

UNIVERSITY OF WATERLOO AQ  
Faculty of Engineering   
      Department of Electrical and Computer Engineering         
ECE 621- Computer Organization

**Memory and Writeback**

Group 2

**Jordan *Ross,* Mustafa Faraj**   
j25ross, mabdulaz 14 November, 2014

# Processor Module Code

The only module that we made changes for this PD was to the processor module. We also created new modules for the pipeline registers between the execute and memory stage as well as between the memory and writeback stage. We show all the processor code below. Each instruction takes 5 cycles to complete before the next instruction is fetch, decoded, etc. In order to do this we implemented a simple counter.

**module** processor**(**

**input** clk**,** // The system clock

**input** srec\_parse // If the SREC parser is active or not.

**);**

**reg** **[**2**:**0**]**cur\_pipe\_state**;**

**reg** **[**2**:**0**]**next\_pipe\_state**;**

// Decoder signals

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

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

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

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

**wire** **[**5**:**0**]**func**;**

**wire** **[**15**:**0**]**immed**;**

**wire** **[**25**:**0**]**target**;**

**wire** **[**5**:**0**]**opcode**;**

**wire** **[**31**:**0**]**pc\_out**;**

**wire** **[**31**:**0**]**insn\_out**;**

// Control signals

**reg** stall**;**

**wire** **[**1**:**0**]**insn\_access\_size**;**

**wire** **[**1**:**0**]**fetch\_access\_size**;**

**wire** fetch\_rw**;**

**wire** insn\_rw**;**

**reg** reg\_file\_write\_enable**;**

// Address lines

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

// Data lines

**wire** **[**31**:**0**]**insn\_data\_out**;**

**wire** **[**31**:**0**]**insn\_address**;**

// SREC registers (only used for helping the parser write to instruction/data memory)

**reg** **[**31**:**0**]**srec\_address**;**

**reg** **[**31**:**0**]**srec\_data\_in**;**

**reg** srec\_rw**;**

**reg** **[**1**:**0**]**srec\_access\_size**;**

// Fetch wires

**wire** **[**31**:**0**]**fetch\_next\_pc**;**

// Decode wires

**wire** **[**31**:**0**]**decode\_pc**;** // We define this as the PC for next instruction to be executed

**wire** **[**31**:**0**]**decode\_ir**;** // We define this as the current instruction being decoded

**wire** **[**31**:**0**]**dec\_A**;**

**wire** **[**31**:**0**]**dec\_B**;**

**wire** **[**4**:**0**]**dest\_reg**;**

// Decode control signals

**reg** dec\_dest\_reg\_sel**;**

**reg** dec\_illegal\_insn**;**

**reg** **[**5**:**0**]** dec\_alu\_op**;**

**reg** dec\_is\_branch**;**

**reg** dec\_op2\_sel**;**

**reg** **[**5**:**0**]** dec\_shift\_amount**;**

**reg** **[**1**:**0**]**dec\_branch\_type**;** // 0-BEQ, 1-BNE, 2-BLEZ, 3-BGTZ

**reg** dec\_is\_jump**;**

**reg** **[**1**:**0**]**dec\_access\_size**;**

**reg** dec\_rw**;**

**reg** dec\_memory\_sign\_extend**;**

**reg** dec\_res\_data\_sel**;**

**reg** dec\_write\_to\_reg**;**

**reg** **[**4**:**0**]**dec\_rt**;**

**reg** **[**4**:**0**]**dec\_rd**;**

**reg** dec\_is\_jal**;**

**reg** dec\_is\_jr**;**

// Execute wires

**wire** **[**31**:**0**]**exe\_pc**;** // We define this as the PC for next instruction to be executed

**wire** **[**31**:**0**]**exe\_ir**;** // We define this as the current instruction being executied

**wire** **[**31**:**0**]**exe\_A**;**

**wire** **[**31**:**0**]**exe\_B**;**

**wire** **[**31**:**0**]**exe\_op2**;**

**wire** **[**31**:**0**]**exe\_O**;**

**wire** **[**31**:**0**]**exe\_extended**;** // The wire the comes for the sign extender

**wire** exe\_zero**;**

**wire** **[**5**:**0**]** exe\_alu\_op**;**

**wire** exe\_is\_branch**;**

**wire** exe\_op2\_sel**;**

**wire** **[**5**:**0**]** exe\_shift\_amount**;**

**wire** **[**31**:**0**]** exe\_shift\_immed**;**

**wire** **[**31**:**0**]** exe\_shift\_target**;**

**wire** **[**31**:**0**]** exe\_jump\_effective\_address**;**

**wire** **[**31**:**0**]** exe\_branch\_effective\_address**;**

**wire** exe\_neg**;**

**wire** **[**1**:**0**]**exe\_branch\_type**;**

**wire** exe\_branch\_taken**;**

**wire** **[**31**:**0**]**exe\_next\_pc**;**

**wire** **[**31**:**0**]**exe\_branch\_next\_pc**;**

**wire** exe\_is\_jump**;**

**wire** **[**1**:**0**]**exe\_access\_size**;**

**wire** exe\_rw**;**

**wire** exe\_memory\_sign\_extend**;**

**wire** exe\_res\_data\_sel**;**

**wire** exe\_write\_to\_reg**;**

**wire** exe\_dest\_reg\_sel**;**

**wire** **[**4**:**0**]**exe\_rt**;**

**wire** **[**4**:**0**]**exe\_rd**;**

**wire** exe\_update\_pc**;**

**wire** exe\_is\_jal**;**

**wire** **[**31**:**0**]**exe\_alu\_result**;**

**wire** **[**31**:**0**]**exe\_jump\_next\_pc**;**

**wire** exe\_is\_jr**;**

// Memory wires

**wire** **[**31**:**0**]**mem\_next\_pc**;**

**wire** **[**31**:**0**]**mem\_alu\_result**;**

**wire** **[**31**:**0**]**mem\_reg\_data**;**

**wire** **[**1**:**0**]**mem\_data\_access\_size**;**

**wire** mem\_data\_rw**;**

**wire** **[**31**:**0**]**mem\_data\_out**;**

**wire** **[**31**:**0**]**mem\_data\_in**;**

**wire** **[**31**:**0**]**mem\_addr\_in**;**

**wire** **[**1**:**0**]**mem\_access\_size**;**

**wire** mem\_rw**;**

**wire** mem\_memory\_sign\_extend**;**

**wire** **[**31**:**0**]**mem\_sign\_extend\_out**;**

**wire** **[**4**:**0**]**mem\_rt**;**

**wire** **[**4**:**0**]**mem\_rd**;**

**wire** mem\_res\_data\_sel**;**

**wire** mem\_write\_to\_reg**;**

**wire** mem\_dest\_reg\_sel**;**

**wire** mem\_update\_pc**;**

**wire** mem\_is\_jal**;**

// Write back wires

**wire** **[**31**:**0**]**wb\_O**;**

**wire** **[**31**:**0**]**wb\_D**;**

**wire** wb\_res\_data\_sel**;**

**wire** wb\_write\_to\_reg**;**

**wire** wb\_dest\_reg\_sel**;**

**wire** **[**31**:**0**]**wb\_data**;**

**wire** **[**4**:**0**]**wb\_rt**;**

**wire** **[**4**:**0**]**wb\_rd**;**

**wire** **[**31**:**0**]**wb\_next\_pc**;**

**wire** wb\_update\_pc**;**

**wire** **[**4**:**0**]**wb\_dest\_reg**;**

**wire** wb\_is\_jal**;**

//--------------------------- FETCH STAGE -----------------------------------------//

// Instantiate mux's for each of the SREC registers to aid the SREC parser.

