```
1   //*************************************************************//
2   //  This document contains information proprietary to the      //
3   //  CSULB student that created the file - any reuse without    //
4   //  adequate approval and documentation is prohibited          //
5   //                                                             //
6   //  Class: CECS 460                                            //
7   //  Project name: TRAMBLAZE PROCESSOR                          //
8   //  File name: tramelblaze.v                                   //
9   //  Release: 1.0 Release Date 17Feb2016                        //
10  //  Release: 1.1 Release Date 25Feb2016                        //
11  //  Release: 1.4 Release Date 04Mar2016                        //
12  //  Release: 1.5 Release Date 17Mar2016                        //
13  //  Release: 1.6 Release Date 04May2016                        //
14  //  Release: 2.0 Release Date 29Aug2016                        //
15  //  Release: 3.0 Release Date 02mar2017                        //
16  //  Release: 3.1 Release Date 30mar2017                        //
17  //  Release: 4.0 Release Date 23aug2017                        //
18  //                                                             //
19  //  Created by John Tramel on 25January2016.                   //
20  //  Copyright  2016 John Tramel. All rights reserved.          //
21  //  Copyright  2017 John Tramel. All rights reserved.          //
22  //                                                             //
23  //  Abstract: Top level for TRAMBLAZE processor                //
24  //  Edit history: 2016JAN25 - created                          //
25  //                2016FEB25 - corrected SHIFT/ROTATE           //
26  //                made sure that CARRY set correctly           //
27  //                2016FEB29 - added MEMHIOL                    //
28  //                MEMHIOL=1 memory access, =0 I/O access       //
29  //                added 512 x 16 scratchpad ram                //
30  //                added 1st FETCH/STORE States                 //
31  //                2016MAR03 - 512x16 scratch ram debugged      //
32  //                2016MAR03 - 512x16 scratch ram debugged      //
33  //                2016MAR17 - 128x16 stack ram debugged        //
34  //                04May2016 - Stack RAM address fix            //
35  //                28Feb2017 - Removed MEMHIOL                  //
36  //                30Mar2017 - Fixed NOP -thanks Chou Thao      //
37  //                                                             //
38  //  In submitting this file for class work at CSULB            //
39  //  I am confirming that this is my work and the work          //
40  //  of no one else.                                            //
41  //                                                             //
42  //  In the event other code sources are utilized I will        //
43  //  document which portion of code and who is the author       //
44  //                                                             //
45  // In submitting this code I acknowledge that plagiarism       //
46  // in student project work is subject to dismissal from the class //
47  //*************************************************************//
48
49  `timescale 1ns/1ns
50
51  module tramelblaze (CLK, RESET, IN_PORT, INTERRUPT,
52                      OUT_PORT, PORT_ID, READ_STROBE, WRITE_STROBE, INTERRUPT_ACK,
53                      ADDRESS, INSTRUCTION);
54
55  input         CLK;
56  input         RESET;
57  input  [15:0] IN_PORT;
```

```
 58   input         INTERRUPT;
 59
 60   output [15:0] OUT_PORT;
 61   output [15:0] PORT_ID;
 62   output        READ_STROBE;
 63   output        WRITE_STROBE;
 64   output        INTERRUPT_ACK;
 65
 66   output [11:0] ADDRESS;
 67   input  [15:0] INSTRUCTION;
 68
 69   reg    [15:0] inst_reg;                    // instruction register
 70   reg    [15:0] const_reg;                   // constant register
 71   reg    [15:0] pc;                          // program counter
 72   reg    [15:0] regfile [0:15];              // 16 x 16 register file
 73   reg    [16:0] alu_out;                     // output of ALU
 74   reg    [16:0] alu_out_reg;                 // output of ALU registered
 75   reg    [ 3:0] stateX,stateQ;               // state machine variable
 76   wire   [11:0] ADDRESS;                     // ADDRESS to instruction memory
 77   reg    [15:0] address_mux;                 // mux to select next address
 78   reg           int_enable;                  // enable interrupt
 79   reg           int_proc;                    // processor interrupt
 80   wire          carryPX;                     // preserve carry bit
 81   reg           carryPQ;                     // preserve carry bit
