

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

**Main Memory and SREC Parser**

Group 2

**Jordan *Ross,* Mustafa Faraj**   
j25ross, mabdulaz 26 September, 2014

# Memory Module Code

**module** memory**(**

**output** **reg** **[**31**:**0**]** data\_out**,**

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

**input** **[**31**:**0**]** data\_in**,**

**input** write**,**

**input** clk**,**

**input** **[**1**:**0**]** access\_size

**);**

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

**parameter** size **=** 'h100000**;**

**parameter** offset **=** 'h80020000**;**

**reg** **[**7**:**0**]** memory **[**0**:**size**];** //1048577 = 1MB

**always** **@(posedge** clk**)** **begin** // on the positive edge we will clear the value of the data\_out line

data\_out **=** 32'h00000000**;**

**end**

// On every negative edge of the clock cycle we will write/read values into memory based the address that was set on the rising edge.

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

**if** **(**write**)** **begin** // write the value from data\_in into memory

**if** **(**access\_size **==** 2'b10**)** **begin** //32 bit access

memory**[**address**-**offset**]** **=** data\_in**[**31**:**24**];**

memory**[(**address**-**offset**)** **+** 1**]** **=** data\_in**[**23**:**16**];**

memory**[(**address**-**offset**)** **+** 2**]** **=** data\_in**[**15**:**8**];**

memory**[(**address**-**offset**)** **+** 3**]** **=** data\_in**[**7**:**0**];**

**end**

**if** **(**access\_size **==** 2'b01**)** **begin** //16 bit access

memory**[(**address**-**offset**)]** **=** data\_in**[**15**:**8**];**

memory**[(**address**-**offset**)** **+** 1**]** **=** data\_in**[**7**:**0**];**

**end**

**if** **(**access\_size **==** 2'b00**)** **begin** //8 bit access

memory**[(**address**-**offset**)]** **=** data\_in**[**7**:**0**];**

**end**

**end**

**else** **begin** // Read the value into data\_out

**if** **(**access\_size **==** 'b10**)** **begin** //32 bit access

data\_out**[**7**:**0**]** **=** memory**[(**address**-**offset**)** **+** 3**];**

data\_out**[**15**:**8**]** **=** memory**[(**address**-**offset**)** **+** 2**];**

data\_out**[**23**:**16**]** **=** memory**[(**address**-**offset**)** **+** 1**];**

data\_out**[**31**:**24**]** **=** memory**[(**address**-**offset**)];**

**end**

**if** **(**access\_size **==** 'b01**)** **begin** //16 bit access

data\_out**[**7**:**0**]** **=** memory**[(**address**-**offset**)** **+** 1**];**

data\_out**[**15**:**8**]** **=** memory**[(**address**-**offset**)];**

**end**

**if** **(**access\_size **==** 'b00**)** **begin** //8 bit access

data\_out**[**7**:**0**]** **=** memory**[(**address**-**offset**)];**

**end**

**end**

**end**

**endmodule**

# Memory Module Testbench

**module** memory\_tb**;**

// Output ports for testing

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

// Input ports for testing

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

**reg** **[**31**:**0**]** data\_in**;**

**reg** write**;**

**reg** clk**;**

**reg** **[**1**:**0**]** access\_size**;**

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

// Instantiate the memory module

memory memory\_uut**(.**data\_out**(**data\_out**),**

**.**address**(**address**),**

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

**.**write**(**write**),**

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

**.**access\_size**(**access\_size**)**

**);**

// Parameters to define the address spaces

**parameter** instruction\_offset **=** 32'h80020000**;**

**initial** **begin**

// Start the clock high

clk **=** 1**;**

**#**100**;** // delay 1 clock cycle

// Start giving inputs here

// We want to test writing an value to address 0x80020000 for a program instruction

$monitor**(**"Testing writing a 32 bit value to the first memory location in the instruction address space...\n"**);**

$monitor**(**"Writing the value 0x98765432 to first address in instruction address space...\n"**);**

// Set the address line on the rising edge;

address **=** instruction\_offset**+**0**;**

write **=** 1**;**

**#**50**;** // delay until falling edge

// Set the data line on the falling edge;

data\_in **=** 32'h98765432**;**

access\_size **=** 2'b10**;**

**#**50**;**

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

$monitor**(**"Testing reading the 32 bit value from the first memory location in the instruction address space...\n"**);**

address **=** instruction\_offset**+**0**;**

write **=** 0**;**

access\_size **=** 2'b10**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

$monitor**(**"The 32 bit value in the first instruction address space is 0x%h\n"**,** data\_out**);**

