL

lh**,**

lb**,**

jr**);** // LH

datapath dp**(**clk**,** reset**,** memtoreg**,** pcsrc**,**

alusrc**,** regdst**,** regwrite**,** jump**,**

alucontrol**,**

zero**,** pc**,** instr**,**

aluresult**,** writedata**,** readdata**,**

ltez**,** // BLEZ

jal**,** // JAL

lh**,**

lb**,**

jr**);** // LH

**endmodule**

**module** controller**(input** **[**5**:**0**]** op**,** funct**,**

**input** zero**,**

**output** memtoreg**,** memwrite**,**

**output** pcsrc**,**

**output** **[**1**:**0**]** alusrc**,** // LUI

**output** **[**1**:**0**]** regdst**,** // JAL

**output** regwrite**,**

**output** jump**,**

**output** **[**3**:**0**]** alucontrol**,** // 4 bits for SLL

**input** ltez**,** // BLEZ

**output** jal**,** // JAL

**output** lh**,**

**output** lb**,**

**output** jr**);** // LH

**wire** **[**2**:**0**]** aluop**;**

**wire** branch**;**

**wire** blez**;** // BLEZ

maindec md**(**op**,** funct**,** memtoreg**,** memwrite**,** branch**,**

alusrc**,** regdst**,** regwrite**,** jump**,**

aluop**,** blez**,** jal**,** lh**,** lb**,**jr**);** // BLEZ, JAL, LH

aludec ad**(**funct**,** aluop**,** alucontrol**);**

**assign** pcsrc **=** **(**branch **&** zero**)** **|** **(**blez **&** ltez**);** // BLEZ

**endmodule**

**module** maindec**(input** **[**5**:**0**]** op**,** funct**,**

**output** memtoreg**,** memwrite**,**

**output** branch**,**

**output** **[**1**:**0**]** alusrc**,** // LUI

**output** **[**1**:**0**]** regdst**,** // JAL

**output** regwrite**,**

**output** jump**,**

**output** **[**2**:**0**]** aluop**,**

**output** blez**,** // BLEZ

**output** jal**,** // JAL

**output** lh**,**

**output** lb**,**

**output** jr**);** // LH

**reg** **[**16**:**0**]** controls**;** // increase controls for LUI, BLEZ, JAL, LH, LB, JR

**assign** **{**regwrite**,** regdst**,** alusrc**,**

branch**,** memwrite**,**

memtoreg**,** jump**,** aluop**,**

blez**,** // BLEZ

jal**,** // JAL

lh**,**

lb**,**

jr**}** // LH

**=** controls**;**

**always** **@(\*)**

**case(**op**)**

6'b000000**:** **if** **(**funct **==** 6'b001000**)**

controls **<=** 17'b00000000000000001**;** //JR

**else**

controls **<=** 17'b10100000011000000**;** //Rtype

6'b100011**:** controls **<=** 17'b10001001000000000**;** //LW

6'b101011**:** controls **<=** 17'b00001010000000000**;** //SW

6'b000100**:** controls **<=** 17'b00000100000100000**;** //BEQ

6'b001000**:** controls **<=** 17'b10001000000000000**;** //ADDI

6'b000010**:** controls **<=** 17'b00000000100000000**;** //J

6'b001010**:** controls **<=** 17'b10001000001100000**;** //SLTI

6'b001111**:** controls **<=** 17'b10010000000000000**;** //LUI

6'b000110**:** controls **<=** 17'b00000000000110000**;** //BLEZ

6'b000011**:** controls **<=** 17'b11000000100001000**;** //JAL

6'b100001**:** controls **<=** 17'b10001001000000100**;** //LH

6'b001100**:** controls **<=** 17'b10011000010000000**;** //ANDI

6'b100000**:** controls **<=** 17'b10001001000000010**;** //LB

**default:** controls **<=** 17'bxxxxxxxxxxxxxxxxx**;** //???

**endcase**

**endmodule**

**module** aludec**(input** **[**5**:**0**]** funct**,**

**input** **[**2**:**0**]** aluop**,**

**output** **reg** **[**3**:**0**]** alucontrol**);** // 4-bits for SLL

**always** **@(\*)**

**case(**aluop**)**

3'b000**:** alucontrol **<=** 4'b0010**;** // add

3'b001**:** alucontrol **<=** 4'b1010**;** // sub