mux\_2\_1\_32\_bit srec\_insn\_address\_mux**(**

**.**line0**(**pc**),**

**.**line1**(**srec\_address**),**

**.**select**(**srec\_parse**),**

**.**output\_line**(**insn\_address**)**

**);**

mux\_2\_1\_1\_bit srec\_insn\_rw\_mux**(**

**.**line0**(**fetch\_rw**),**

**.**line1**(**srec\_rw**),**

**.**select**(**srec\_parse**),**

**.**output\_line**(**insn\_rw**)**

**);**

mux\_2\_1\_2\_bit srec\_insn\_access\_size\_mux**(**

**.**line0**(**fetch\_access\_size**),**

**.**line1**(**srec\_access\_size**),**

**.**select**(**srec\_parse**),**

**.**output\_line**(**insn\_access\_size**)**

**);**

// Instantiate the fetch module

fetch fetch**(**

**.**clk\_in**(**clk**),**

**.**stall\_in**(**stall**),**

**.**pc\_in**(**wb\_next\_pc**),**

**.**update\_pc**(**wb\_update\_pc**),**

**.**pc\_out**(**pc**),**

**.**next\_pc**(**fetch\_next\_pc**),**

**.**rw\_out**(**fetch\_rw**),**

**.**access\_size\_out**(**fetch\_access\_size**)**

**);**

// Instantiate the instruction memory module

memory insn\_memory**(**

**.**data\_out**(**decode\_ir**),**

**.**address**(**insn\_address**),**

**.**data\_in**(**srec\_data\_in**),** // We can tie the srec\_data\_in wire to this port since we should never be writing to instruction memory unless we are srec parsing

**.**write**(**insn\_rw**),**

**.**clk**(**clk**),**

**.**access\_size**(**insn\_access\_size**)**

**);**

// ------------------------------ DECODE STAGE --------------------------------------//

// Instatiate the IF/ID pipeline register to kep the PC and IR

if\_id\_pipleline\_reg if\_id\_pipleline\_reg**(**

**.**clk**(**clk**),**

**.**pc\_in**(**fetch\_next\_pc**),**

//.ir\_in(insn\_data\_out), // We don't need to latch the ir because it won't be available until next clock cycle

**.**pc\_out**(**decode\_pc**)**

//.ir\_out(decode\_ir)

**);**

// Instantiate the register file

reg\_file reg\_file**(**

**.**clk**(**clk**),**

**.**write\_enable**(**reg\_file\_write\_enable**),**

**.**source1**(**rs**),**

**.**source2**(**rt**),**

**.**dest**(**dest\_reg**),**

**.**destVal**(**wb\_data**),**

**.**s1val**(**dec\_A**),**

**.**s2val**(**dec\_B**)**

**);**

// Instantiate the decode module

decode decoder**(**

**.**clk**(**clk**),**

**.**stall**(**stall**),**

**.**insn\_in**(**decode\_ir**),**

**.**pc\_in**(**pc**),** // TODO: I don't see what this is needed

**.**rs**(**rs**),**

**.**rt**(**rt**),**

**.**rd**(**rd**),**

**.**sha**(**sha**),**

**.**func**(**func**),**

**.**immed**(**immed**),**

**.**target**(**target**),**

**.**opcode**(**opcode**),**

**.**pc\_out**(**pc\_out**),** // TODO: I don't see why this is needed

**.**insn\_out**(**insn\_out**)**

**);**

// ------------------------------ EXECUTE STAGE --------------------------------------//

// Instatiate the ID/IX pipeline register

id\_ix\_pipleline\_reg id\_ix\_pipleline\_reg**(**

**.**clk**(**clk**),**

**.**pc\_in**(**decode\_pc**),**

**.**ir\_in**(**decode\_ir**),**

**.**A\_in**(**dec\_A**),**

**.**B\_in**(**dec\_B**),**

**.**alu\_op\_in**(**dec\_alu\_op**),**

**.**is\_branch\_in**(**dec\_is\_branch**),**

**.**is\_jump\_in**(**dec\_is\_jump**),**

**.**op2\_sel\_in**(**dec\_op2\_sel**),**

**.**shift\_amount\_in**(**dec\_shift\_amount**),**

**.**branch\_type\_in**(**dec\_branch\_type**),**

**.**access\_size\_in**(**dec\_access\_size**),**

**.**rw\_in**(**dec\_rw**),**

**.**memory\_sign\_extend\_in**(**dec\_memory\_sign\_extend**),**

**.**res\_data\_sel\_in**(**dec\_res\_data\_sel**),**

**.**rt\_in**(**dec\_rt**),**

**.**rd\_in**(**dec\_rd**),**

**.**dest\_reg\_sel\_in**(**dec\_dest\_reg\_sel**),**

**.**write\_to\_reg\_in**(**dec\_write\_to\_reg**),**

**.**is\_jal\_in**(**dec\_is\_jal**),**

**.**is\_jr\_in**(**dec\_is\_jr**),**

**.**pc\_out**(**exe\_pc**),**

**.**ir\_out**(**exe\_ir**),**

**.**A\_out**(**exe\_A**),**

**.**B\_out**(**exe\_B**),**

**.**alu\_op\_out**(**exe\_alu\_op**),**

**.**is\_branch\_out**(**exe\_is\_branch**),**

**.**is\_jump\_out**(**exe\_is\_jump**),**

**.**op2\_sel\_out**(**exe\_op2\_sel**),**

**.**shift\_amount\_out**(**exe\_shift\_amount**),**

**.**branch\_type\_out**(**exe\_branch\_type**),**

**.**access\_size\_out**(**exe\_access\_size**),**

**.**rw\_out**(**exe\_rw**),**

**.**memory\_sign\_extend\_out**(**exe\_memory\_sign\_extend**),**

**.**res\_data\_sel\_out**(**exe\_res\_data\_sel**),**

**.**rt\_out**(**exe\_rt**),**

**.**rd\_out**(**exe\_rd**),**

**.**dest\_reg\_sel\_out**(**exe\_dest\_reg\_sel**),**

**.**write\_to\_reg\_out**(**exe\_write\_to\_reg**),**

**.**is\_jal\_out**(**exe\_is\_jal**),**

**.**is\_jr\_out**(**exe\_is\_jr**)**

**);**

sign\_extender sign\_extender**(**

**.**in\_data**(**exe\_ir**[**15**:**0**]),**

**.**out\_data**(**exe\_extended**)**

**);**

**assign** exe\_shift\_target **=** exe\_ir**[**25**:**0**];**

**assign** exe\_jump\_effective\_address **=** **(**exe\_pc **&** 32'hf0000000**)** **|** **((**exe\_shift\_target **<<** 2**)** **&** 32'h0fffffff**);**

