Central Connecticut State University

**BUILDING A COMPLETE   
SINGLE-CYCLE DATAPATH**

**Report 2**

prepared by

Thi Minh Huyen Le

Brendan Manzolli

Erik Marrero

Fall - 2021

1. **Tasks:**

Thi Minh Huyen Le: Instruction Memory, Combine Code.

Brendan Manzolli: Multiplexes, Diagrams

Erik Marrero: Branch Control, MainControl

1. **Description**

Instructions:

| Instruction (I-Type: addi) | Opcode  (4 bits) [15-12] | Rs  (2 bits) [11-10] | Rt (2 bits) [9-8] | Value or Address  (8 bits) [7-0] | Binary Instruction |
| --- | --- | --- | --- | --- | --- |
| lw $1, 0($0) | 1001 | 00 | 01 | 00000000 | 1001000100000000 |
| lw $2, 2($0) | 1001 | 00 | 10 | 00000010 | 1001001000000010 |
| beg $3,$0,IMemory[6] | 1011 | 00 | 11 | 00000010 | 1011001100000010 |
| sw $1,2($0) | 1010 | 00 | 01 | 00000010 | 1010000100000010 |
| sw $2,0($0) | 1010 | 00 | 10 | 00000000 | 1010001000000000 |
| lw $1,0($0) | 1001 | 00 | 01 | 00000000 | 1001000100000000 |
| lw $2,2($0) | 1001 | 00 | 10 | 00000010 | 1001001000000010 |
| bne $3,$0,IMemory[6] | 1100 | 00 | 11 | 00000010 | 1100110000000010 |

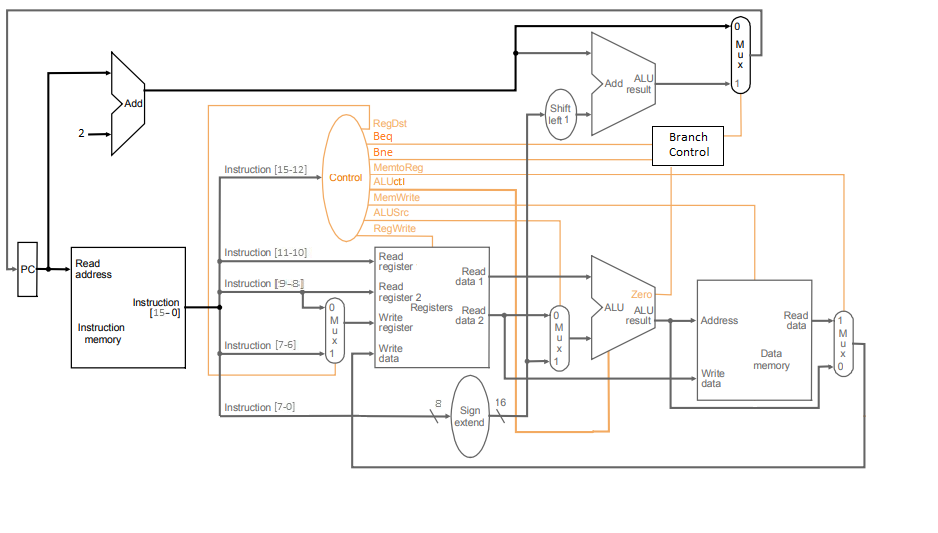
| Instruction (R-Type) | Opcode  (4 bits) [15-12] | Rs  (2 bits) [11-10] | Rt  (2bit) [9-8] | Rd  (2bits) [7-6] | Unused  (6 bits) [5-0] | Binary Instruction |
| --- | --- | --- | --- | --- | --- | --- |
| slt $3, $1, $2 | 0100 | 01 | 10 | 11 | 000000 | 0100011011000000 |
| sub $3, $1, $2 | 0001 | 01 | 10 | 11 | 000000 | 0001011011000000 |