 82   wire          zeroPX;                      // preserve zero bit
 83   reg           zeroPQ;                      // preserve zero bit
 84   reg           zeroX,zeroQ;                 // flag
 85   reg           carryX,carryQ;               // flag
 86   reg           loadKX,loadKQ;               // load constant register
 87   reg           ldirX,ldirQ;                 // load instruction register
 88   wire          ldk;                         // load constant register
 89   reg           ldpcX,ldpcQ;                 // load program counter
 90   reg           ldflagX,ldflagQ;             // load carry and zero registers
 91   reg           ldflagPX,ldflagPQ;           // preserve load carry and zero registers
 92   reg           wtrfX,wtrfQ;                 // write register file
 93   reg           wtsrX,wtsrQ;                 // write scratchpad ram
 94   reg           sel_alubX,sel_alubQ;         // select alu operand b
 95   reg           pushX,pushQ;                 // push pc address onto stack
 96   reg           popX,popQ;                   // pop pc address from stack
 97   reg           enintX,enintQ;               // enable interrupts
 98   reg           disintX,disintQ;             // disable interrupts
 99   reg    [1:0]  sel_pcX,sel_pcQ;             // select pc source
100   reg           sel_portidX,sel_portidQ;     // select source for port id
101   reg    [2:0]  sel_rfwX,sel_rfwQ;           // select reg write data source
102   reg    [2:0]  flag_selX,flag_selQ;         // select source to change zero/carry
103   reg    [4:0]  alu_opX,alu_opQ;             // select which operation alu does
104   reg           enableportidX,enableportidQ; // allow port id to switch
105   reg           enableinportX,enableinportQ; // allow in port to be read
106   reg           enableoutportX,enableoutportQ; // allow out port to switch
107   reg           readstrobeX,readstrobeQ;     // set read strobe output
108   reg           writestrobeX,writestrobeQ;   // set write strobe output
109   reg           interruptackX,interruptackQ; // interrupt acknowledge
110   wire   [15:0] pc_min1;                     // program counter minus one
111   wire   [15:0] stackWdata;                  // data written into stack
112
113   wire    [6:0] opcode;                      // current opcode being executed
114   wire    [3:0] regX_adrs, regY_adrs;        // address to x and y registers
```

```
115   reg    [15:0] rf_wdata;                    // data to write into register file
116   wire   [15:0] stackRdata;                 // contents of stack pointed to (last write)
117   wire   [15:0] alu_a;                       // input to ALU A
118   wire   [15:0] alu_b;                       // input to ALUB
119   wire          INTERRUPT_ACK;
120   wire          READ_STROBE;
121   wire          WRITE_STROBE;
122   wire   [15:0] regA;                        // output of register file - A
123   wire   [15:0] regB;                        // output of register file - B
124   wire   [15:0] scratch_dout;                      // scratch pad ram output
125   wire    [8:0] scratch_adrs;
126   wire   [15:0] scratch_din;
127   wire   [ 6:0] stackAdrs;
128   reg    [ 6:0] stackPointQ;
129   wire   [ 6:0] stackPointD;
130
131   // assume instruction memory is 8K deep (000 - FFF)
132
133   parameter INTERRUPT_ADDRESS = 16'H0FFE;
134
135   // parameters for STATES
136
137   parameter FETCH = 5'H00, DECODE  = 5'H01, SECOND = 5'H02, THIRD   = 5'H03, EXECUTE =
      5'H04,
138             ENDIT = 5'H05, ENDCALL = 5'H06, ENDRET = 5'H07, ENDRET2 = 5'H08, ENDRET3 =
      5'H09,
139             OUTPUT_XK_2 = 5'H0A, OUTPUT_XY_2 = 5'H0B, INPUT_XP_2 = 5'H0C, INPUT_XY_2 =
      5'H0D,
140             FETCH_XK_2 = 5'H0E, FETCH_XY_2 = 5'H0F, STORE_XK_2 = 5'H10, STORE_XY_2 =
      5'H11;
141
142   // parameters for ALU OPERATIONS
143
144   parameter NOTHING = 5'H00, ADD  = 5'H01, ADDC = 5'H02, AND  = 5'H03,
145             SUB     = 5'H04, OR   = 5'H05, RLX  = 5'H06, RRX  = 5'H07,
146             SL0X    = 5'H08, SL1X = 5'H09, SLAX = 5'H0A, SLXX = 5'H0B,
147             SR0X    = 5'H0C, SR1X = 5'H0D, SRAX = 5'H0E, SRXX = 5'H0F,
148             XOR     = 5'H10, SUBC = 5'H11;
149
150   // parameters for OPCODES
151   //
152   parameter NOP         = 7'H00, ADD_XK      = 7'H02, ADD_XY      = 7'H04,
153             ADDCY_XK    = 7'H06, ADDCY_XY    = 7'H08, AND_XK      = 7'H0A,
154             AND_XY      = 7'H0C, CALL_AAA    = 7'H0E, CALLC_AAA   = 7'H10,
155             CALLNC_AAA  = 7'H12, CALLZ_AAA   = 7'H14, CALLNZ_AAA  = 7'H16,
156             COMP_XK     = 7'H18, COMP_XY     = 7'H1A, DISINT      = 7'H1C,
157             ENINT       = 7'H1E, INPUT_XY    = 7'H20, INPUT_XP    = 7'H22,
158             JUMP_AAA    = 7'H24, JUMPC_AAA   = 7'H26, JUMPNC_AAA  = 7'H28,
159             JUMPZ_AAA   = 7'H2A, JUMPNZ_AAA  = 7'H2C, LOAD_XK     = 7'H2E,
160             LOAD_XY     = 7'H30, OR_XK       = 7'H32, OR_XY       = 7'H34,
161             OUTPUT_XY   = 7'H36, OUTPUT_XK   = 7'H38, RETURN      = 7'H3A,
162             RETURN_C    = 7'H3C, RETURN_NC   = 7'H3E, RETURN_Z    = 7'H40,
163             RETURN_NZ   = 7'H42, RETURN_DIS  = 7'H44, RETURN_EN   = 7'H46,
164             RL_X        = 7'H48, RR_X        = 7'H4A, SL0_X       = 7'H4C,
165             SL1_X       = 7'H4E, SLA_X       = 7'H50, SLX_X       = 7'H52,
166             SR0_X       = 7'H54, SR1_X       = 7'H56, SRA_X       = 7'H58,
167             SRX_X       = 7'H5A, SUB_XK      = 7'H5C, SUB_XY      = 7'H5E,
```

```
168             SUBC_XK     = 7'H60, SUBC_XY     = 7'H62, TEST_XK     = 7'H64,
169             TEST_XY     = 7'H66, XOR_XK      = 7'H68, XOR_XY      = 7'H6A,
170             FETCH_XK    = 7'H70, FETCH_XY    = 7'H72, STORE_XK    = 7'H74,
171             STORE_XY    = 7'H76;
172
173   assign READ_STROBE = readstrobeQ;
174   assign WRITE_STROBE = writestrobeQ;
175   assign INTERRUPT_ACK = interruptackQ;
176
177   ////////////////////////////
178   // address register - PC    //
179   ////////////////////////////
180
181   assign ADDRESS = pc[11:0];
182
183   always @(posedge CLK, posedge RESET)
184      if (RESET)
185         pc <= 16'b0;
186      else
187         if (ldpcQ)
188            pc <= address_mux;
189
190   always @(*)
191      case(sel_pcQ)
192         2'b00: address_mux = pc + 16'b1;
193         2'b01: address_mux = stackRdata;
194         2'b10: address_mux = const_reg;
195         2'b11: address_mux = INTERRUPT_ADDRESS;
196      endcase
197
198   ////////////////////////////
199   // address stack            //
200   ////////////////////////////
201
202   assign ldsp = popQ | pushQ;
203
204   always @(posedge CLK, posedge RESET)
205           if (RESET) stackPointQ <= 7'b0; else
206           if (ldsp)  stackPointQ <= stackPointD;
207
208   assign pc_min1 = pc - 16'b1;
209   assign stackWdata = int_proc ? pc_min1 : pc;
210
211
212   assign stackAdrs   = pushQ ? stackPointQ      : stackPointQ - 7'b1;
213   assign stackPointD = pushQ ? stackAdrs + 7'b1 : stackAdrs;
214
215   stack_ram stkr (
216           .addra(stackAdrs),
217           .dina(stackWdata),
218           .wea(pushQ),
219           .clka(CLK),
220           .douta(stackRdata)
221           );
222
223   ////////////////////////////
224   // instruction register     //
```

```
225   ////////////////////////////////
226
227   assign opcode    = inst_reg[14:8];       // opcode for instruction decode
228   assign regY_adrs = inst_reg[7:4];
229   assign regX_adrs = inst_reg[3:0];
230   assign ldk  = inst_reg[15] && (stateQ==DECODE);
231
232   always @(posedge CLK, posedge RESET)
233      if (RESET) inst_reg <= 16'b0; else
234      if (ldirQ)  inst_reg <= INSTRUCTION;
235
236   ////////////////////////////////
237   // constant register        //
238   ////////////////////////////////
239
240   always @(posedge CLK, posedge RESET)
241      if (RESET)  const_reg <= 16'b0; else
242      if (loadKQ) const_reg <= INSTRUCTION;
243
244   ////////////////////////////////
245   // interrupt control        //
246   ////////////////////////////////
247
248   always @(posedge CLK, posedge RESET)
249      if (RESET)   int_enable <= 1'b0; else
250      if (enintQ)  int_enable <= 1'b1; else
251      if (disintQ) int_enable <= 1'b0;
252
253   always @(posedge CLK, posedge RESET)
254      if (RESET)                    int_proc <= 1'b0; else
255      if (INTERRUPT_ACK)            int_proc <= 1'b0; else
256      if (int_enable & INTERRUPT)  int_proc <= 1'b1;
257
258   ////////////////////////////////
259   // register file operations //
260   ////////////////////////////////
261
262   assign regA = regfile[regX_adrs];
263   assign regB = regfile[regY_adrs];
264
265   always @(*)
266           case (sel_rfwQ)
267                   3'b000: rf_wdata = alu_out[15:0];
268                   3'b001: rf_wdata = IN_PORT & {16{enableinportQ}};
269                   3'b010: rf_wdata = const_reg;
270                   3'b011: rf_wdata = regB;
271                   3'b100: rf_wdata = scratch_dout;
272                  default: rf_wdata = alu_out[15:0];
273                   endcase
274
275   always @(posedge CLK, posedge RESET)
276      if (RESET) begin
277         regfile[0]  <= 16'b0;
278         regfile[1]  <= 16'b0;
279         regfile[2]  <= 16'b0;
280         regfile[3]  <= 16'b0;
281         regfile[4]  <= 16'b0;
```

```
282         regfile[5]  <= 16'b0;
283         regfile[6]  <= 16'b0;
284         regfile[7]  <= 16'b0;
285         regfile[8]  <= 16'b0;
286         regfile[9]  <= 16'b0;
287         regfile[10] <= 16'b0;
288         regfile[11] <= 16'b0;
289         regfile[12] <= 16'b0;
290         regfile[13] <= 16'b0;
291         regfile[14] <= 16'b0;
292         regfile[15] <= 16'b0;
293         end else
294      if (wtrfQ) begin
295         regfile[regX_adrs] <= rf_wdata;
296         end
297
298   ////////////////////////////
299   // CARRY/ZERO operations   //
300   ////////////////////////////
301
302   assign zeroPX  = zeroQ;
303   assign carryPX = carryQ;
304
305   always @(posedge CLK, posedge RESET)
306      if (RESET)    {zeroPQ,carryPQ} <= 2'b0; else
307      if (ldflagPQ) {zeroPQ,carryPQ} <= {zeroPX,carryPX};
308
309   always @(posedge CLK, posedge RESET)
310      if (RESET)    {zeroQ,carryQ} <= 2'b0; else
311      if (ldflagQ)  {zeroQ,carryQ} <= {zeroX,carryX};
312
313   always @(*)
314      case(flag_selQ)
315         3'h0: {zeroX, carryX} = {zeroQ, carryQ};
316         3'h1: {zeroX, carryX} = {~|alu_out[15:0],1'b0};
317         3'h2: {zeroX, carryX} = {~|alu_out[15:0],alu_out[16]};
318         3'h3: {zeroX, carryX} = {zeroPQ, carryPQ};
319         3'h4: {zeroX, carryX} = {~|alu_out[15:0],alu_out[15]};
320         3'h5: {zeroX, carryX} = {~|alu_out[15:0],alu_out[0]};
321         3'h6: {zeroX, carryX} = {~|(alu_a & alu_b),^(alu_a & alu_b)};
322      default: {zeroX, carryX} = {zeroQ, carryQ};
323         endcase
324
325   ////////////////////////////
326   // ALU operations          //
327   ////////////////////////////
328
329   assign alu_a = regA;
330   assign OUT_PORT = regA & {16{enableoutportQ}};
331   assign alu_b = (sel_alubQ) ? const_reg : regB;
332   assign PORT_ID = (sel_portidQ) ? (const_reg & {16{enableportidQ}}) : (alu_b & {16{
      enableportidQ}});
333
334   always @(posedge CLK, posedge RESET)
335      if (RESET) alu_out_reg <= 17'b0;
336      else       alu_out_reg <= alu_out;
337
```

```
338   always @(*)
339      case(alu_opQ)
340         NOTHING: alu_out = alu_out_reg;                        // noop so no change
341         ADD:     alu_out = alu_a + alu_b;                      // ADD
342         ADDC:    alu_out = alu_a + alu_b + carryQ;             // ADDC
343         SUB:     alu_out = alu_a - alu_b;                      // SUB (COMP)
344         SUBC:    alu_out = alu_a - alu_b - carryQ;             // SUBC
345         AND:     alu_out = alu_a & alu_b;                      // AND
346         OR:      alu_out = alu_a | alu_b;                      // OR
347         XOR:     alu_out = alu_a ^ alu_b;                      // XOR
348         RLX:     alu_out = {alu_a[15],alu_a[14:0],alu_a[15]};  // RL rX
349         RRX:     alu_out = {alu_a[ 0],alu_a[0],alu_a[15:1]};   // RR rX
350         SL0X:    alu_out = {alu_a[15],alu_a[14:0],1'b0};       // SL0 rX
351         SL1X:    alu_out = {alu_a[15],alu_a[14:0],1'b1};       // SL1 rX
352         SLAX:    alu_out = {alu_a[15],alu_a[14:0],carryQ};     // SLA rX
353         SLXX:    alu_out = {alu_a[15],alu_a[14:0],alu_a[0]};   // SLX rX
354         SR0X:    alu_out = {alu_a[ 0],1'b0,alu_a[15:1]};       // SR0 rX
355         SR1X:    alu_out = {alu_a[ 0],1'b1,alu_a[15:1]};       // SR1 rX
356         SRAX:    alu_out = {alu_a[ 0],carryQ,alu_a[15:1]};     // SRA rX
357         SRXX:    alu_out = {alu_a[ 0],alu_a[15],alu_a[15:1]};  // SRX rX
358       default: alu_out = 16'b0;
359      endcase
360
361   ////////////////////////////////
362   // Scratchpad RAM Instance    //
363   // 512x16 Scratchpad Memory   //
364   ////////////////////////////////
365
366   assign scratch_din = alu_a;
367   assign scratch_adrs = alu_b[8:0];
368
369   scratch_ram sr (
370      .clka(CLK),
371      .wea(wtsrQ),
372      .addra(scratch_adrs),
373      .dina(scratch_din),
374      .douta(scratch_dout)
375      );
376
377   ////////////////////////////////
378   // Instruction Control Logic //
379   ////////////////////////////////
380
381
382   always @(posedge CLK, posedge RESET)
383      if (RESET) begin
384         stateQ <= FETCH;                 // start up state variable
385         ldirQ <= 1'b1;                   // load instruction register
386         ldpcQ <= 1'b1;                   // load program counter
387         ldflagQ <= 1'b0;                 // load carry and zero registers
388         ldflagPQ <= 1'b0;                // load preserve carry and zero registers
389         loadKQ <= 1'b0;                  // load constant register
390         wtrfQ <= 1'b0;                   // write register file
391         wtsrQ <= 1'b0;                   // write scratch pad ram
392         sel_alubQ <= 1'b0;               // select alu operand b
393         pushQ <= 1'b0;                   // push pc address onto stack
394         popQ <= 1'b0;                    // pop pc address from stack
```

```
395         enintQ <= 1'b0;                    // enable interrupts
396         disintQ <= 1'b0;                   // disable interrupts
397         sel_pcQ <= 2'b0;                   // select pc source
398         sel_portidQ <= 1'b0;               // select source for port id
399         sel_rfwQ <= 3'b0;                  // select reg write data source
400         flag_selQ <= 2'b0;                 // select source to change zero/carry
401         alu_opQ <= 5'b0;                   // select which operation alu does
402         enableportidQ <= 1'b0;             // allow port id to switch
403         enableinportQ <= 1'b0;             // allow in port to be read
404         enableoutportQ <= 1'b0;            // allow out port to switch
405         readstrobeQ <= 1'b0;               // set read strobe output
406         writestrobeQ <= 1'b0;              // set write strobe output
407         interruptackQ <= 1'b0;             // set interrupt ack
408         end
409     else
410         begin
411         stateQ <= stateX;                  // update up state variable
412         ldirQ <= ldirX;                    // load instruction register
413         ldpcQ <= ldpcX;                    // load program counter
414         ldflagQ <= ldflagX;                // load carry and zero registers
415         ldflagPQ <= ldflagPX;              // load preserve carry and zero registers
416         loadKQ <= loadKX;                  // load constant register
417         wtrfQ <= wtrfX;                    // write register file
418         wtsrQ <= wtsrX;                    // write scratch pad ram
419         sel_alubQ <= sel_alubX;            // select alu operand b
420         pushQ <= pushX;                    // push pc address onto stack
421         popQ <= popX;                      // pop pc address from stack
422         enintQ <= enintX;                  // enable interrupts
423         disintQ <= disintX;                // disable interrupts
424         sel_pcQ <= sel_pcX;                // select pc source
425         sel_portidQ <= sel_portidX;        // select source for port id
426         sel_rfwQ <= sel_rfwX;              // select reg write data source
427         flag_selQ <= flag_selX;            // select source to change zero/carry
428         alu_opQ <= alu_opX;                // select which operation alu does
429         enableportidQ <= enableportidX;    // allow port id to switch
430         enableinportQ <= enableinportX;    // allow in port to be read
431         enableoutportQ <= enableoutportX; // allow out port to switch
432         readstrobeQ <= readstrobeX;        // set read strobe output
433         writestrobeQ <= writestrobeX;      // set write strobe output
434         interruptackQ <= interruptackX;    // set interrupt ack
435         end
436
437 ////////////////////////////////////////
438 // State Machine Decision Making Block //
439 ////////////////////////////////////////
440
441 always@(*)
442    begin
443    ldirX = 1'b0;                  // load instruction register
444    ldpcX = 1'b0;                  // load program counter
445    ldflagX = 1'b0;                // load carry and zero registers
446    ldflagPX = 1'b0;               // load preserve carry and zero registers
447    loadKX = 1'b0;                 // load constant register
448    wtrfX = 1'b0;                  // write register file
449    wtsrX = 1'b0;                  // write scratch pad ram
450    sel_alubX = 1'b0;               // select alu operand b
451    pushX = 1'b0;                  // push pc address onto stack
```

```
452       popX = 1'b0;                    // pop pc address from stack
453       enintX = 1'b0;                  // enable interrupts
454       disintX = 1'b0;                 // disable interrupts
455       sel_pcX = 2'b0;                 // select pc source
456       sel_portidX = 1'b0;              // select source for port id
457       sel_rfwX = 3'b0;                // select reg write data source
458       flag_selX = 2'b0;               // select source to change zero/carry
459       alu_opX = 5'b0;                 // select which operation alu does
460       enableportidX = 1'b0;           // allow port id to switch
461       enableinportX = 1'b0;           // allow in port to be read
462       enableoutportX = 1'b0;          // allow out port to switch
463       readstrobeX = 1'b0;             // set read strobe output
464       writestrobeX = 1'b0;            // set write strobe output
465       interruptackX = 1'b0;           // set interrupt ack
466       stateX = FETCH;
467
468       case(stateQ)
469          FETCH: begin
470          if (int_proc) begin
471             sel_pcX=2'b11;               // goto interrupt
472             ldpcX=1'b1;                  // update new pc