3'b011**:** alucontrol **<=** 4'b1011**;** // slt

3'b100**:** alucontrol **<=** 4'b0000**;** //and

**default:** **case(**funct**)** // RTYPE

6'b100000**:** alucontrol **<=** 4'b0010**;** // ADD

6'b100010**:** alucontrol **<=** 4'b1010**;** // SUB

6'b100100**:** alucontrol **<=** 4'b0000**;** // AND

6'b100101**:** alucontrol **<=** 4'b0001**;** // OR

6'b101010**:** alucontrol **<=** 4'b1011**;** // SLT

6'b000000**:** alucontrol **<=** 4'b0100**;** // SLL

**default:** alucontrol **<=** 4'bxxxx**;** // ???

**endcase**

**endcase**

**endmodule**

**module** datapath**(input** clk**,** reset**,**

**input** memtoreg**,** pcsrc**,**

**input** **[**1**:**0**]** alusrc**,** // LUI

**input** **[**1**:**0**]** regdst**,** // JAL

**input** regwrite**,** jump**,**

**input** **[**3**:**0**]** alucontrol**,** // SLL

**output** zero**,**

**output** **[**31**:**0**]** pc**,**

**input** **[**31**:**0**]** instr**,**

**output** **[**31**:**0**]** aluresult**,** writedata**,**

**input** **[**31**:**0**]** readdata**,**

**output** ltez**,** // BLEZ

**input** jal**,** // JAL

**input** lh**,**

**input** lb**,**

**input** jr**);** // LH

**wire** **[**4**:**0**]** writereg**;**

**wire** **[**31**:**0**]** pcnext**,** pcnextbr**,** pcplus4**,** pcbranch**;**

**wire** **[**31**:**0**]** signimm**,** signimmsh**;**

**wire** **[**31**:**0**]** upperimm**;** // LUI

**wire** **[**31**:**0**]** srca**,** srcb**;**

**wire** **[**31**:**0**]** result**;**

**wire** **[**31**:**0**]** writeresult**;** // JAL

**wire** **[**15**:**0**]** half**;** // LH

**wire** **[**31**:**0**]** signhalf**,** memdata**;** // LH

**wire** **[**31**:**0**]** zeroimm**,** byteImm**,** SelectBtye**,** SelectData**,** pcJR**;** // Wires added for new instructions

// next PC logic

flopr **#(**32**)** pcreg**(**clk**,** reset**,** pcnext**,** pc**);**

adder pcadd1**(**pc**,** 32'b100**,** pcplus4**);**

sl2 immsh**(**signimm**,** signimmsh**);**

adder pcadd2**(**pcplus4**,** signimmsh**,** pcbranch**);**

mux2 **#(**32**)** pcbrmux**(**pcplus4**,** pcbranch**,** pcsrc**,**

pcnextbr**);**

//Mux for JR

mux2 **#(**32**)** pcjrmux**(**pcnextbr**,** srca**,** jr**,** pcJR**);**

mux2 **#(**32**)** pcmux**(**pcJR**,** **{**pcplus4**[**31**:**28**],**

instr**[**25**:**0**],** 2'b00**},**

jump**,** pcnext**);**

// register file logic

regfile rf**(**clk**,** regwrite**,** instr**[**25**:**21**],**

instr**[**20**:**16**],** writereg**,**

writeresult**,** // JAL

srca**,** writedata**);**

mux2 **#(**32**)** wamux**(**result**,** pcplus4**,** jal**,** writeresult**);** // JAL

mux3 **#(**5**)** wrmux**(**instr**[**20**:**16**],** instr**[**15**:**11**],** 5'd31**,**

regdst**,** writereg**);** // JAL

// hardware to support LH

mux2 **#(**16**)** lhmux1**(**readdata**[**15**:**0**],** readdata**[**31**:**16**],**

aluresult**[**1**],** half**);** // LH

signext lhse**(**half**,** signhalf**);** // LH

mux2 **#(**32**)** lhmux2**(**readdata**,** signhalf**,** lh**,** memdata**);** // LH

//Selecting which byte for LB

mux4 **#(**32**)** lbmux**(**memdata**[**7**:**0**],** memdata**[**15**:**8**],** memdata**[**23**:**16**],** memdata**[**31**:**24**],**aluresult**[**1**:**0**]** **,** SelectBtye**);**

//Selecting weather to pass the byte or the whole word

mux2 **#(**32**)** bytemux**(**memdata**,** SelectBtye**,** lb**,** SelectData**);**