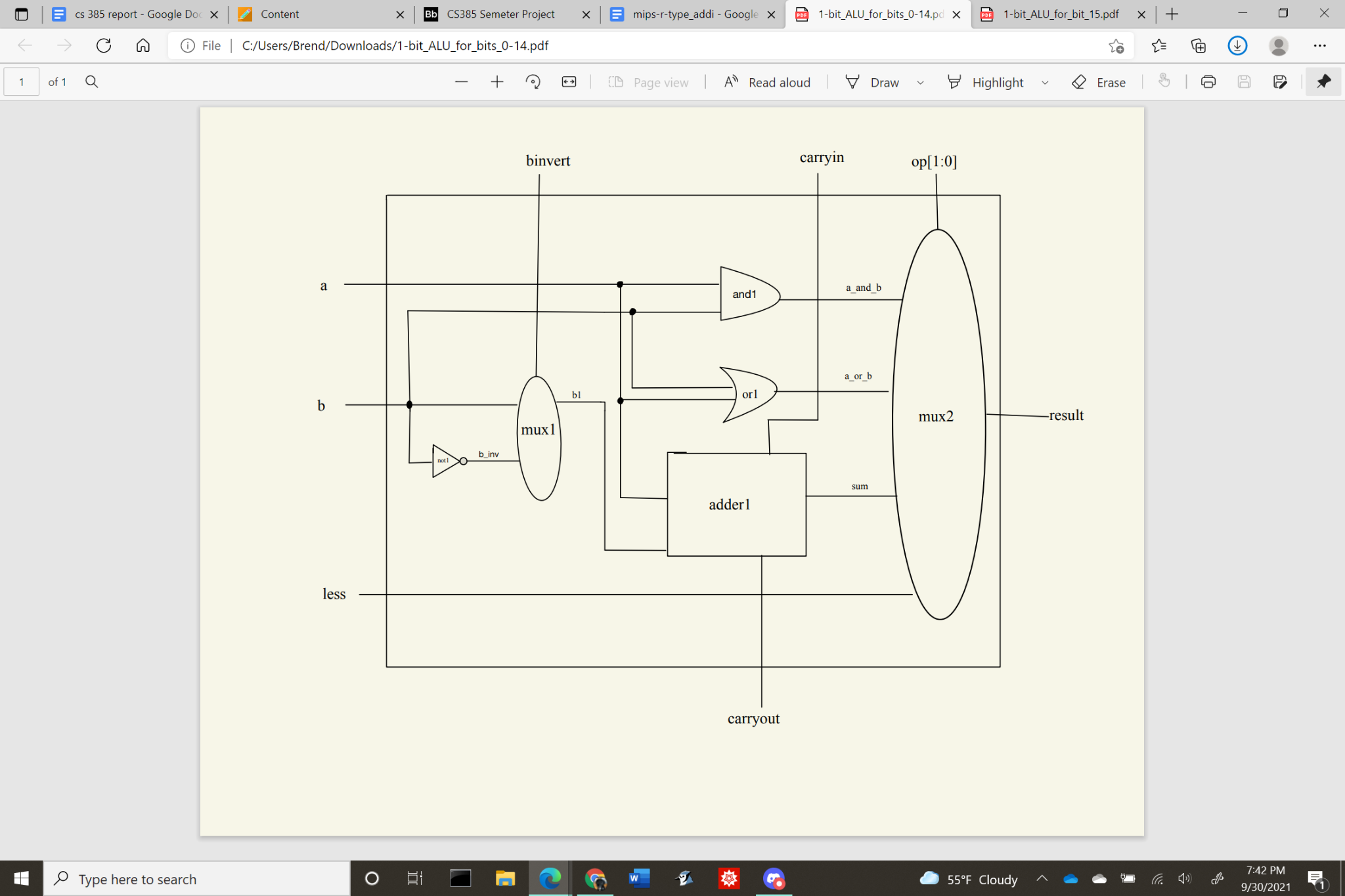
Instruction Set:

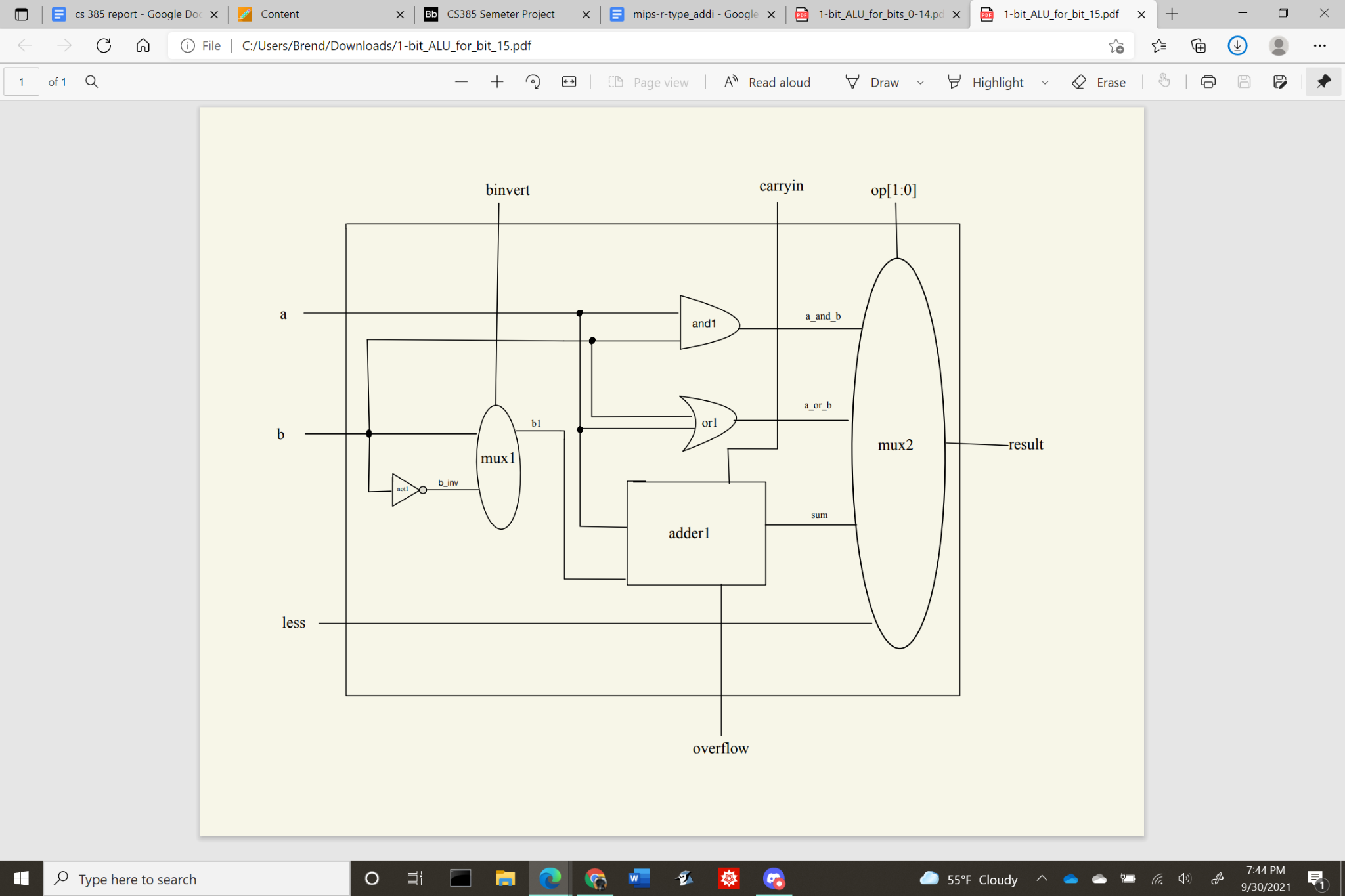
| **Instruction** | **Opcode** |
| --- | --- |
| add | 0000 |
| sub | 0001 |
| and | 0010 |
| or | 0011 |
| slt | 0100 |
| addi | 0101 |
| slti | 0110 |
| andi | 0111 |
| ori | 1000 |
| lw | 1001 |
| sw | 1010 |
| beg | 1011 |
| bne | 1100 |