**assign** exe\_shift\_immed **=** exe\_extended **<<** 2**;**

**assign** exe\_branch\_effective\_address **=** exe\_shift\_immed **+** exe\_pc**;**

// Instantiate a 32-bit mux for selecting which operand to provide to op2 of the ALU

mux\_2\_1\_32\_bit alu\_op2\_sel\_mux**(**

**.**line0**(**exe\_B**),**

**.**line1**(**exe\_extended**),**

**.**select**(**exe\_op2\_sel**),**

**.**output\_line**(**exe\_op2**)**

**);**

alu alu**(**

**.**op1**(**exe\_A**),** // operand 1 (always from rs)

**.**op2**(**exe\_op2**),** // operand 2

**.**operation**(**exe\_alu\_op**),** // The arithmatic operation to perform

**.**shift\_amount**(**exe\_shift\_amount**),** // The number of bits to shift

**.**result**(**exe\_alu\_result**),** // The arithmatic result based on the operation

**.**zero**(**exe\_zero**),** // Indicates if the result of the operation is zero.

**.**neg**(**exe\_neg**)**

**);**

// Instantiate a mux to pass the execution operation result as pc+4 for the JAL instruction

mux\_2\_1\_32\_bit store\_pc\_mux**(**

**.**line0**(**exe\_alu\_result**),**

**.**line1**(**exe\_pc**),**

**.**select**(**exe\_is\_jal**),**

**.**output\_line**(**exe\_O**)**

**);**

branch\_resolve branch\_resolve**(**

**.**zero**(**exe\_zero**),**

**.**neg**(**exe\_neg**),**

**.**branch\_type**(**exe\_branch\_type**),**

**.**is\_branch**(**exe\_is\_branch**),**

**.**branch\_taken**(**exe\_branch\_taken**)** // Indicates if the branch is taken or not

**);**

mux\_2\_1\_32\_bit branch\_next\_pc\_mux**(**

**.**line0**(**exe\_pc**),**

**.**line1**(**exe\_branch\_effective\_address**),**

**.**select**(**exe\_branch\_taken**),**

**.**output\_line**(**exe\_branch\_next\_pc**)**

**);**

mux\_2\_1\_32\_bit jump\_next\_pc\_mux**(**

**.**line0**(**exe\_branch\_next\_pc**),**

**.**line1**(**exe\_jump\_effective\_address**),**

**.**select**(**exe\_is\_jump**),**

**.**output\_line**(**exe\_jump\_next\_pc**)**

**);**

mux\_2\_1\_32\_bit jump\_ret\_next\_pc\_mux**(**

**.**line0**(**exe\_jump\_next\_pc**),**

**.**line1**(**exe\_O**),**

**.**select**(**exe\_is\_jr**),**

**.**output\_line**(**exe\_next\_pc**)**

**);**

**assign** exe\_update\_pc **=** exe\_is\_jump **|** exe\_branch\_taken **|** exe\_is\_jr**;**

// ------------------------------ MEMORY STAGE --------------------------------------//

// Instatiate the IX/IM pipeline register

ix\_im\_pipleline\_reg ix\_im\_pipleline\_reg**(**

**.**clk**(**clk**),**

**.**pc\_in**(**exe\_next\_pc**),**

**.**O\_in**(**exe\_O**),**

**.**B\_in**(**exe\_B**),**

**.**access\_size\_in**(**exe\_access\_size**),**

**.**rw\_in**(**exe\_rw**),**

**.**memory\_sign\_extend\_in**(**exe\_memory\_sign\_extend**),**

**.**res\_data\_sel\_in**(**exe\_res\_data\_sel**),**

**.**rt\_in**(**exe\_rt**),**

**.**rd\_in**(**exe\_rd**),**

**.**dest\_reg\_sel\_in**(**exe\_dest\_reg\_sel**),**

**.**write\_to\_reg\_in**(**exe\_write\_to\_reg**),**

**.**update\_pc\_in**(**exe\_update\_pc**),**

**.**is\_jal\_in**(**exe\_is\_jal**),**

**.**pc\_out**(**mem\_next\_pc**),**

**.**O\_out**(**mem\_alu\_result**),** // This will be the R-type data to write or EA for mem

**.**B\_out**(**mem\_reg\_data**),**

**.**access\_size\_out**(**mem\_data\_access\_size**),**

**.**rw\_out**(**mem\_data\_rw**),**

**.**memory\_sign\_extend\_out**(**mem\_memory\_sign\_extend**),**

**.**res\_data\_sel\_out**(**mem\_res\_data\_sel**),**

**.**rt\_out**(**mem\_rt**),**

**.**rd\_out**(**mem\_rd**),**

**.**dest\_reg\_sel\_out**(**mem\_dest\_reg\_sel**),**

**.**write\_to\_reg\_out**(**mem\_write\_to\_reg**),**

**.**update\_pc\_out**(**mem\_update\_pc**),**

**.**is\_jal\_out**(**mem\_is\_jal**)**

**);**

// We have some muxes here to write the data memory during SREC parsing

mux\_2\_1\_32\_bit srec\_address\_mux**(**

**.**line0**(**mem\_alu\_result**),**

**.**line1**(**srec\_address**),**

**.**select**(**srec\_parse**),**

**.**output\_line**(**mem\_addr\_in**)**

**);**

mux\_2\_1\_32\_bit srec\_data\_data\_mux**(**

**.**line0**(**mem\_reg\_data**),**

**.**line1**(**srec\_data\_in**),**

**.**select**(**srec\_parse**),**

**.**output\_line**(**mem\_data\_in**)**

**);**

mux\_2\_1\_1\_bit srec\_data\_rw\_mux**(**

**.**line0**(**mem\_data\_rw**),**

**.**line1**(**srec\_rw**),**

**.**select**(**srec\_parse**),**

**.**output\_line**(**mem\_rw**)**

**);**

mux\_2\_1\_2\_bit srec\_data\_access\_size\_mux**(**

**.**line0**(**mem\_data\_access\_size**),**

**.**line1**(**srec\_access\_size**),**

**.**select**(**srec\_parse**),**

**.**output\_line**(**mem\_access\_size**)**

**);**

// Instantiate the data memory module

memory data\_memory**(**

**.**data\_out**(**mem\_data\_out**),**

**.**address**(**mem\_addr\_in**),**

**.**data\_in**(**mem\_data\_in**),**

**.**write**(**mem\_rw**),**

**.**clk**(**clk**),**

**.**access\_size**(**mem\_access\_size**)**

**);**

// Instantiate the sign extender for loading half words and bytes

memory\_sign\_extender memory\_sign\_extender**(**