473             pushX  =1'b1;                // push next pc onto stack
474             disintX=1'b1;                // entering interrupt clear interrupt
475             ldflagPX=1'b1;               // preserve the flag registers
476             interruptackX=1'b1;          // set interrupt ack
477             stateX=ENDRET2;              // let int_proc reset
478             end
479          else begin
480             ldpcX=1'b0;
481             ldirX=1'b0;
482             stateX=DECODE;
483             end
484          end
485
486          DECODE: begin
487          if(ldk) begin
488             loadKX=1'b1;
489             stateX=SECOND;
490             end
491          else begin
492             stateX=EXECUTE;
493             end
494          end
495
496         SECOND: begin
497         ldpcX=1'b1;
498         stateX=THIRD;
499         end
500
501         THIRD: begin
502          stateX=EXECUTE;
503          end
504
505          EXECUTE: begin
506          case(opcode)
507             NOP:stateX=ENDIT;
508
```

```
509             ADD_XK: begin
510             wtrfX=1'b1;
511             sel_alubX=1'b1;
512             flag_selX=3'b010;
513             ldflagX=1'b1;
514             alu_opX=ADD;
515             stateX=ENDIT;
516             end
517
518             ADD_XY: begin
519             wtrfX=1'b1;
520             flag_selX=3'b010;
521             alu_opX=ADD;
522             ldflagX=1'b1;
523             stateX=ENDIT;
524             end
525
526             ADDCY_XK: begin
527             wtrfX=1'b1;
528             sel_alubX=1'b1;
529             flag_selX=3'b010;
530             ldflagX=1'b1;
531             alu_opX=ADDC;
532             stateX=ENDIT;
533             end
534
535             ADDCY_XY: begin
536             wtrfX=1'b1;
537             flag_selX=3'b010;
538             ldflagX=1'b1;
539             alu_opX=ADDC;
540             stateX=ENDIT;
541             end
542
543             AND_XK: begin
544             wtrfX=1'b1;
545             sel_alubX=1'b1;
546             flag_selX=3'b001;
547             ldflagX=1'b1;
548             alu_opX=AND;
549             stateX=ENDIT;
550             end
551
552             AND_XY: begin
553             wtrfX=1'b1;
554             alu_opX=AND;
555             flag_selX=3'b001;
556             ldflagX=1'b1;
557             stateX=ENDIT;
558             end
559
560             CALL_AAA: begin
561             pushX=1'b1;
562             sel_pcX=2'b10;
563             ldpcX=1'b1;
564             stateX=ENDCALL;
565             end
```

```
566
567             CALLC_AAA: begin
568             if(carryQ) begin
569                pushX=1'b1;
570                sel_pcX=2'b10;
571                ldpcX=1'b1;
572                stateX=ENDCALL;
573                end else
574             stateX=ENDIT;
575             end
576
577             CALLNC_AAA: begin
578             if(!carryQ) begin
579                pushX=1'b1;
580                sel_pcX=2'b10;
581                ldpcX=1'b1;
582                stateX=ENDCALL;
583                end else
584             stateX=ENDIT;
585             end
586
587             CALLZ_AAA: begin
588             if(zeroQ) begin
589                pushX=1'b1;
590                sel_pcX=2'b10;
591                ldpcX=1'b1;
592                stateX=ENDCALL;
593                end else
594             stateX=ENDIT;
595             end
596
597             CALLNZ_AAA: begin
598             if(!zeroQ) begin
599                pushX=1'b1;
600                sel_pcX=2'b10;
601                ldpcX=1'b1;
602                stateX=ENDCALL;
603                end else
604             stateX=ENDIT;
605             end
606
607             COMP_XK: begin
608             alu_opX=SUB;
609             sel_alubX=1'b1;
610             flag_selX=3'b010;
611             ldflagX=1'b1;
612             stateX=ENDIT;
613             end
614
615             COMP_XY: begin
616             alu_opX=SUB;
617             flag_selX=3'b010;
618             ldflagX=1'b1;
619             stateX=ENDIT;
620             end
621
622             DISINT: begin
```

```
623             disintX=1'b1;
624             stateX=ENDCALL;
625             end
626
627             ENINT: begin
628             enintX=1'b1;
629             stateX=ENDCALL;
630             end
631
632             STORE_XY:begin
633             wtsrX=1'b1;
634             stateX=ENDIT;
635             end
636
637             STORE_XK: begin
638             sel_alubX=1'b1;
639             wtsrX=1'b1;
640             stateX=ENDIT;
641             end
642
643             FETCH_XY:begin
644             stateX=FETCH_XY_2;
645             end
646
647             FETCH_XK: begin
648             sel_alubX=1'b1;
649             stateX=FETCH_XK_2;
650             end
651
652             INPUT_XY:begin
653             enableportidX=1'b1;
654             enableinportX=1'b1;
655             stateX=INPUT_XY_2;
656             end
657
658             INPUT_XP: begin
659             enableinportX=1'b1;
660             enableportidX=1'b1;
661             sel_portidX=1'b1;
662             stateX=INPUT_XP_2;
663             end
664
665             JUMP_AAA: begin
666             sel_pcX=2'b10;
667             ldpcX=1'b1;
668             ldirX=1'b1;
669             stateX=ENDCALL;
670             end
671
672             JUMPC_AAA: begin
673             if(carryQ) begin
674                sel_pcX=2'b10;
675                ldpcX=1'b1;
676                ldirX=1'b1;
677                end
678             stateX=ENDCALL;
679             end
```

```
680
681             JUMPNC_AAA: begin
682             if(!carryQ) begin
683                sel_pcX=2'b10;
684                ldpcX=1'b1;
685                ldirX=1'b1;
686                end
687             stateX=ENDCALL;
688             end
689
690             JUMPZ_AAA: begin
691             if(zeroQ) begin
692                sel_pcX=2'b10;
693                ldpcX=1'b1;
694                ldirX=1'b1;
695                end
696             stateX=ENDCALL;
697             end
698
699             JUMPNZ_AAA: begin
700             if(!zeroQ) begin
701                sel_pcX=2'b10;
702                ldpcX=1'b1;
703                ldirX=1'b1;
704                end
705             stateX=ENDCALL;
706             end
707
708             LOAD_XK: begin
709             sel_rfwX=3'b010;
710             wtrfX=1'b1;
711
712             stateX=ENDIT;
713             end
714
715             LOAD_XY: begin
716             sel_rfwX=3'b011;
717             wtrfX=1'b1;
718             stateX=ENDIT;
719             end
720
721             OR_XK: begin
722             wtrfX=1'b1;
723             flag_selX=3'b001;
724             alu_opX=OR;
725             sel_alubX=1'b1;
726             ldflagX=1'b1;
727             stateX=ENDIT;
728             end
729
730             OR_XY: begin
731             wtrfX=1'b1;
732             flag_selX=3'b001;
733             alu_opX=OR;
734             ldflagX=1'b1;
735             stateX=ENDIT;
736             end
```

```
737
738             OUTPUT_XK: begin
739             sel_portidX=1'b1;
740             enableportidX=1'b1;
741             enableoutportX=1'b1;
742             stateX=OUTPUT_XK_2;
743             end
744
745             OUTPUT_XY: begin
746             enableportidX=1'b1;
747             enableoutportX=1'b1;
748             stateX=OUTPUT_XY_2;
749             end
750
751             RETURN: begin
752             popX=1'b1;
753             stateX=ENDRET;
754             end
755
756             RETURN_C: begin
757             if(carryQ) begin
758                popX=1'b1;
759                stateX=ENDRET;
760                end else
761             stateX=ENDIT;
762             end
763
764             RETURN_NC: begin
765             if(!carryQ) begin
766                popX=1'b1;
767                stateX=ENDRET;
768                end else
769             stateX=ENDIT;
770             end
771
772             RETURN_Z: begin
773             if(zeroQ) begin
774                popX=1'b1;
775                stateX=ENDRET;
776                end else
777             stateX=ENDIT;
778             end
779
780             RETURN_NZ: begin
781             if(!zeroQ) begin
782                popX=1'b1;
783                stateX=ENDRET;
784                end else
785             stateX=ENDIT;
786             end
787
788             RETURN_DIS: begin
789             flag_selX=3'b011;
790             ldflagX=1'b1;
791             disintX=1'b1;
792             popX=1'b1;
793             stateX=ENDRET;
```

```
794             end
795
796             RETURN_EN: begin
797             flag_selX=3'b011;
798             ldflagX=1'b1;
799             enintX=1'b1;
800             popX=1'b1;
801             stateX=ENDRET;
802             end
803
804             RL_X: begin
805             wtrfX=1'b1;
806             alu_opX=RLX;
807             flag_selX=3'b010;
808             ldflagX=1'b1;
809             stateX=ENDIT;
810             end
811
812             RR_X: begin
813             wtrfX=1'b1;
814             alu_opX=RRX;
815             flag_selX=3'b010;
816             ldflagX=1'b1;
817             stateX=ENDIT;
818             end
819
820             SL0_X: begin
821             wtrfX=1'b1;
822             alu_opX=SL0X;
823             flag_selX=3'b010;
824             ldflagX=1'b1;
825             stateX=ENDIT;
826             end
827
828             SL1_X: begin
829             wtrfX=1'b1;
830             alu_opX=SL1X;
831             flag_selX=3'b010;
832             ldflagX=1'b1;
833             stateX=ENDIT;
834             end