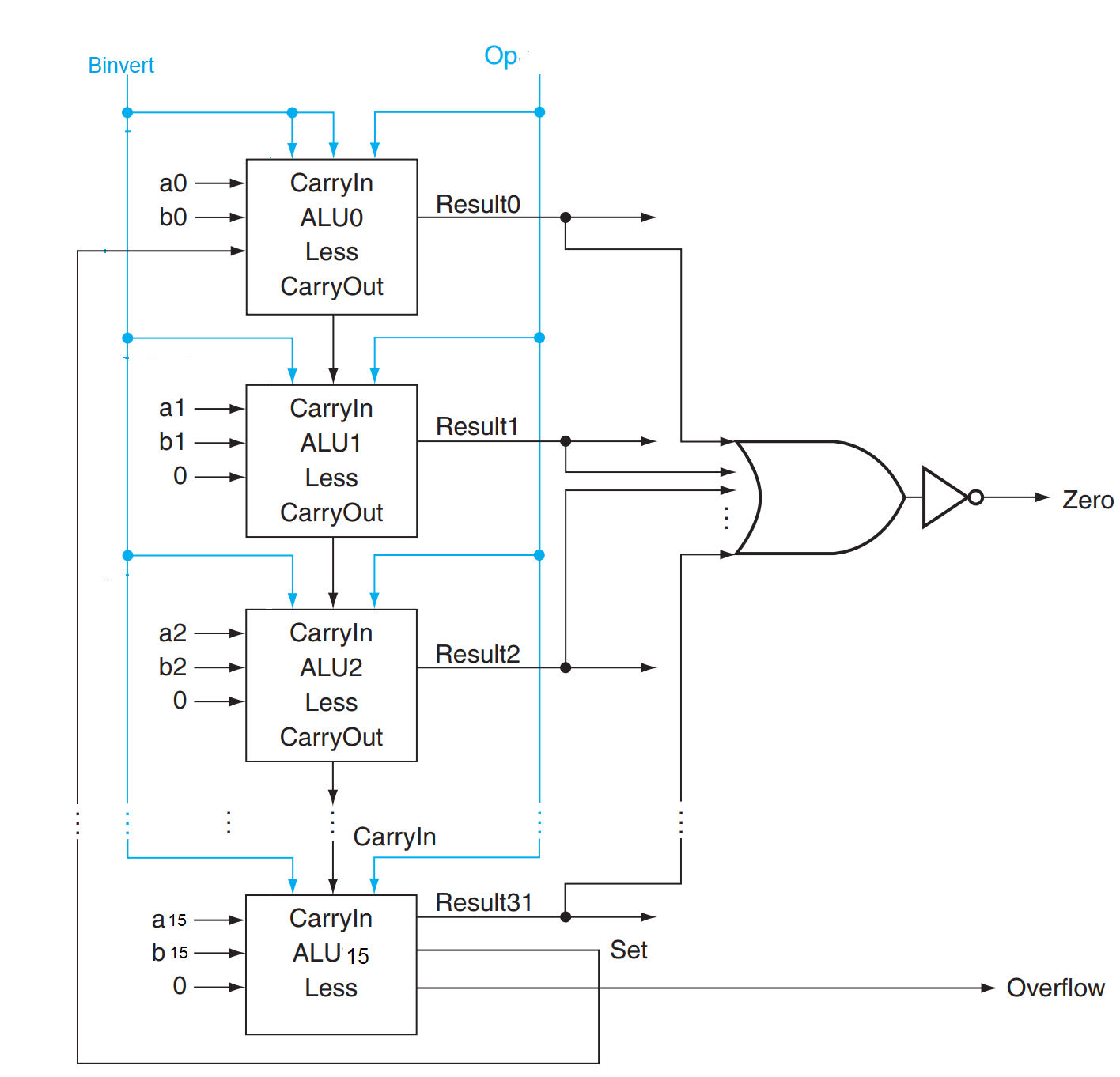
1. **Block logic diagrams**

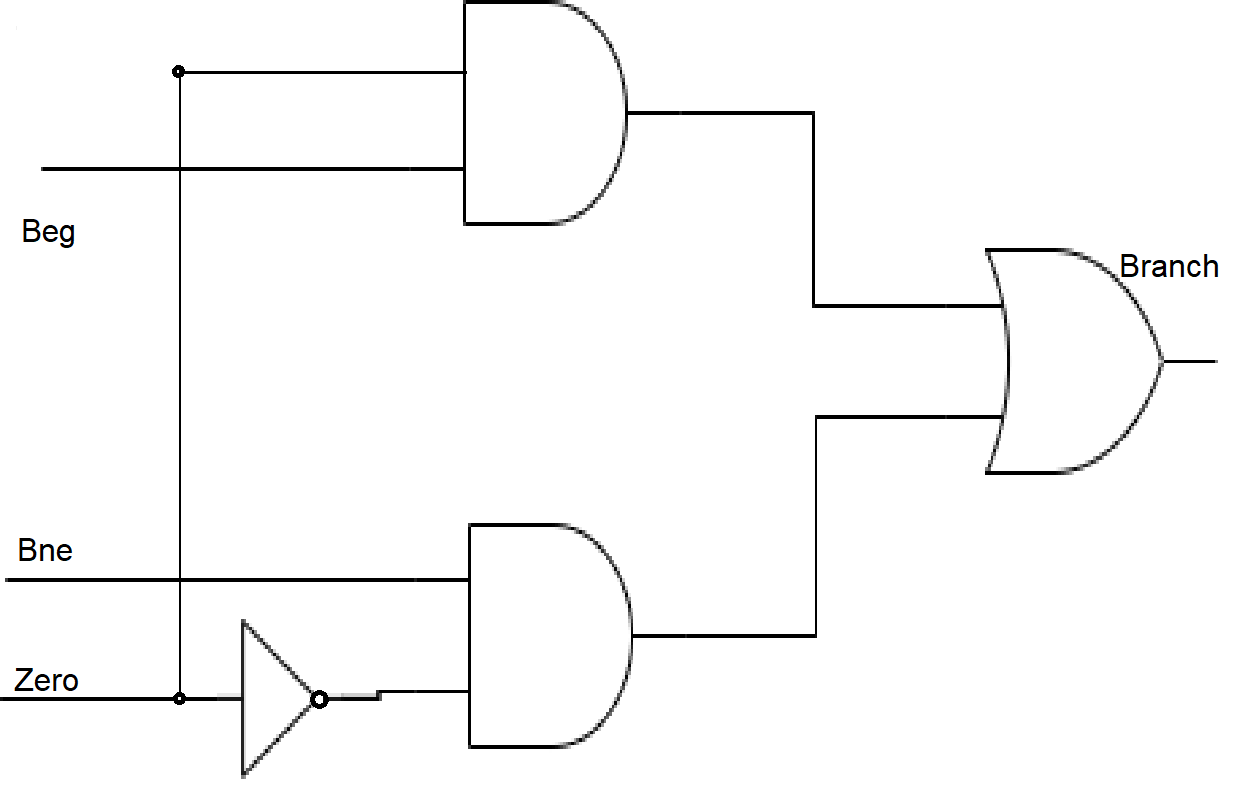
**CPU**

****

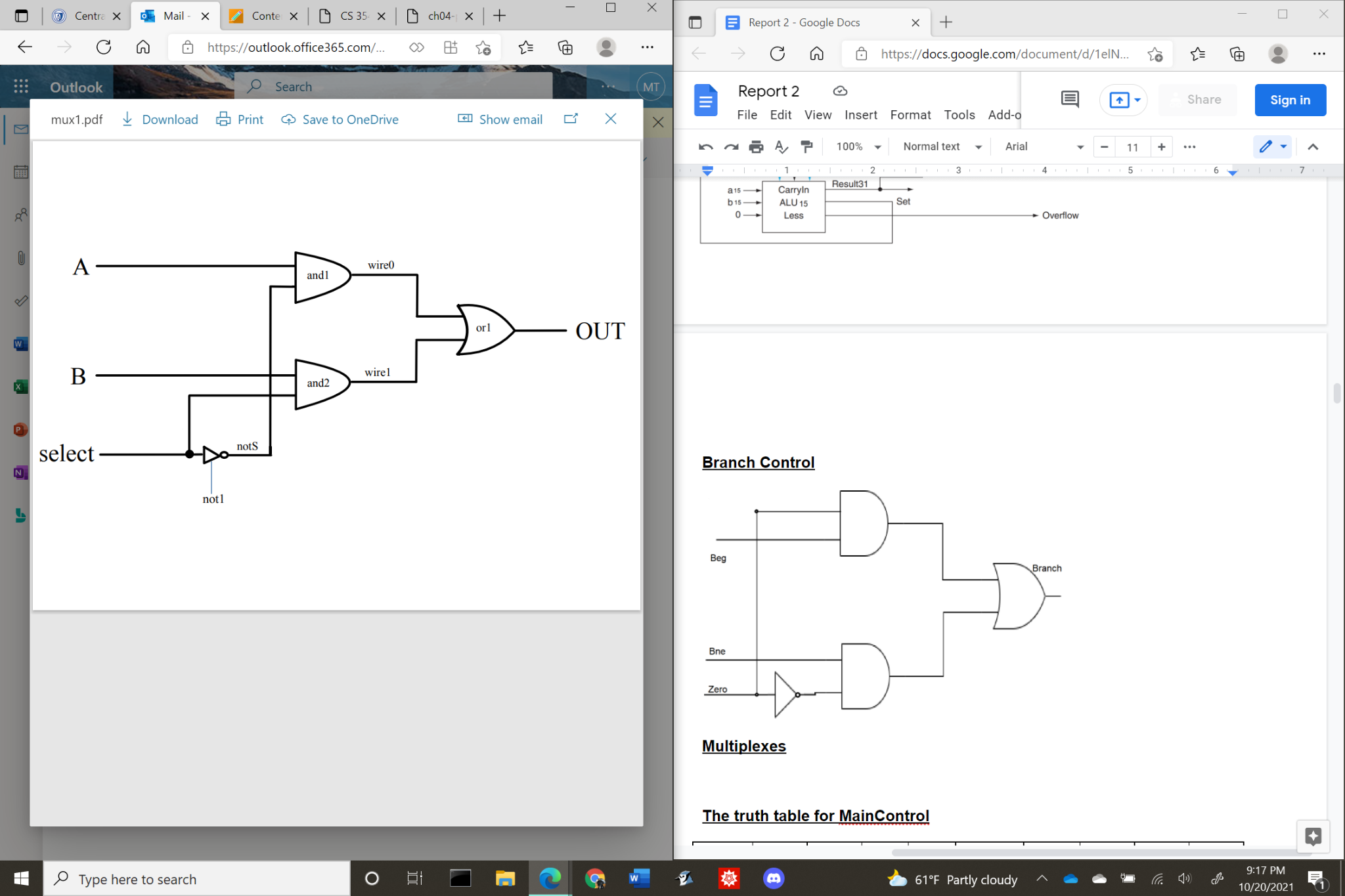
**1-bit ALU for bits 0-14**

**1-bit ALU for bit 15**

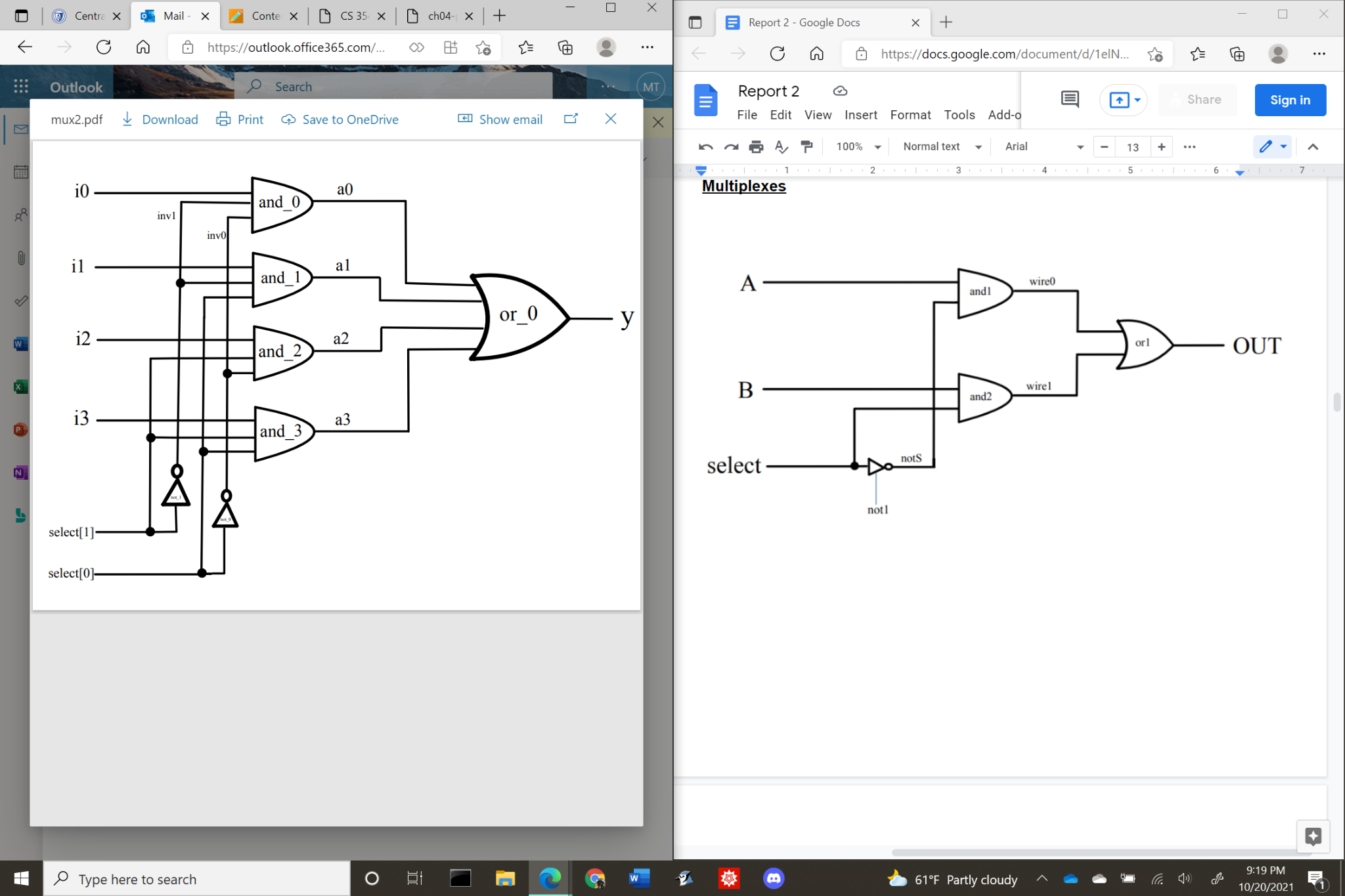
**16bit-ALU** 

**Branch Control**

**2x1 Multiplexer**

****

**4x1 Multiplexer**

****

**The truth table for MainControl**

| **Instruction** | **OpCode** | **RegDst** | **Beg** | **Bne** | **MemtoReg** | **MemWrite** | **ALUSrc** | **RegWrite** | **ALUCtl** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **add** | **0000** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **010** |
| **sub** | **0001** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **110** |
| **and** | **0010** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **000** |
| **or** | **0011** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **001** |
| **slt** | **0100** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **111** |
| **addi** | **0101** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **010** |
| **slti** | **0110** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **111** |
| **andi** | **0111** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **000** |
| **ori** | **1000** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **001** |
| **lw** | **1001** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **010** |
| **sw** | **1010** | **0** | **0** | **0** | **0** | **1** | **1** | **0** | **010** |
| **beg** | **1011** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **110** |
| **bne** | **1100** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **110** |