**.**in\_data**(**mem\_data\_out**),**

**.**data\_size**(**mem\_access\_size**),**

**.**sign\_extend**(**mem\_memory\_sign\_extend**),**

**.**out\_data**(**wb\_D**)** //.out\_data(mem\_sign\_extend\_out) TODO: This may throw some issues

//when we pipleine... It will be the same for handling the fetch

//and decode memory handoff

**);**

// ------------------------------ WRITE BACK STAGE --------------------------------------//

// Instatiate the IM/IW pipeline register

im\_iw\_pipleline\_reg im\_iw\_pipleline\_reg**(**

**.**clk**(**clk**),**

**.**pc\_in**(**mem\_next\_pc**),**

**.**O\_in**(**mem\_alu\_result**),**

//.D\_in(mem\_sign\_extend\_out),

**.**res\_data\_sel\_in**(**mem\_res\_data\_sel**),**

**.**write\_to\_reg\_in**(**mem\_write\_to\_reg**),**

**.**dest\_reg\_sel\_in**(**mem\_dest\_reg\_sel**),**

**.**rt\_in**(**mem\_rt**),**

**.**rd\_in**(**mem\_rd**),**

**.**update\_pc\_in**(**mem\_update\_pc**),**

**.**is\_jal\_in**(**mem\_is\_jal**),**

**.**pc\_out**(**wb\_next\_pc**),**

**.**O\_out**(**wb\_O**),**

//.D\_out(wb\_D),

**.**res\_data\_sel\_out**(**wb\_res\_data\_sel**),**

**.**write\_to\_reg\_out**(**wb\_write\_to\_reg**),**

**.**dest\_reg\_sel\_out**(**wb\_dest\_reg\_sel**),**

**.**rt\_out**(**wb\_rt**),**

**.**rd\_out**(**wb\_rd**),**

**.**update\_pc\_out**(**wb\_update\_pc**),**

**.**is\_jal\_out**(**wb\_is\_jal**)**

**);**

// Mux for selecting between which data we should be writing back to the register

mux\_2\_1\_32\_bit wb\_data\_mux**(**

**.**line0**(**wb\_O**),**

**.**line1**(**wb\_D**),**

**.**select**(**wb\_res\_data\_sel**),**

**.**output\_line**(**wb\_data**)**

**);**

// // Instantiate a mux for selecting which destination to choose

mux\_2\_1\_5\_bit dest\_reg\_mux**(**

**.**line0**(**wb\_rd**),**

**.**line1**(**wb\_rt**),**

**.**select**(**wb\_dest\_reg\_sel**),**

**.**output\_line**(**wb\_dest\_reg**)**

**);**

// Instantiate a mux for selecting between the destination register in the previous

// mux or selecting the return address register (31) if we have a JAL

mux\_2\_1\_5\_bit sel\_ra\_reg\_mux**(**

**.**line0**(**wb\_dest\_reg**),**

**.**line1**(**5'd31**),**

**.**select**(**wb\_is\_jal**),**

**.**output\_line**(**dest\_reg**)**

**);**

//assign dest\_reg = (wb\_dest\_reg\_sel) ? (wb\_rd) : (wb\_rt)

// Control

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

**case** **(**cur\_pipe\_state**)**

3'b000 **:** **begin**

next\_pipe\_state **=** 3'b001**;**

stall **=** 1**;**

**end**

3'b001 **:** **begin**

next\_pipe\_state **=** 3'b010**;**

// ----------- Decode Stage Control Signal Logic --------------------------- //

dec\_illegal\_insn **=** 0**;**

dec\_dest\_reg\_sel **=** 0**;**

dec\_alu\_op **=** 0**;**

dec\_op2\_sel **=** 0**;**

dec\_is\_branch **=** 0**;**

dec\_is\_jump **=** 0**;**

dec\_branch\_type **=** 0**;**

dec\_rw **=** 0**;**

dec\_access\_size **=** 0**;**

dec\_memory\_sign\_extend **=** 0**;**

dec\_res\_data\_sel **=** 0**;**

dec\_write\_to\_reg **=** 1**;**

dec\_rt **=** rt**;**

dec\_rd **=** rd**;**

dec\_is\_jal **=** 0**;**

dec\_is\_jr **=** 0**;**

**if** **(((**opcode **&** 6'b111000**)>>** 3**)** **==** 3'h0**)** **begin**

**if** **((**opcode **&** 6'b000111**)** **==** 3'h0**)** **begin**

// We are in the SPECIAL Opcode encoding table

// This is an R-type instruction

dec\_dest\_reg\_sel **=** 0**;**

**if** **(**func **==** 6'b100000 **||** func **==** 6'b100001**)** // ADD, ADDU

dec\_alu\_op **=** 0**;** // Do an add operation

**else** **if** **(**func **==** 6'b100010 **||** func **==** 6'b100011**)** // SUB, SUBU

dec\_alu\_op **=** 1**;**

**else** **if** **(**func **==** 6'b011000 **||** func **==** 6'b011001**)** // MULT, MULTU

dec\_alu\_op **=** 2**;**

**else** **if** **(**func **==** 6'b011010 **||** func **==** 6'b011011**)** // DIV, DIVU

dec\_alu\_op **=** 3**;**

**else** **if** **(**func **==** 6'b000000**)** **begin** // SLL

dec\_alu\_op **=** 4**;**

dec\_shift\_amount **=** sha**;**

**end**

**else** **if** **(**func **==** 6'b000010**)** **begin** // SLL

dec\_alu\_op **=** 5**;**

dec\_shift\_amount **=** sha**;**

**end**

**else** **if** **(**func **==** 6'b000011**)** **begin** // SRA

dec\_alu\_op **=** 11**;**

dec\_shift\_amount **=** sha**;**

dec\_illegal\_insn **=** 1**;**

**end**

**else** **if** **(**func **==** 6'b101010 **||** func **==** 6'b101011**)** **begin**// SLT, SLTU

dec\_alu\_op **=** 6**;**

**end**

**else** **if** **(**func **==** 6'b001000**)** **begin** // JR

dec\_rt **=** 0**;**

dec\_alu\_op **=** 0**;** // Essentially just get the return register as the execution output by ading it with the zero reg

dec\_is\_jr **=** 1**;**

dec\_write\_to\_reg **=** 0**;**

**end**

**else** **begin**

dec\_illegal\_insn **=** 1**;**

**end**

**end** **else** **if** **(((**opcode **&** 6'b000111**)** **==** 3'd2**)** **||** **((**opcode **&** 6'b000111**)** **==** 3'd3**))** **begin**

