Complete Code of all Modules involved in SAP-1

Accumulator

```
/*
  accumulator for SAP1
  @author Bibek Shrestha
  ---------------------------
  in = input (WBUS),
  out = tristate output with EA as enable
  outStd = standard two state output to be connected with addersubtracter
  EA = tristate enable active(high)
  /LA = latch active(low) mistake in first picture which has active(high)
  CLK = clock
*/
module accumulator(out, outStd, in, EA, LA, CLK);
  parameter wordSize = 8;

  output [wordSize-1:0] out, outStd;
  input [wordSize-1:0] in;
  input EA, LA, CLK;

  reg [wordSize-1:0] outStd;
  wire [wordSize-1:0] out;

  // tristate output => out
  assign out = (EA)?outStd:8'hzz;

  always @(negedge CLK)
  begin
    if (!LA)
      outStd = in;
  end
endmodule
```

Adder Subtracter

```
/*
  8 bit Adder subtracter
  ----------------------
  @author Bibek Shrestha
  EU active(high)
  SU subtract(high)
  on subtract does inputA - inputB
*/
module addersubtractor(out, inputA, inputB, SU, EU);
  parameter wordSize = 8;
  parameter addressSize = 4;

  output [wordSize-1:0] out;
  input [wordSize-1:0] inputA;
  input [wordSize-1:0] inputB;
  input SU;
  input EU;

  wire [wordSize-1:0] out;
  assign out = (EU)?( (SU)?(inputA - inputB):(inputA + inputB)):8'bzz;
endmodule
```

Binary Display

```
/*
 * Binary Display Module For SAP1 Computer
 * @author Bibek Shrestha
 * ---------------------------
 * some for of output device
 * contains register whose value is to be monitored
 */
module binarydisplay(in);
  input[7:0] in;
  always @(in)
    $display("The final result in Accumulator is, %h", in);
endmodule
```

Clock

```
/*
 * Clock Generator for SAP-1
 * @author Abhishek Dutta
 * 50 Hz ( 1000ms / 20ms = 50 times in 1 sec ) clock Source for SAP1 computer
 * each T state lasts for 40ms (2*20ms)
```

```
 */
module clk(c);
  output c;
  reg c;

  initial
    c = 1;  // clock begins at low state


  always begin
    #20; c = ~c;
  end
endmodule
Control Unit (Controller)
/*
 * Module to implement the Controller/Sequencer Logic for SAP-1 computer
 * @author: Abhishek Dutta, Bibek Shrestha
 * May 21, 2007 07:00
 * ------------------
 * nRUN    - start SAP1 computer
 * nRESET    - reset SAP1 computer
 */
module cu( nRUN, nRESET );
  input nRUN, nRESET;
  reg[7:0] WBUSreg;                // WBUS - the shared bus for SAP1 computer
  reg CLR;                         // CLEAR signal for SAP1 computer

  wire[7:0] WBUS = WBUSreg[7:0];      // wire tapped into whole (8 bit) WBUS bus
  wire[3:0] WBUSlower = WBUSreg[3:0];  // wire tapped into lower nibble ( WBUS3-WBUS0 ) of W

  /*
   * Master clock initialization
   */
  wire clk;
  clk CLK( clk );

  /*
   * Ring Counter initialization
   */
  wire[6:1] T;                     // T states
  rc RC(clk, ~CLR, T);

  /*
   * Program Counter block initialization
   */
  reg Cp, Ep;
  pc PC(Cp,~clk,~CLR,Ep,WBUSlower);

  /*
   * 16x8 RAM block initialization
   */
  reg nCE;              // Chip Enable for 16x8 RAM
  reg RAM_nWE;          // WE = 1 as only read operations will be performed on RAM
  wire RAM_nOE;
  wire[3:0] RAMaddress;    // Contents of MAR fetched from PC

  // instance of mem16 module
  mem16K MEM( WBUS, RAMaddress, nCE, RAM_nWE, RAM_nOE );

  /*
   * INPUT and MAR block initialization
   */
  reg nLm;
  inputMAR MAR( nLm, clk, WBUSlower, RAMaddress, RAM_nOE );

  /*
   * Instruction Register initialization
   */
  reg nLi;
  reg nEi;
  wire[3:0] OPCODE;      // contains the OPCODE extracted from current IR contents
  wire[3:0] ADDRESS;      // contains the ADDRESS extracted from current IR contents

  ir IR( nLi, clk, CLR, nEi, WBUS, WBUSlower, OPCODE, ADDRESS );

  /*
```

```
153     * Accumulator Initialization
154     */
155    reg nLa;
156    reg Ea;
157    wire[7:0] outStd;    // Output to Adder/Subtractor
158    accumulator ACC( WBUS, outStd, WBUS, Ea, nLa, clk);
159
160    /*
161     * Register B Initialization
162      */
163    reg nLb;
164    wire[7:0] wrBout;
165    register registerB( wrBout, WBUS, nLb, clk);
166
167    /*
168     * AdderSubtracter
169     */
170    reg Su, Eu;
171    addersubtractor ADDSUB(WBUS, outStd, wrBout, Su, Eu);
172
173    /*
174     * Binary Display Initialize
175     */
176    reg nLo;
177    wire[7:0] wrOutput;
178    register outputReg( wrOutput, WBUS, nLo, clk);
179    binarydisplay BinDisplay( wrOutput );
180
181    initial begin
182      #0;
183      #0; Cp = 0; Ep = 0; CLR = 0;
184      #0; nLm = 1; nLb = 1;
185      #0; nCE = 1;
186      #0; nLi = 1; nEi = 1;
187      #0; RAM_nWE = 1;        // in SAP1, we will never write to our 16x8 RAM
188      #0; Ea = 0; nLa = 1;
189      #0; Su = 0; Eu = 0; nLo = 1;
190    end
191
192    always @(negedge clk)
193      begin
194
195      // start the computer only when nRUN = 0
196      if( nRUN == 0 )
197        begin
198
199        /*
200         * Fetch Cycle
201         * consists of T1, T2, T3 states
202         * this cycle will be commong to all opcodes
203         */
204
205          // NOTE:   this delay (#1) required so that the change in value
206          //       of T becomes stable
207          #1;
208
209          if( T == 6'b000001 )        // T1 state
210            begin
211              Eu = 0; Ep = 1;   nLm = 0;      // MAR <- PC
212              // all other signals inactive
213              Cp = 0; nCE = 1; nLi = 1; nEi = 1; nLa = 1; Ea = 0; nLo = 1;
214            end
215
216          else if( T == 6'b000010 )        // T2 state
217            begin
218              Cp = 1;                // PC <- PC + 1
219              // all other signals inactive
220              nLm = 1; Ep = 0;
221            end
222
223          else if( T == 6'b000100 )        // T3 state
224            begin
225              nCE = 0; nLi = 0;          // IR <- [MAR]
226              Cp = 0;
227            end
228
```

```
229         /*
230          * Now decode the opcode field to execute the OPCODE specific
231          * control unit sequence
232          */
233
234           else if( T == 6'b001000 )        // T4 state
235             begin
236               #1;    // allows the settling of WBUS
237               nCE = 1; nLi = 1; // disable RAM and IR
238
239               if(OPCODE == 4'b0000)
240                 // ie: OPCODE = LDA
241                 begin
242                   nEi = 0; nLm = 0;
243                 end
244
245               if(OPCODE == 4'b0001)
246                 // ie: OPCODE = ADD
247                 begin
248                   nLm = 0; nEi = 0;
249                 end
250
251               if(OPCODE == 4'b0010)
252                 // ie: OPCODE = SUB
253                 begin
254                   nLm = 0; nEi = 0;
255                 end
256
257               if(OPCODE == 4'b1110)
258                 // ie: OPCODE = OUT
259                 begin
260                   Ea = 1; nLo = 0;
261                 end
262
263               if(OPCODE == 4'b1111)
264                 // ie: OPCODE = HLT
265                 begin
266                   $stop;
267                           // control sequence for HLT
268                 end
269
270             end
271
272           else if( T == 6'b010000 )        // T5 state
273             begin
274               #1;         // allows the settling of WBUS
275               if(OPCODE == 4'b0000)
276                 // ie: OPCODE = LDA
277                 begin
278                   nLa = 0; nCE = 0;
279                   nEi = 1;    nLm = 1; // resent T4 on wires
280                 end
281
282               if(OPCODE == 4'b0001)
283                 // ie: OPCODE = ADD
284                 begin
285                   nEi = 1;    nLm = 1; // reset T4 on wires
286                   nLb = 0; nCE = 0;
287                 end
288
289               if(OPCODE == 4'b0010)
290                 // ie: OPCODE = SUB
291                 begin
292                   nEi = 1;    nLm = 1; // reset T4 on wires
293                   nLb = 0; nCE = 0;
294                 end
295
296               if(OPCODE == 4'b1110)
297                 // ie: OPCODE = OUT
298                 begin
299                   // control sequence for OUT
300                 end
301
302               if(OPCODE == 4'b1111)
303                 // ie: OPCODE = HLT
304                 begin
```

```
305                 // control sequence for HLT
306               end
307           end
308
309         else if( T == 6'b100000 )        // T6 state
310           begin
311             #1;          // allows the settling of WBUS
312             if(OPCODE == 4'b0000)
313               // ie: OPCODE = LDA
314               begin
315                 // No Operation State (NOP)
316                 Cp = 0; Ep = 0; CLR = 0; nLm = 1; nCE = 1; nLi = 1; nEi = 1; nLa = 1; nCE =
317               end
318
319             if(OPCODE == 4'b0001)
320               // ie: OPCODE = ADD
321               begin
322                 nLb = 1; nCE = 1; // Clear Remains of T5
323                 nLa = 0; Eu = 1; Su = 0;
324               end
325
326             if(OPCODE == 4'b0010)
327               // ie: OPCODE = SUB
328               begin
329                 nLb = 1; nCE = 1; // Clear Remains of T5
330                 nLa = 0; Eu = 1; Su = 1;
331               end
332
333             if(OPCODE == 4'b1110)
334               // ie: OPCODE = OUT
335               begin
336                 // control sequence for OUT
337               end
338
339             if(OPCODE == 4'b1111)
340               // ie: OPCODE = HLT
341               begin
342                 // control sequence for HLT
343               end
344           end
345
346       end
347
348     // reset the computer when nRESET = 0
349     if( nRESET == 0 )
350       begin
351         CLR = 1;
352       end
353   end
354 endmodule
355
356 // test bench for Control Unit/ Sequencer
357 module cu_tb();
358   reg nRUN, nRESET;
359
360   cu CU(nRUN, nRESET);
361
362   initial begin
363     // Initialize the control signals
364     #0; nRUN = 0; nRESET = 1;
365   end
366 endmodule
367
368 Input MAR
369 /*
370  * Module to implement "Input and MAR" block of SAP-1 computer
371  * @author: Abhishek Dutta
372  * May 20, 2007 07:25
373  * ------------------
374  * nLm        - Loads the lower nibble of WBUS in MAR
375  * WBUSlower   - lower nibble (ie: WBUS0 - WBUS 3) of WBUS
376  * RAMaddress - 4 bit address of RAM block stored by MAR
377  * RAM_nOE    - Output Enable signal for 16x8 RAM block
378  */
379 module inputMAR( nLm, clk, WBUSlower, RAMaddress, RAM_nOE );
380   input nLm,clk;
```

```
381    input[3:0] WBUSlower;
382
383    output[3:0] RAMaddress;
384    reg[3:0] RAMaddress;
385
386    output RAM_nOE;
387    wire RAM_nOE;
388    reg tempReg;
389
390    initial begin
391      RAMaddress = 4'hz;
392      tempReg = 1;
393    end
394
395    always @(posedge clk)
396      begin
397        if ( nLm == 0)
398          begin
399            RAMaddress[3:0] = WBUSlower[3:0];
400            tempReg = 0;
401          end
402      end
403    assign RAM_nOE = (tempReg) ? 1:0;
404  endmodule
405
406  Instruction Register
407  /*
408   * Module to implement "Instruction Register[IR]" block of SAP-1 computer
409   * @author: Abhishek Dutta
410   * May 20, 2007 17:25
411   * ------------------
412   * nLi      - Load 8 bit data from WBUS
413   *            higher nibble (WBUS7 - WBUS4)  = OPCODE
414   *           lower nibble (WBUS3 - WBUS0)    = ADDRESS
415   * clr      - Clears the contents of IR
416   * nEi      - Loads the lower nibble of IR (ie: address field) in lower WBUS
417   * WBUS    - 8 bit data retrived from WBUS ( contents of IR = OPCODE + ADDRESS )
418   * WBUSlower- 4 bit address field of IR ( ie: lower nibble of IR = ADDRESS )
419   */
420  module ir( nLi, clk, clr, nEi, WBUS, WBUSlower, OPCODE, ADDRESS );
421    input nLi;
422    input clk;
423    input clr;
424    input nEi;
425    input[7:0] WBUS;
426
427    output[3:0] WBUSlower;
428    output[3:0] OPCODE;
429    output[3:0] ADDRESS;
430
431    reg[7:0] IR;      // internal copy of the contents of IR
432    reg[3:0] OPCD;
433    reg[3:0] ADDR;
434
435    initial begin
436      IR = 8'hzz;
437      OPCD = 4'bzzzz;
438      ADDR = 4'bzzzz;
439    end
440
441    always @(posedge clk)
442      begin
443        if( nLi == 0 )
444          begin
445            IR = WBUS;
446            OPCD = IR[7:4];
447            ADDR = IR[3:0];
448          end
449        if( clr == 1 )
450          begin
451            IR = 8'hzz;
452            OPCD = 4'bzzzz;
453            ADDR = 4'bzzzz;
454          end
455      end
456    assign WBUSlower = (nEi) ? 4'bzzzz : ADDR;
```

```
457   assign OPCODE = (1)? OPCD : 4'bzzzz;
458   assign ADDRESS = (1)? ADDR : 4'bzzzz;
459
460 endmodule
461
462 Memory 16 Bytes
463 /*
464   SAPI RAM module
465   Asynchronus RAM block
466   16 x 8 static TTL RAM
467   ----------------------------------
468   address bus of 4 bit
469   databus of 8 bit
470   /CS = active low
471   /WE = active low
472   /OE = active low
473   @author Bibek Shrestha
474   ----------------------------------
475 */
476 module mem16K(data, address, CS, WE, OE);
477   parameter wordSize = 8;
478   parameter addressSize = 4;
479
480   inout [wordSize-1:0] data;
481   input [addressSize-1:0] address;
482   input CS, WE, OE;
483
484   reg [wordSize-1:0] dataReg;
485   reg [wordSize-1:0] memory [0:1<<addressSize];
486
487   // attach a tristate buffer to the databus
488   // if OE is active(low), connect datawires to the dataRegisters
489   // if OE is inactive(high), set datawires to high impedance for input
490
491   assign data = (!OE && WE && !CS)?dataReg:8'hzz;
492
493   // the reason cs is not put in the condition is
494   // internal wires are in accordance with databus
495   // but the data is not stored in memory so no problem
496   initial
497   begin
498     /* execution of a small program */
499     memory[0] = 8'h09;  // LDA 9H
500     memory[1] = 8'h1A;  // ADD AH
501     memory[2] = 8'h1B;  // ADD BH
502     memory[3] = 8'h2C;  // SUB CH
503     memory[4] = 8'hE0;  // OUT
504     memory[5] = 8'hF0;  // HLT
505
506       memory[9] = 8'h01;
507     memory[10] = 8'h02;
508     memory[11] = 8'h03;
509     memory[12] = 8'h04;
510   end
511
512   always @(WE or OE or CS)
513   begin