**#**50**;**

$monitor**(**"Testing reading the 16 bit value from the first memory location in the instruction address space...\n"**);**

address **=** instruction\_offset**+**0**;**

write **=** 0**;**

access\_size **=** 2'b01**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

$monitor**(**"The 16 bit value in the first instruction address space is 0x%h\n"**,** data\_out**[**15**:**0**]);**

**#**50**;**

$monitor**(**"Testing reading the 8 bit value from the first memory location in the instruction address space...\n"**);**

address **=** instruction\_offset**+**0**;**

write **=** 0**;**

access\_size **=** 2'b00**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

$monitor**(**"The 8 bit value in the first instruction address space is 0x%h\n"**,** data\_out**[**7**:**0**]);**

**#**50**;**

// Now we are going to test writing a 16-bit value and then reading it back.

// We want to test writing an value to address 0x80020008 for a program instruction

$monitor**(**"Testing writing a 16 bit value into memory...\n"**);**

$monitor**(**"Writing the value 0xAAAA to address 0x80020008...\n"**);**

// Set the address line on the rising edge;

address **=** instruction\_offset**+**8**;**

write **=** 1**;**

**#**50**;** // delay until falling edge

// Set the data line on the falling edge;

data\_in **=** 16'hAAAA**;**

access\_size **=** 2'b01**;**

**#**50**;**

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

$monitor**(**"Testing reading the 16 bit value from 0x80020008...\n"**);**

address **=** instruction\_offset**+**8**;**

write **=** 0**;**

access\_size **=** 2'b01**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

$monitor**(**"The 16 bit value is 0x%h\n"**,** data\_out**);**

**#**50**;**

// Now we are going to test writing a 8-bit value and then reading it back.

// We want to test writing an value to address 0x8002000D for a program instruction

$monitor**(**"Testing writing a 8 bit value into memory...\n"**);**

$monitor**(**"Writing the value 0xBB to address 0x8002000D...\n"**);**

// Set the address line on the rising edge;

address **=** instruction\_offset**+**12**;**

write **=** 1**;**

**#**50**;** // delay until falling edge

// Set the data line on the falling edge;

data\_in **=** 8'hBB**;**

access\_size **=** 2'b00**;**

**#**50**;**

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

$monitor**(**"Testing reading the 8 bit value from 0x8002000D...\n"**);**

address **=** instruction\_offset**+**12**;**

write **=** 0**;**

access\_size **=** 2'b00**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

$monitor**(**"The 8 bit value is 0x%h\n"**,** data\_out**);**

**#**50**;**

**end**

**always** **begin**

**#**50 clk **=** **!**clk**;**

**end**

**endmodule**

# SREC Parser

**module** srec\_parser**;**

// Output ports for memory module

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

// Input ports for memory module

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

**reg** **[**31**:**0**]** data\_in**;**

**reg** write**;**

**reg** clk**;**

**reg** **[**1**:**0**]** access\_size**;**

**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**;** // the address given by the record.

**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**;**

// Instantiate the memory module

memory memory**(.**data\_out**(**data\_out**),**

**.**address**(**address**),**

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

**.**write**(**write**),**

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

**.**access\_size**(**access\_size**)**

**);**

// Parameters to define the address spaces

**parameter** instruction\_offset **=** 32'h80020000**;**

**initial** **begin**

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

// Open the SREC file to read

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

// Start the clock high

clk **=** 1**;**

// 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.

**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**]** **=** 8'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.

address **=** rec\_address**+**data\_offset**;**

data\_in **=** rec\_data**;**

access\_size **=** 2'b01**;**

write **=** 1**;**

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

write **=** 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 32 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.

address **=** rec\_address**+**data\_offset**;**

data\_in **=** rec\_data**;**

access\_size **=** 2'b00**;**

write **=** 1**;**

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

write **=** 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.

address **=** rec\_address**+**data\_offset**;**

data\_in **=** rec\_data**;**

access\_size **=** 2'b00**;**

write **=** 1**;**

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

write **=** 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**

// Close up the file

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

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

**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

//TODO//