//Aluresult or Data from memory

mux2 **#(**32**)** resmux**(**aluresult**,** SelectData**,** memtoreg**,** result**);** // LH

signext se**(**instr**[**15**:**0**],** signimm**);**

upimm ui**(**instr**[**15**:**0**],** upperimm**);** // LUI

// Extenders for LB and ANDI

signext32 seLB**(**memdata**[**7**:**0**],** byteImm**);**

zeroext ze**(**instr**[**15**:**0**],** zeroimm**);**

// ALU logic

mux4 **#(**32**)** srcbmux**(**writedata**,** signimm**,** upperimm**,** zeroimm**,** alusrc**,**

srcb**);** // LUI

alu alu**(**srca**,** srcb**,** alucontrol**,** instr**[**10**:**6**],** // SLL

aluresult**,** zero**,** ltez**);** // BLEZ

**endmodule**

I added two control signals to the control unit (LB, JR) and made aluop into 3 bits. Aluop was made into 3 bits to allow for ANDI. LB determines whether the whole word or the byte will pass through the mux. JR determines whether the addresses in a register is taken. I added the three instructions into decoders. The zero extender and the sign extender for 8 bits is also added to the Datapath. Two 2:1 muxes where added for LB and JR. Two 4:1 muxes were added for alusrc, and LB.

Mipstest.v

//------------------------------------------------

// mipstest.v

// Sarah\_Harris@hmc.edu 23 October 2005

// Testbench for single-cycle MIPS processor

//------------------------------------------------

**module** testbench**();**

**reg** clk**;**

**reg** reset**;**

**wire** **[**31**:**0**]** aluout**,** writedata**,** readdata**;**

**wire** memwrite**;**

// instantiate device to be tested

topsingle dut**(**clk**,** reset**,** readdata**,** writedata**,**

aluout**,** memwrite**);**

// initialize test

**initial**

**begin**

reset **<=** 1**;** **#** 22**;** reset **<=** 0**;**

**end**

// generate clock to sequence tests

**always**

**begin**

clk **<=** 1**;** **#** 5**;** clk **<=** 0**;** **#** 5**;**

**end**

// check results

**always@(negedge** clk**)**

**begin**

**if(**memwrite **&** aluout **==** 44**)** **begin** // Change to see if 0XFE is written im mem

**if(**writedata **==** 254**)**

$display**(**"Simulation succeeded"**);**

**else** **begin**

$display**(**"Simulation failed"**);**

**end**

$stop**;**

**end**

**end**

**endmodule**

# Memdat2.asm

# memfile2.asm

# Dr. Harris **-** 7 Feb 2019

#

# **Test** the MIPS processor.

# Instructions**:** **add,** **sub,** **and,** **or,** slt**,** addi**,** lw**,** sw**,** beq**,** j

# Added instructions**:** andi**,** lb**,** jr

#

# Fill **in** the Description column below

#

# Assembly Description Address Machine

**.text**

.globl main

main**:** addi $2**,** $0**,** 0x732c #$2 **=** 29484 0 2002732c

addi $3**,** $0**,** **-**32768 #$3 **=** **-**32768 4 20038000

**add** $4**,** $3**,** $3 #$4 **=-**32768 **+** **-**32768 **=** **-**65536 8 00632020

**add** $4**,** $4**,** $3 #$4 **=-**65536 **+** **-** 32768**=** **-**98304 **c** 00832020

**sub** $4**,** $4**,** $2 #$4 **=** **-**98304 **-** 29484 **=** **-**127788 10 00822022

andi $3**,** $4**,** 0x897F #$3 **=** FFFE0cd4 **AND** 0000897f **=** 0x854 14 3083897f

sw $4**,** 32**(**$0**)** #mem**[**32**]** **=** 0xFFFE0CD4 18 ac040020

lb $6**,** 34**(**$0**)** #$6**=** 0xFE 1c 80060022

andi $3**,** $3**,** 0xEF #$3 **=** 0x44 20 306300ef

jr $3 #j to address **in** $3 0x44 is PC 24 00600008

**add** $0**,** $0**,** $0 #**not** executed 28 00000020

andi $6**,** $6**,** 0x123 #**not** executed 2c 30c60123

**add** $6**,** $6**,** $6 #**not** executed 30 00c63020

addi $6**,** $6**,** **-**578 #**not** executed 34 20c6fdbe

addi $6**,** $6**,** 77 #**not** executed 38 20c6004d

**sub** $6**,** $2**,** $6 #**not** executed 3c 00463022

**sub** $6**,** $2**,** $6 #**not** executed 40 00463022

sw $6**,** 12**(**$3**)** #mem**[**0x44**]** **=** 0xFE **or** 433 44 ac66000c

jr $3 #j to address **in** $3 0X44 48 00600008

# Waveforms

From top to bottom, the signals are : clk, reset, PC+4, PC, Aluout, MemData, ReadData, Memwrite

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated