**CSE4117 – Microprocessors**



**HOMEWORK-3**

**ALU WITH ACCUMULATOR**

**NAME SURNAME** **STUDENT NUMBER**

Aydın DUYGU 150118981

Tugay SARICI 150119829

**EXPLANATIONS**

In this homework we designed a structure consists of a control unit, an accumulator and an alu.So that one of two operands of the alu is the output of the accumulator.

The accumulator is -on the other hand- an eight bit register which stores the result coming from the alu.The alu uses accumulator’s output and an another input which comes from instruction register as the literal part of the instruction.

The control unit produces approprite signals which are ‘fetch’, ‘decode’ and ‘execute’.To produce these signals respectively we implemented a final state machine in the control unit.

After fetch fase completed we see the relating instruction on the instruction register.

After decode fase completed we see the opcode and the literal part as outputs of the instructrion register and inputs of the alu.

We simulated our model and we obtained successfull results.

Below secreenshots are presented relating our simulation:

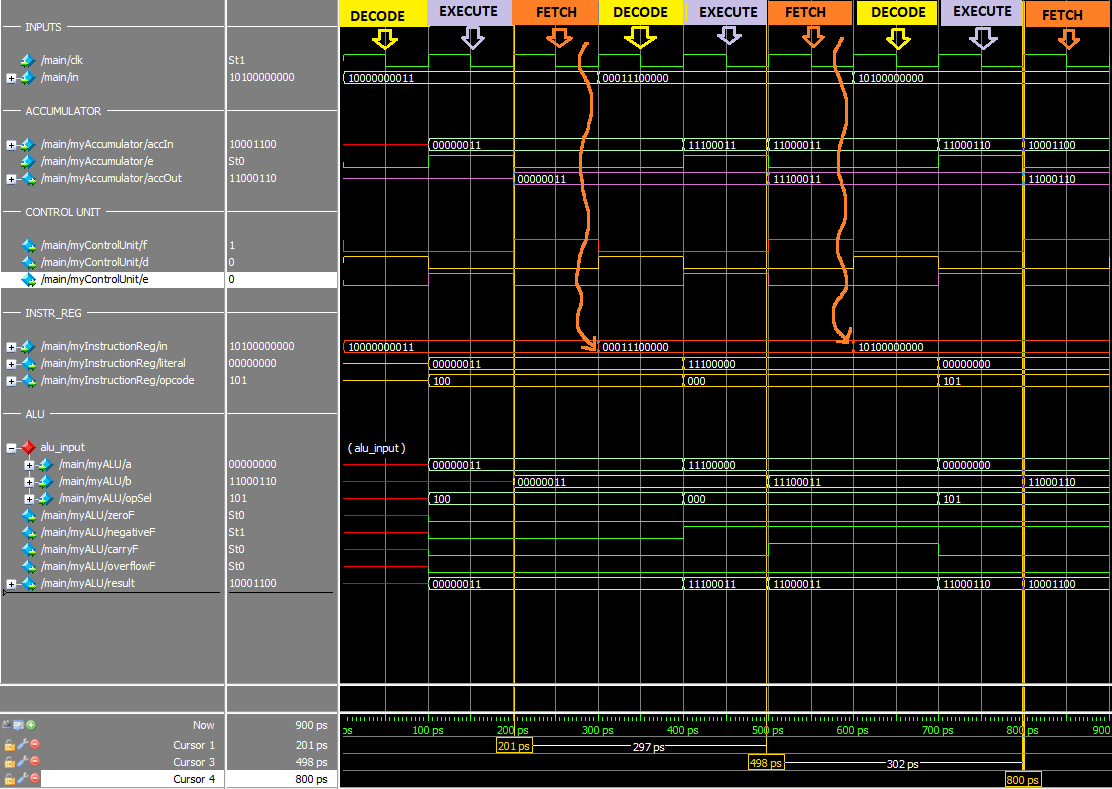
In the simulation below there are three instructions executed which are:

* MoveA 3 (100 00000011) => //Accumulator=3
* AddA 224 (000 11100000) => // Accumulator=Accumulator+224
* SHFLA (101 00000000) => //Accumulator=Accumulator<<

**Diagram, timeline

Description automatically generated**

Below you can see that instruction register is set after fetch fase completed:



Below you can see that opcode and literal values coming into alu are set after decode fase completed: Diagram

Description automatically generated

Below you can also see that result of the accumulator is set after execute fase completed:

Diagram

Description automatically generated

**Our Sourcode:**

|  |
| --- |
| // Aydın DUYGU - 150118981  // Tugay SARICI - 150119829  //---------------------------------  module onebitsubstractor(input a, input b, input c\_in, output c\_out, output result);  assign c\_out = (~a&b)|(b&c\_in)|(~a&c\_in);  assign result = (a ^ b) ^ c\_in;  endmodule  //---------------------------------  module twobitsubstractor(input [1:0]a,input [1:0]b, input c\_in, output c\_out, output [1:0] result);  wire cout;  onebitsubstractor s0(a[0],b[0],c\_in,cout,result[0]);  onebitsubstractor s1(a[1],b[1],cout,c\_out,result[1]);  endmodule  //---------------------------------  module fourbitsubstractor(input [3:0]a,input [3:0]b, input c\_in, output c\_out, output [3:0] result);  wire cout;  twobitsubstractor s0(a[1:0],b[1:0],c\_in,cout,result[1:0]);  twobitsubstractor s1(a[3:2],b[3:2],cout,c\_out,result[3:2]);  endmodule  //---------------------------------  module eightbitsubstractor(input [7:0]a,input [7:0]b, input c\_in, output c\_out, output [7:0] result);  wire cout;  fourbitsubstractor s0(a[3:0],b[3:0],c\_in,cout,result[3:0]);  fourbitsubstractor s1(a[7:4],b[7:4],cout,c\_out,result[7:4]);  endmodule  //---------------------------------  module fulladder(  input [7:0] a,  input [7:0] b,  input c\_in,  output reg c\_out,  output reg [7:0] resultFullAdder);  always @(a or b or c\_in)  begin  {c\_out, resultFullAdder} = a+b+c\_in;  end  endmodule  //---------------------------------  module mux2to1(in0,in1,sel0,out);  input[7:0] in0,in1;  input sel0;  output[7:0] out;  assign out[7] = (in0[7] & ~sel0)|(in1[7] & sel0);  assign out[6] = (in0[6] & ~sel0)|(in1[6] & sel0);  assign out[5] = (in0[5] & ~sel0)|(in1[5] & sel0);  assign out[4] = (in0[4] & ~sel0)|(in1[4] & sel0);  assign out[3] = (in0[3] & ~sel0)|(in1[3] & sel0);  assign out[2] = (in0[2] & ~sel0)|(in1[2] & sel0);  assign out[1] = (in0[1] & ~sel0)|(in1[1] & sel0);  assign out[0] = (in0[0] & ~sel0)|(in1[0] & sel0);  endmodule  //---------------------------------  module mux4to1(in0,in1,in2,in3,sel0,sel1,out);  input[7:0] in0,in1,in2,in3;  input sel0,sel1;  output[7:0] out;  wire[7:0] w1,w2;  mux2to1 m1(in0,in1,sel0,w1);  mux2to1 m2(in2,in3,sel0,w2);  mux2to1 m3(w1,w2,sel1,out);  endmodule  //---------------------------------  module mux\_8to1(in0,in1,in2,in3,in4,in5,in6,in7,sel0,sel1,sel2,out);  input[7:0] in0,in1,in2,in3,in4,in5,in6,in7;  input sel0,sel1,sel2;  output[7:0] out;  wire[7:0] w1,w2;  mux4to1 m1(in0,in1,in2,in3,sel0,sel1,w1);  mux4to1 m2(in4,in5,in6,in7,sel0,sel1,w2);  mux2to1 m3(w1,w2,sel2,out);  endmodule  //---------------------------------  module alu(a, b, opSel, zeroF, carryF, negativeF, overflowF, result);  input [7:0] a, b;  input [2:0] opSel;  output zeroF, negativeF, carryF, overflowF;  output [7:0] result;  wire [7:0] result0, result1, result2, result3, result4, result5, result6, result7;  wire carryFS, carryFA;  fulladder f(a, b, 0, carryFA, result0);  eightbitsubstractor s(a, b, 0, carryFS, result1);  assign result2 = a & b;  assign result3 = a | b;  assign result4 = a;  assign result5 = b << 1;  assign result6 = b >> 1;  assign result7 = a ^ b;  //---------------------------------  mux\_8to1 mux1(result0, result1,result2, result3, result4, result5, result6, result7, opSel[0], opSel[1], opSel[2], result);  assign zeroF = ~(result[7] | result[6] | result[5] | result[4] | result[3] | result[2] | result[1] | result[0]);  assign negativeF = result[7] ;  assign overflowF = (((~a[7] & ~b[7] & result[7]) | (a[7] & b[7] & ~result[7])) & (~opSel[0] & ~opSel[1] & ~opSel[2])) |  (((~a[7] & b[7] & result[7]) | (a[7] & ~b[7] & ~result[7])) & (opSel[0] & ~opSel[1] & ~opSel[2]));    assign carryF = (~opSel[0] & ~opSel[1] & ~opSel[2] & carryFA) | (opSel[0] & ~opSel[1] & ~opSel[2] & carryFS);    endmodule  //---------------------------------  module accumulator(accIn, e, clk, accOut);  input [7:0] accIn;  input e, clk;  output [7:0] accOut;  reg [7:0] accOut;  always @ (posedge clk) begin  if(e == 1)  accOut = accIn;  end  endmodule  //---------------------------------  module instruction\_register(in, d, clk, literal, opcode);  // opcode: 3 bits + value: 8 bits  input [10:0] in;  input d, clk;  output [7:0] literal;  reg [7:0] literal;  output [2:0] opcode;  reg [2:0] opcode;  always @ (posedge clk)  begin  if (d == 1) begin  literal[0] <= in[0];  literal[1] <= in[1];  literal[2] <= in[2];  literal[3] <= in[3];  literal[4] <= in[4];  literal[5] <= in[5];  literal[6] <= in[6];  literal[7] <= in[7];  opcode[0] <= in[8];  opcode[1] <= in[9];  opcode[2] <= in[10];  end  end  endmodule  //-----------------------------------------------------------------------------  module controlUnit(in,clk,state,e,d,f);  input [10:0] in;  input clk;  output e,d,f;  output [1:0] state;  reg [1:0] state = 2'b01;  f=(~state[1])&(~state[0]);  d=(~state[1])&(state[0]);  e=(state[1])&(~state[0]);  always @ (posedge clk)  begin  state[0]<= (~state[0])&(~state[1]);  state[1]<= (~state[1])&(state[0]);  end  endmodule  //---------------------------------  module main(clk,in,zeroF, carryF, negativeF, overflowF, accOut,f);  input clk;  input [10:0] in;  wire [7:0] literal,aluResult;  wire [2:0] opcode;  wire e,d;  output f;  wire k;    output zeroF, carryF, negativeF, overflowF;  output [7:0] accOut;  controlUnit myControlUnit(in,clk,state,e,d,k);  instruction\_register myInstructionReg(in,d, clk, literal, opcode);  alu myALU(literal, accOut, opcode, zeroF, carryF, negativeF, overflowF, aluResult);  accumulator myAccumulator(aluResult, e, clk, accOut);  assign f=k;  endmodule  //--------------------------------- |

Finally our RTL View (in closed shape) are presented below:

Diagram

Description automatically generated