// J and JAL instructions

// This is J-type instruction

dec\_dest\_reg\_sel **=** 0**;**

dec\_is\_jump **=** 1**;**

**if** **(**opcode **==** 6'b000011**)** **begin**

// JAL instruction

dec\_res\_data\_sel **=** 0**;**

dec\_is\_jal **=** 1**;**

dec\_write\_to\_reg **=** 1**;**

**end**

**else** **begin**

dec\_write\_to\_reg **=** 0**;**

**end**

**end** **else** **begin**

// BEQ, BNE, BLEZ, BGTZ

// This is an I-type instruction

dec\_is\_branch **=** 1**;**

dec\_write\_to\_reg **=** 0**;**

**if** **(**opcode **==** 6'b000100**)** **begin**

// BEQ

dec\_alu\_op **=** 1**;** // We want to do a test if the result is zero from a subtract

dec\_branch\_type **=** 0**;**

**end** **else** **if** **(**opcode **==** 6'b000101**)** **begin**

// BNE

dec\_alu\_op **=** 1**;** // We want to do a test if the result is zero from a subtract

dec\_branch\_type **=** 1**;**

**end** **else** **if** **(**opcode **==** 6'b000110**)** **begin**

// BLEZ

// TODO: Test that rt has zero in it

dec\_illegal\_insn **=** 1**;**

dec\_alu\_op **=** 0**;**

dec\_branch\_type **=** 2**;**

**end** **else** **begin**

// BGTZ

// TODO: Test that rt has zero in it

dec\_illegal\_insn **=** 1**;**

dec\_alu\_op **=** 0**;**

dec\_branch\_type **=** 3**;**

**end**

**end**

**end** **else** **if** **(((**opcode **&** 6'b111000**)>>** 3**)** **==** 3'd1**)** **begin**

// ADDI, ADDIU, SLTI, SLTIU, ANDI, ORI, XORI, LUI

// This is an I-type instruction

**case** **(**opcode**)**

6'b001000**:** dec\_alu\_op **=** 0**;** // ADDI

6'b001001**:** dec\_alu\_op **=** 0**;** // ADDIU

6'b001010**:** dec\_alu\_op **=** 6**;** // SLTI

6'b001011**:** dec\_alu\_op **=** 6**;** // SLTUI

6'b001100**:** dec\_alu\_op **=** 7**;** // ANDI

6'b001101**:** dec\_alu\_op **=** 8**;** // ORI

6'b001110**:** dec\_alu\_op **=** 9**;** // XORI

6'b001111**:** dec\_alu\_op **=** 12**;** // LUI

**endcase**

dec\_write\_to\_reg **=** 1**;**

dec\_dest\_reg\_sel **=** 1**;**

dec\_op2\_sel **=** 1**;** // We want op2 to be the immediate in the ALU

**end** **else** **if** **(((**opcode **&** 6'b111000**)>>** 3**)** **==** 3'd3**)** **begin**

// No idea?

dec\_dest\_reg\_sel **=** 0**;**

dec\_illegal\_insn **=** 1**;**

**end** **else** **if** **(((**opcode **&** 6'b111000**)>>** 3**)** **==** 3'd4**)** **begin**

// LB, LH, LWL, LW, LBU, LHU, LWR

// This is an I-type instruction

dec\_dest\_reg\_sel **=** 1**;**

dec\_op2\_sel **=** 1**;**

dec\_alu\_op **=** 0**;** // We want to add the value of rs to the immediate

dec\_res\_data\_sel **=** 1**;**

**case** **(**opcode**)**

6'b100000 **:** **begin**

dec\_access\_size **=** 0**;** //LB

dec\_memory\_sign\_extend **=** 1**;**

**end**

6'b100001 **:** **begin**

dec\_access\_size **=** 1**;** //LH

dec\_memory\_sign\_extend **=** 1**;**

**end**

6'b100010 **:** dec\_access\_size **=** 2**;** //LWL

6'b100011 **:** dec\_access\_size **=** 2**;** //LW

6'b100100 **:** dec\_access\_size **=** 0**;** //LBU

6'b100101 **:** dec\_access\_size **=** 1**;** //LHU

6'b100110 **:** dec\_access\_size **=** 2**;** //LWR

**default** **:** dec\_illegal\_insn **=** 1**;**

**endcase**

**end** **else** **if** **(((**opcode **&** 6'b111000**)>>** 3**)** **==** 3'd5**)** **begin**

// SB, SH, SWL, SW, SWR

// This is an I-type instruction

dec\_dest\_reg\_sel **=** 1**;**

dec\_op2\_sel **=** 1**;**

dec\_alu\_op **=** 0**;** // We want to add the value of rs to the immediate

dec\_rw **=** 1**;** // We want to write to memory when we store

dec\_write\_to\_reg **=** 0**;**

**case** **(**opcode**)**

6'b101000 **:** dec\_access\_size **=** 0**;** //SB

6'b101001 **:** dec\_access\_size **=** 1**;** //SH

6'b101010 **:** dec\_illegal\_insn **=** 1**;** //dec\_access\_size = 2; //SWL

6'b101011 **:** dec\_access\_size **=** 2**;** //SW

6'b101100 **:** dec\_access\_size **=** 0**;** //SBU

6'b101101 **:** dec\_access\_size **=** 1**;** //SHU

6'b101110 **:** dec\_illegal\_insn **=** 1**;** //dec\_access\_size = 2; //SWR

**default** **:** dec\_illegal\_insn **=** 1**;**

**endcase**

**end** **else** **begin**

dec\_illegal\_insn **=** 1**;**

**end**

**end**

3'b010 **:** next\_pipe\_state **=** 3'b011**;**

3'b011 **:** **begin**

next\_pipe\_state **=** 3'b100**;**

**end**

3'b100 **:** **begin**

next\_pipe\_state **=** 3'b000**;**

stall **=** 0**;**

**end**

**endcase**

//reg\_file\_write\_enable = 0; // Clear the write enable in case we wrote to reg file on neg edge

**end**

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

**if** **(**srec\_parse **==** 0**)** **begin**

cur\_pipe\_state **=** next\_pipe\_state**;**

// case (cur\_pipe\_state)

// 3'b100 : begin

// // ----------------- WRITE BACK CONTROL LOGIC -------------------------------- //

// if (wb\_write\_to\_reg == 1) begin

// // We need to write the the register so we should set the read write enable and

// // select the destination reigster then feed the data to the input port.

// reg\_file\_write\_enable = 1;

// end

// end

// endcase

**end**

**end**

**always** **@(**cur\_pipe\_state **or** wb\_write\_to\_reg**)** **begin**

**if** **(**cur\_pipe\_state **==** 3'b100 **&&** wb\_write\_to\_reg **==** 1'b1**)** **begin**

reg\_file\_write\_enable **=** 1**;**

**end**

**else** **begin**

reg\_file\_write\_enable **=** 0**;**

**end**

**end**

**endmodule**

# Processor Testbench Code

The processor test bench did not change very much from the last PD. The only change we made was we initialized the stack pointer register to be a large enough amount of bytes away from the initial contents of the data memory. We did not initialize it to the last aligned address in this PD to show that the data memory has the correct initialized contents as well as the correct contents after execution without multiple screenshots. The other modification we made to the code was initializing the counter in the processor module.

**module** processor\_tb**;**

**reg** clk**;**

**reg** srec\_parse**;** // control signal for if the srec parser is active or not.

// Registers, writes, and variables for parser

**integer** fh **=** 0**;** // file handler for output

**integer** i **=** 0**;** // loop variable

**integer** data\_byte **=** 0**;** // variable to keep track of what byte we are on.

**integer** data\_offset **=** 0**;** // keep track of the offset from the data address to write the next byte.

**reg** **[**1**:**0**]**nibble\_count **=** 0**;** // keep track of which nibble is being written (upper/lower).

**reg** **[**7**:**0**]**rec\_type**;** // record type number

**reg** **[**7**:**0**]** byte\_count**;** // the number of bytes for the address, data, and checksum

**integer** record\_code**;** // A record\_code is equivalent to 1 ASCII digit/letter in the .srec file

**reg** **[**31**:**0**]** rec\_address **=** 'b0**;** // the address given by the record.

**integer** highest\_address **=** 0**;**

**reg** **[**7**:**0**]**rec\_data**;** // A single byte of the data from the record.

**reg** **[**7**:**0**]**temp**;** // a temporary byte used for place holding.

**reg** done **=** 0**;** // this will set high when we are done parsing the file.

**reg** **[**7**:**0**]** file\_char **=** 8'h0A**;** // Set the initial character from the file read to be the new line character

// Instantiate the processor as the unit under test.

processor processor\_uut**(.**clk**(**clk**),** **.**srec\_parse**(**srec\_parse**));**

**initial** **begin**

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

// Parsing stage of testbench - memory is not valid until

// after the parser has finished!

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

// Before we begin parsing we want to make sure the fetch module does nothing until the the memory has been

// populated with instructions. We will do this by setting the stall\_in to 1 so nothing happens.

processor\_uut**.**stall **=** 1**;**

srec\_parse **=** 1**;**

$monitor**(**"Starting the SREC parser..."**);**

// Open the SREC file to read

//fh = $fopen("D:/Git\_Repositories/ECE621\_PiplinedProcessor/BubbleSort.srec", "r");

fh **=** $fopen**(**"D:/GitHub/ECE621\_PiplinedProcessor/CheckVowel.srec"**,** "r"**);**

//fh = $fopen("D:/Dropbox/Grad/01 Fall14/ECE621/Labs/L1/code/ECE621\_PiplinedProcessor/BubbleSort.srec", "r");

// Start the clock high

clk **=** 0**;**

// loop until we set the done bit

**while** **(**done **==** 0**)** **begin**

**#**100**;** // Delay 1 clock cycle.

// Read the first/next character from the file.

file\_char **=** $fgetc**(**fh**);**

**if** **(**file\_char **==** 8'hff**)** **begin**

done **=** 1**;**

file\_char **=** 8'h0A**;**

**end**

// Reset the record byte which keeps track of the current byte of the line you are reading in.

// This is equivalent to 1 ASCII code from the file.

record\_code **=** 0**;**

// Loop until we reach a new line character which signifies a new record.

**while** **(**file\_char **!=** 8'h0A**)** **begin**

**#**50**;** // Delay 1/2 clock cycle.

highest\_address **=** **(**rec\_address **>** highest\_address**)?** rec\_address**:** highest\_address**;**

**if** **(**record\_code **==** 0**)** **begin**

// Clear out all the bit fields.

rec\_type **=** 8'h4**;**

byte\_count **=** 16'h0**;**

rec\_address **=** 32'h0**;**

rec\_data **=** 132'h0**;**

data\_offset **=** 0**;**

data\_byte **=** 0**;**

**end** **else** **if** **(**record\_code **==** 1**)** **begin**

// read the record type.

rec\_type**[**7**:**0**]** **=** atoh**(**file\_char**);**

**end** **else** **if** **(**record\_code **==** 2**)** **begin**

// read the upper byte of the byte count.

temp **=** atoh**(**file\_char**);**

byte\_count**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 3**)** **begin**

// read the lower byte of the byte count.

temp **=** atoh**(**file\_char**);**

byte\_count**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **>** 3**)** **begin**

**if** **(**rec\_type **==** 1**)** **begin** // If the record type is for a 16 bit address.

rec\_address**[**31**:**16**]** **=** 16'h0000**;**

**if** **(**record\_code **==** 4**)** **begin**

// read the middle byte of the address.

temp **=** atoh**(**file\_char**);**

// remove the upper most nibble since we only have single digits to represent memory addresses

rec\_address**[**15**:**12**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 5**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**11**:**8**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 6**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 7**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **begin**

// Check to see if we have reached the end of the data

**if** **(**data\_byte **<** byte\_count **-** 2 **-** 1**)** **begin** // Make sure we are less than the byte count minus the address size in bytes and checksum

// We are reading data so we want to create a lower and an upper nibble of a byte then write it to memory when we have both.

temp **=** atoh**(**file\_char**);**

rec\_data **=** rec\_data **<<** 4**;**

rec\_data**[**3**:**0**]** **=** temp**[**3**:**0**];**

nibble\_count **=** nibble\_count **+** 1**;**

**#**50**;**

**if** **(**nibble\_count **>** 1**)** **begin**

// We have both nibbles so we should write the byte to memory

// set all the lines on the falling edge of the clock.

processor\_uut**.**srec\_address **=** rec\_address**+**data\_offset**;**

processor\_uut**.**srec\_data\_in **=** rec\_data**;**

processor\_uut**.**srec\_access\_size **=** 2'b01**;**

processor\_uut**.**srec\_rw **=** 1**;**

**#**100**;** // Delay one clock cycle

processor\_uut**.**srec\_rw **=** 0**;**

// update the data\_offest.

data\_offset **=** data\_offset **+** 1**;**

// reset the nibble count

nibble\_count **=** 0**;**

data\_byte **=** data\_byte **+** 1**;**

**end**

**end**

**end**

**end**

**if** **(**rec\_type **==** 2**)** **begin** // If the record type is for a 24 bit address.

rec\_address**[**31**:**24**]** **=** 8'h00**;**

**if** **(**record\_code **==** 4**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**23**:**20**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 5**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**19**:**16**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 6**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**15**:**12**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 7**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**11**:**8**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 8**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 9**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **begin**

// Check to see if we have reached the end of the data

**if** **(**data\_byte **<** byte\_count **-** 3 **-** 1**)** **begin** // Make sure we are less than the byte count minus the address size in bytes and checksum

// We are reading data so we want to create a lower and an upper nibble of a byte then write it to memory when we have both.

temp **=** atoh**(**file\_char**);**

rec\_data **=** rec\_data **<<** 4**;**

rec\_data**[**3**:**0**]** **=** temp**[**3**:**0**];**

nibble\_count **=** nibble\_count **+** 1**;**

**#**50**;**

**if** **(**nibble\_count **>** 1**)** **begin**

// We have both nibbles so we should write the byte to memory

// set all the lines on the falling edge of the clock.

processor\_uut**.**srec\_address **=** rec\_address**+**data\_offset**;**

processor\_uut**.**srec\_data\_in **=** rec\_data**;**

processor\_uut**.**srec\_access\_size **=** 2'b00**;**

processor\_uut**.**srec\_rw **=** 1**;**

**#**100**;** // Delay one clock cycle

processor\_uut**.**srec\_rw **=** 0**;**

// update the data\_offest.

data\_offset **=** data\_offset **+** 1**;**

// reset the nibble count

nibble\_count **=** 0**;**

data\_byte **=** data\_byte **+** 1**;**

**end**

**end**

**end**

**end**

**if** **(**rec\_type **==** 3**)** **begin** // If the record type is for a 32 bit address.

**if** **(**record\_code **==** 4**)** **begin**

// read the upper most byte of the address.

temp **=** atoh**(**file\_char**);**

// remove the upper most nibble since we only have single digits to represent memory addresses

rec\_address**[**31**:**28**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 5**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**27**:**24**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 6**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**23**:**20**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 7**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**19**:**16**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 8**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**15**:**12**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 9**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**11**:**8**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 10**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 11**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **begin**

// Check to see if we have reached the end of the data

**if** **(**data\_byte **<** byte\_count **-** 4 **-** 1**)** **begin** // Make sure we are less than the byte count minus the address size in bytes and checksum

// We are reading data so we want to create a lower and an upper nibble of a byte then write it to memory when we have both.

temp **=** atoh**(**file\_char**);**

rec\_data **=** rec\_data **<<** 4**;**

rec\_data**[**3**:**0**]** **=** temp**[**3**:**0**];**

nibble\_count **=** nibble\_count **+** 1**;**

**#**50**;**

**if** **(**nibble\_count **>** 1**)** **begin**

// We have both nibbles so we should write the byte to memory

// set all the lines on the falling edge of the clock.

processor\_uut**.**srec\_address **=** rec\_address**+**data\_offset**;**

processor\_uut**.**srec\_data\_in **=** rec\_data**;**

processor\_uut**.**srec\_access\_size **=** 2'b00**;**

processor\_uut**.**srec\_rw **=** 1**;**

**#**100**;** // Delay one clock cycle

processor\_uut**.**srec\_rw **=** 0**;**

// update the data\_offest.

data\_offset **=** data\_offset **+** 1**;**

// reset the nibble count

nibble\_count **=** 0**;**

data\_byte **=** data\_byte **+** 1**;**

**end**

**end**

**end**

**end**

**end**

**#**50**;** // delay 1/2 clock cycle

// increment record\_code

record\_code **=** record\_code **+** 1**;**

// read the next character from the file.

file\_char **=** $fgetc**(**fh**);**

**end**

**end**

**#**100**;**

// Close up the file

$fclose**(**fh**);**

$monitor**(**"Done parsing the SREC file!"**);**

**#**100**;**

$monitor**(**"Initializing the register file with values the same as index"**);**

**#**100**;**

// Zero out the contents of the registers

**for** **(**i**=**0**;** i **<**32**;** i**=**i**+**1**)** **begin**

**if** **(**i **==** 29**)** **begin**

// Initialize the stack pointer to be a large number of bytes from the memory start

processor\_uut**.**reg\_file**.**register**[**i**]** **=** 32'h80020300**;**

**end** **else** **begin**

processor\_uut**.**reg\_file**.**register**[**i**]** **=** 0**;**

**end**

**#**100**;**

**end**

$monitor**(**"Done initializing the register file"**);**

**#**100**;**

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

// Memory is ready to be used after this point!

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

$monitor**(**"Beginning the fetch-decode-execute loop!"**);**

**#**100**;**

// Set the stall in to be 0 just read out the pc, rw, and access size.

srec\_parse **=** 0**;**

**#**200**;**

processor\_uut**.**stall **=** 1**;**

processor\_uut**.**fetch**.**pc **=** 32'h80020000**;**

processor\_uut**.**cur\_pipe\_state **=** 3'b100**;**

processor\_uut**.**next\_pipe\_state **=** 3'b000**;**

**#**100**;**

//$monitor("%h: %h ", processor\_uut.pc\_out, processor\_uut.decode\_ir);

**while** **(**processor\_uut**.**pc **!=** 0**)** **begin** //processor\_uut.pc <= highest\_address) begin

**@(posedge** clk**);**

**if** **(**processor\_uut**.**cur\_pipe\_state **==** 3'b001**)** **begin**

**case(**processor\_uut**.**opcode**)**

6'd0**:** **begin** //JR, ADD, ADDU, SUB SUBU, DIV, SLT, SLTU, SLL, SRL, SRA, AND, OR, XOR, NOR, NOP

**case(**processor\_uut**.**func**)**

6'd0**:** **begin** //SLL, NOP

**if** **(**processor\_uut**.**sha **==** 5'b0**)** $strobe**(**"%h: %h NOP"**,** processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**);**

**else** $strobe**(**"%h: %h sll %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rt**,** processor\_uut**.**sha**);**

**end**

6'd2**:** **begin** //SRL

$strobe**(**"%h: %h srl %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rt**,** processor\_uut**.**sha**);**

**end**

6'd3**:** **begin** //SRA

$strobe**(**"%h: %h sra %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rt**,** processor\_uut**.**sha**);**

**end**

6'd8**:** **begin** //JR

$strobe**(**"%h: %h jr %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**);**

**end**

6'd26**:** **begin** //DIV

$strobe**(**"%h: %h div %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd32**:** **begin** //ADD

$strobe**(**"%h: %h add %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd33**:** **begin** //ADDU

$strobe**(**"%h: %h addu %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd34**:** **begin** //SUB

$strobe**(**"%h: %h sub %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd35**:** **begin** //SUBU

$strobe**(**"%h: %h subu %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd36**:** **begin** //AND

$strobe**(**"%h: %h and %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd37**:** **begin** //OR

$strobe**(**"%h: %h or %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd38**:** **begin** //XOR

$strobe**(**"%h: %h xor %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd39**:** **begin** //NOR

$strobe**(**"%h: %h nor %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd42**:** **begin** //SLT

$strobe**(**"%h: %h slt %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd43**:** **begin** //SLTU

$strobe**(**"%h: %h sltu %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

**default:** **begin**

$strobe**(**"ERROR! %h: %h"**,** processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**);**

$stop**;**

**end**

**endcase**

**end**

6'd1**:** **begin** //BLTZ, BGEZ

**case(**processor\_uut**.**rt**)**

5'd0**:** **begin** //BLTZ

$strobe**(**"%h: %h bltz %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**,** processor\_uut**.**immed**);**

**end**

5'd1**:** **begin** //BGEZ

$strobe**(**"%h: %h bgez %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**,** processor\_uut**.**immed**);**

**end**

**default:** **begin**

$strobe**(**"ERROR! %h: %h"**,** processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**);**

$stop**;**

**end**

**endcase**

**end**

6'd2**:** **begin** //J

//display the destination address of the jump, not the offset.

$strobe**(**"%h: %h j %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** **{**processor\_uut**.**pc\_out**[**31**-:**4**],** 28'b0**}+{**processor\_uut**.**target**,**2'b0**});**

**end**

6'd3**:** **begin** //JAL

//display the destination address of the jump, not the offset.