835
836             SLA_X: begin
837             wtrfX=1'b1;
838             alu_opX=SLAX;
839             flag_selX=3'b010;
840             ldflagX=1'b1;
841             stateX=ENDIT;
842             end
843
844             SLX_X: begin
845             wtrfX=1'b1;
846             alu_opX=SLXX;
847             flag_selX=3'b010;
848             ldflagX=1'b1;
849             stateX=ENDIT;
850             end
```

```
851
852             SR0_X: begin
853             wtrfX=1'b1;
854             alu_opX=SR0X;
855             flag_selX=3'b010;
856             ldflagX=1'b1;
857             stateX=ENDIT;
858             end
859
860             SR1_X: begin
861             wtrfX=1'b1;
862             alu_opX=SR1X;
863             flag_selX=3'b010;
864             ldflagX=1'b1;
865             stateX=ENDIT;
866             end
867
868             SRA_X: begin
869             wtrfX=1'b1;
870             alu_opX=SRAX;
871             flag_selX=3'b010;
872             ldflagX=1'b1;
873             stateX=ENDIT;
874             end
875
876             SRX_X: begin
877             wtrfX=1'b1;
878             alu_opX=SRXX;
879             flag_selX=3'b010;
880             ldflagX=1'b1;
881             stateX=ENDIT;
882             end
883
884             SUB_XK: begin
885             wtrfX=1'b1;
886             sel_alubX=1'b1;
887             alu_opX=SUB;
888             flag_selX=3'b010;
889             ldflagX=1'b1;
890             stateX=ENDIT;
891             end
892
893             SUB_XY: begin
894             wtrfX=1'b1;
895             flag_selX=3'b010;
896             alu_opX=SUB;
897             ldflagX=1'b1;
898             stateX=ENDIT;
899             end
900
901             SUBC_XK: begin
902             wtrfX=1'b1;
903             sel_alubX=1'b1;
904             alu_opX=SUBC;
905             flag_selX=3'b010;
906             ldflagX=1'b1;
907             stateX=ENDIT;
```

```
908             end
909
910             SUBC_XY: begin
911             wtrfX=1'b1;
912             alu_opX=SUBC;
913             flag_selX=3'b010;
914             ldflagX=1'b1;
915             stateX=ENDIT;
916             end
917
918             TEST_XK: begin
919             sel_alubX=1'b1;
920             alu_opX=AND;
921             flag_selX=3'b110;
922             ldflagX=1'b1;
923             stateX=ENDIT;
924             end
925
926             TEST_XY: begin
927             alu_opX=AND;
928             flag_selX=3'b110;
929             ldflagX=1'b1;
930             stateX=ENDIT;
931             end
932
933             XOR_XK: begin
934             wtrfX=1'b1;
935             sel_alubX=1'b1;
936             alu_opX=XOR;
937             flag_selX=3'b001;
938             ldflagX=1'b1;
939             stateX=ENDIT;
940             end
941
942             XOR_XY:begin
943             wtrfX=1'b1;
944             alu_opX=XOR;
945             flag_selX=3'b001;
946             ldflagX=1'b1;
947             stateX=ENDIT;
948             end
949
950             default:stateX=FETCH;
951
952         endcase
953         end
954
955         INPUT_XP_2: begin
956         enableportidX=1'b1;
957         enableinportX=1'b1;
958         readstrobeX=1'b1;
959         sel_portidX=1'b1;
960         sel_rfwX=3'b001;
961         wtrfX=1'b1;
962         stateX=ENDIT;
963         end
964
```

```
965         INPUT_XY_2:begin
966         enableportidX=1'b1;
967         enableinportX=1'b1;
968         readstrobeX=1'b1;
969         sel_rfwX=3'b001;
970         wtrfX=1'b1;
971         stateX=ENDIT;
972         end
973
974         OUTPUT_XK_2: begin
975         sel_portidX=1'b1;
976         enableoutportX=1'b1;
977         enableportidX=1'b1;
978         writestrobeX=1'b1;
979         stateX=ENDIT;
980         end
981
982         OUTPUT_XY_2: begin
983         enableoutportX=1'b1;
984         enableportidX=1'b1;
985         writestrobeX=1'b1;
986         stateX=ENDIT;
987         end
988
989         FETCH_XK_2:begin
990         sel_rfwX=3'b100;
991         wtrfX=1'b1;
992         stateX=ENDIT;
993         end
994
995         FETCH_XY_2:begin
996         sel_rfwX=3'b100;
997         wtrfX=1'b1;
998         stateX=ENDIT;
999         end
1000
1001        ENDCALL: begin
1002        stateX=ENDIT;
1003        end
1004
1005        ENDRET: begin
1006        sel_pcX=2'b01;
1007        ldpcX=1'b1;
1008        stateX=ENDRET2;
1009        end
1010
1011        ENDRET2: begin
1012        stateX=ENDRET3;
1013        end
1014
1015        ENDRET3: begin
1016        ldpcX=1'b1;
1017        ldirX=1'b1;
1018        stateX=FETCH;
1019        end
1020
1021        ENDIT: begin
```

```
1022          ldpcX=1'b1;
1023          ldirX=1'b1;
1024          stateX=FETCH;
1025          end
1026
1027       endcase
1028    end
1029
1030    endmodule//TRAMBLAZE.v
1031
1032
1033
```