514     // incase of active(low) CS and active(low) WE
515     if (!CS && !WE)
516     begin
517       #1;       memory[address] = data;
518     end
519   end
520   always @(WE or OE or CS)
521   begin
522       if (!OE && !CS && WE)
523         dataReg = memory[address];
524   end
525 endmodule
526 Program Counter
527 /*
528  * Module to implement the Program Counter Logic for SAP-1 computer
529  * @author: Abhishek Dutta
530  * May 19, 2007 21:00
531  * ------------------
532  * WBUSlower  - lower nibble of WBUS (WBUS0 to WBUS3)
```

```
533  * Cp        - causes Program Counter (PC) to increment by 1
534  * nCLK      - inverted clock signal
535  * nCLR      - to reset PC
536  * Ep        - outputs the contents of PC to WBUSlower
537  */
538 module pc( Cp, nCLK, nCLR, Ep, WBUSlower);
539   input Cp;
540   input nCLK;
541   input nCLR;
542   input Ep;
543
544   output[3:0] WBUSlower;
545
546   reg[3:0] nextPC;
547
548   initial begin
549     nextPC = 4'b0000;      // PC starts from zero
550   end
551
552   always @(negedge nCLK)      // negative edge of nCLK means positive edge of CLK
553   begin
554     if( Cp == 1)
555       /* Increment PC on every negative edge of nCLK
556        * (ie: positive edge of CLK - mid point of single T state
557        */
558       begin
559         nextPC = nextPC + 1;
560       end
561     if ( nCLR == 0)
562       /* Reset Program Counter */
563       begin
564         nextPC = 4'b0000;
565       end
566   end
567   /*
568    * Keep WBUS in high impedence state when Ep is low (during this time some other module is
569    * When Ep is high, output the contents of PC to WBUS
570    */
571
572   assign WBUSlower = (Ep) ? nextPC : 4'bzzzz;
573
574 endmodule
575
576 Ring Counter
577 /*
578  * Module to implement the Ring Counter Logic for SAP-1 computer
579  * @author: Abhishek Dutta
580  * May 21, 2007 06:56
581  * ------------------
582  * clk       - Clock of SAP1
583  * nCLR     - Clear the contents of Ring counter and restart count from T1 state
584  * T         - Counts the T states (3 bit register that counts from 000 to 101)
585  *             T[1] = 1 for first T state, T[2] = 1 for second T state , ........
586  *             and on T[6], it resets back to T1. Each instruction has 6 T states time to exe
587  */
588 module rc(clk, nCLR, T);
589   input clk;
590   input nCLR;
591   output[6:1] T;
592   reg[6:1] T;
593
594   initial begin
595     T[6:1] = 6'b000000;
596   end
597
598   always @(negedge clk)
599     begin
600       if( nCLR == 0)
601         T = 6'b000001;
602       else
603       begin
604         case(T)
605           6'b000000: T = 6'b000001;
606           6'b000001: T = 6'b000010;
607           6'b000010: T = 6'b000100;
608           6'b000100: T = 6'b001000;
```

```
609             6'b001000: T = 6'b010000;
610             6'b010000: T = 6'b100000;
611             6'b100000: T = 6'b000001;    // restart counting T states
612           endcase
613         end
614       end
615 endmodule
616
617 Register B
618 /*
619   general register
620   to be used as RegB in SAP1
621   ---------------------------
622   @author Bibek Shrestha
623   out [7:0],
624   in [7:0],
625   LB - active(low)
626   CLK - clock
627
628 */
629 module register(out, in, LB, CLK);
630   parameter wordSize = 8;
631
632   output [wordSize-1:0] out;
633   input [wordSize-1:0] in;
634   input LB, CLK;
635
636   reg [wordSize-1:0] out;
637
638   initial
639     out = 8'b00;
640
641   always @(negedge CLK)
642   begin
643     if (!LB)
644       // store input => out
645       out = in;
646   end
647 endmodule
```