$strobe**(**"%h: %h jal %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** **{**processor\_uut**.**pc\_out**[**31**-:**4**],** 28'b0**}+{**processor\_uut**.**target**,**2'b0**});**

**end**

6'd4**:** **begin** //BEQ

$strobe**(**"%h: %h beq %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**,** processor\_uut**.**rt**,** processor\_uut**.**immed**);**

**end**

6'd5**:** **begin** //BNE

$strobe**(**"%h: %h bne %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**,** processor\_uut**.**rt**,** processor\_uut**.**immed**);**

**end**

6'd6**:** **begin** //BLEZ

$strobe**(**"%h: %h beq %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**,** processor\_uut**.**immed**);**

**end**

6'd7**:** **begin** //BGTZ

$strobe**(**"%h: %h bgtz %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rs**,** processor\_uut**.**immed**);**

**end**

6'd9**:** **begin** //ADDIU

$strobe**(**"%h: %h addiu %h, %h, %d"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**rs**,** processor\_uut**.**immed**);**

**end**

6'd10**:** **begin** //SLTI

$strobe**(**"%h: %h slti %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**rs**,** processor\_uut**.**immed**);**

**end**

6'd13**:** **begin** //ORI

$strobe**(**"%h: %h ori %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**rs**,** processor\_uut**.**immed**);**

**end**

6'd15**:** **begin** //LUI

$strobe**(**"%h: %h lui %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**immed**);**

**end**

6'd28**:** **begin** //MUL

$strobe**(**"%h: %h mul %h, %h, %h"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rd**,** processor\_uut**.**rs**,** processor\_uut**.**rt**);**

**end**

6'd32**:** **begin** //LB

$strobe**(**"%h: %h lb %h, %d(%h)"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**immed**,** processor\_uut**.**rs**);**

**end**

6'd35**:** **begin** //LW

$strobe**(**"%h: %h lw %h, %d(%h)"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**immed**,** processor\_uut**.**rs**);**

**end**

6'd36**:** **begin** //LBU

$strobe**(**"%h: %h lbu %h, %d(%h)"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**immed**,** processor\_uut**.**rs**);**

**end**

6'd40**:** **begin** //SB

$strobe**(**"%h: %h sb %h, %d(%h)"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**immed**,** processor\_uut**.**rs**);**

**end**

6'd43**:** **begin** //SW

$strobe**(**"%h: %h sw %h, %d(%h)"**,**

processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**,** processor\_uut**.**rt**,** processor\_uut**.**immed**,** processor\_uut**.**rs**);**

**end**

**default:** **begin**

$strobe**(**"ERROR! %h: %h"**,** processor\_uut**.**pc\_out**,** processor\_uut**.**decode\_ir**);**

$stop**;**

**end**

**endcase**

**end**

**end**

//write logic to grab instruction and its operands

//$monitor("Instruction = %h", processor\_uut.decoder.insn\_in);

//#100;

//$stop;

processor\_uut**.**stall **=** 1**;**

processor\_uut**.**fetch**.**pc **=** 32'h80020000**;**

$strobe**(**"Program done running!"**);**

**end**

**always** **begin**

**#**50 clk **=** **!**clk**;**

**end**

// A function to convert ASCII upper case letters and digits to their hexadecimal value.

**function** **[**7**:**0**]**atoh**;**

**input** **[**7**:**0**]**aCode**;**

**begin**

**if** **(**aCode **>=** 8'h30 **&&** aCode **<=** 8'h39**)** **begin**

atoh **=** aCode **-** 8'h30**;**

**end** **else** **if** **(**aCode **>=** 8'h41 **&&** aCode **<=** 8'h5A**)** **begin**

atoh **=** aCode **-** 8'h37**;**

**end**

**end**

**endfunction**

**endmodule**

# Verification Demonstration

For the verification we choose to execute the bubble sort benchmark as it shows all the functionality that the processor has in a single benchmark (i.e. ALU operations, memory operations, branches, and procedure calls). Figure 1 below shows the initial contents of our register before our program begins the fetch, decode, execute loop.

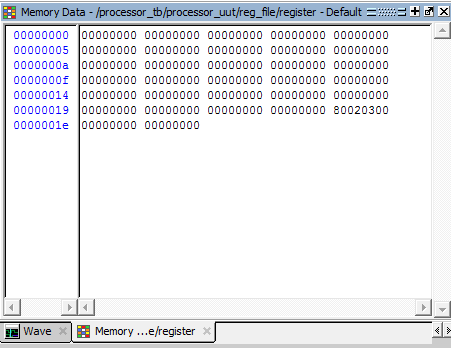


Figure 1. Screen grab of register file contents before BubbleSort.c execution. Top left most entry indicates R0 register value. Bottom right most entry indicates R31 register value.

Figure 2 then shows the register contents after BubbleSort.c has completed execution on the processor.

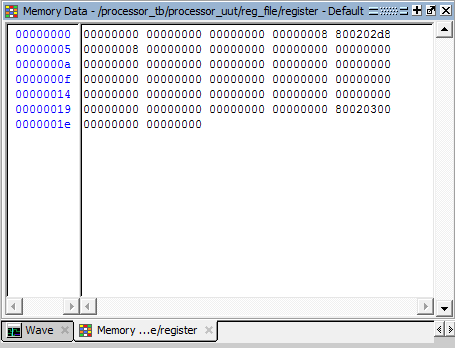


Figure 2. Screen grab of register contents after BubbleSort.c execution.

Since the sorted array is stored in memory we show the memory contents below in Figure 3.

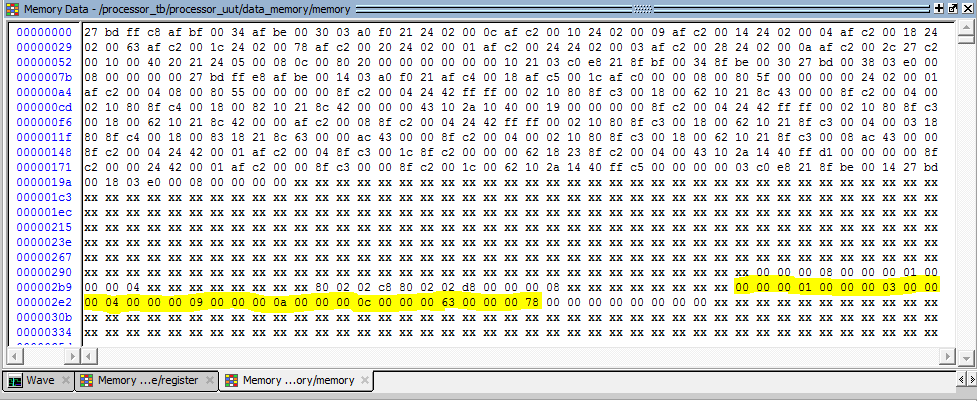


Figure 3. Screen grab of data memory contents after BubbleSort.c execution. The highlighted values are the sorted entries in the array stored in memory.

In the bubble sort benchmark the array has the initial contents of {12, 9, 4, 99, 120, 1, 3, 10}. And our sorted array when you read from the memory contents is {1, 3, 4, 9, 10, 12, 99, 120} which is what was expected.