1. **Verilog source code**

| //Group : Thi Minh Huyen Le, Brendan Manzolli, Erik Marrero  // Behavioral model of MIPS - single cycle implementation, R-types and addi,slti,anddi,ori  // --------------REGISTER FILE ---------------  // Assign RR1, RR2 to RD1, RD2  // When RegWrite is on, assign WD to WR  module reg\_file (RR1,RR2,WR,WD,RegWrite,RD1,RD2,clock);  input [1:0] RR1,RR2,WR;  input [15:0] WD;  input RegWrite,clock;  output [15:0] RD1,RD2;  reg [15:0] Regs[0:3];  assign RD1 = Regs[RR1];  assign RD2 = Regs[RR2];  initial Regs[0] = 0;  always @(negedge clock)  if (RegWrite==1 & WR!=0)  Regs[WR] <= WD;  endmodule  //-----------END REGISTER FILE-----------  //--------------ALU----------------------  // A 32-bit ALU constructed from the 31 copies of the 1-bit ALU in the top and one 1-bit ALUmsb in the bottom.  module ALU (op,a,b,result,zero);  input [15:0] a;  input [15:0] b;  input [2:0] op;  output [15:0] result;  output zero;  wire c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16;    ALU1 alu0 (a[0],b[0],op[2],op[1:0],set, op[2], c1,result[0]);  ALU1 alu1 (a[1],b[1],op[2],op[1:0],1'b0, c1, c2,result[1]);  ALU1 alu2 (a[2],b[2],op[2],op[1:0],1'b0, c2, c3,result[2]);  ALU1 alu3 (a[3],b[3],op[2],op[1:0],1'b0, c3, c4,result[3]);  ALU1 alu4 (a[4],b[4],op[2],op[1:0],1'b0, c4, c5,result[4]);  ALU1 alu5 (a[5],b[5],op[2],op[1:0],1'b0, c5, c6,result[5]);  ALU1 alu6 (a[6],b[6],op[2],op[1:0],1'b0, c6, c7,result[6]);  ALU1 alu7 (a[7],b[7],op[2],op[1:0],1'b0, c7, c8,result[7]);  ALU1 alu8 (a[8],b[8],op[2],op[1:0],1'b0, c8, c9,result[8]);  ALU1 alu9 (a[9],b[9],op[2],op[1:0],1'b0, c9, c10,result[9]);  ALU1 alu10 (a[10],b[10],op[2],op[1:0],1'b0, c10, c11,result[10]);  ALU1 alu11 (a[11],b[11],op[2],op[1:0],1'b0, c11, c12,result[11]);  ALU1 alu12 (a[12],b[12],op[2],op[1:0],1'b0, c12, c13,result[12]);  ALU1 alu13 (a[13],b[13],op[2],op[1:0],1'b0, c13, c14,result[13]);  ALU1 alu14 (a[14],b[14],op[2],op[1:0],1'b0, c14, c15,result[14]);  ALUmsb alu15 (a[15],b[15],op[2],op[1:0],1'b0, c15, c16,result[15],set);    or or1(or01, result[0], result[1], result[2], result[3], result[4], result[5], result[6], result[7], result[8], result[9], result[10], result[11], result[12], result[13], result[14], result[15]);  not not1(zero, or01);  endmodule  // 1-bit ALU for bits 0-2  // A 1-bit ALU that performs AND, OR, addition , subtraction on a and b,  module ALU1 (a,b,binvert,op,less,carryin,carryout,result);  input a,b,less,carryin,binvert;  input [1:0] op;  output carryout,result;  wire sum, a\_and\_b, a\_or\_b, b\_inv;    not not1(b\_inv, b);  mux2x1 mux1(b,b\_inv,binvert,b1);  and and1(a\_and\_b, a, b);  or or1(a\_or\_b, a, b);  fulladder adder1(sum,carryout,a,b1,carryin);  mux4x1 mux2(a\_and\_b,a\_or\_b,sum,less,op[1:0],result);  endmodule  // 1-bit ALU for the most significant bit  // A 1-bit ALU that performs AND, OR, addition , subtraction on a and b, GIVE THE MOST SIGNIFICANT BIT AS AN OUTPUT SUM  module ALUmsb (a,b,binvert,op,less,carryin,overflow,result,sum);  input a,b,less,carryin,binvert;  input [1:0] op;  output overflow,result,sum;  wire sum, a\_and\_b, a\_or\_b, b\_inv;    not not1(b\_inv, b);  mux2x1 mux1(b,b\_inv,binvert,b1);  and and1(a\_and\_b, a, b);  or or1(a\_or\_b, a, b);  fulladder adder1(sum,overflow,a,b1,carryin);  mux4x1 mux2(a\_and\_b,a\_or\_b,sum,less,op[1:0],result);  endmodule  module halfadder (S,C,x,y);  input x,y;  output S,C;  xor (S,x,y);  and (C,x,y);  endmodule  module fulladder (S,C,x,y,z);  input x,y,z;  output S,C;  wire S1,D1,D2;  halfadder HA1 (S1,D1,x,y),  HA2 (S,D2,S1,z);  or g1(C,D2,D1);  endmodule  // Multiplexor two 1bit-inputs  module mux2x1(A,B,select,OUT);  input A,B,select;  output OUT;  wire wire0, wire1;  not not1 (notS,select);  and and1 (wire0, A,notS),  and2 (wire1, B,select);  or or1 (OUT,wire0,wire1);  endmodule  // Multiplexor four 1bit-inputs  module mux4x1(i0,i1,i2,i3,select,y);  output y; // Output  input i0, i1, i2, i3; // Input ports.  input [1:0] select; // Select lines.    // intermediate wires  wire inv0, inv1; // Inverter outputs.  wire a0, a1, a2, a3; // AND gates outputs.    // Inverters.  not not\_1 (inv1, select[1]);  not not\_0 (inv0, select[0]);    // 4-input AND gates.  and and\_0 (a0, i0, inv1, inv0);  and and\_1 (a1, i1, inv1, select[0]);  and and\_2 (a2, i2, select[1], inv0);  and and\_3 (a3, i3, select[1], select[0]);    // 4-input OR gate.  or or\_0 (y, a0, a1, a2, a3);  endmodule  //--------------END ALU-----------------  //--------------MULTIPLEXORS-------------  // Multiplexor two 16-bit inputs.  module mux16bitx2x1 (A,B,Select,Out);  input [15:0] A,B;  input Select;  output [15:0] Out;  mux2x1 m0 (A[0],B[0],Select,Out[0]),  m1 (A[1],B[1],Select,Out[1]),  m2 (A[2],B[2],Select,Out[2]),  m3 (A[3],B[3],Select,Out[3]),  m4 (A[4],B[4],Select,Out[4]),  m5 (A[5],B[5],Select,Out[5]),  m6 (A[6],B[6],Select,Out[6]),  m7 (A[7],B[7],Select,Out[7]),  m8 (A[8],B[8],Select,Out[8]),  m9 (A[9],B[9],Select,Out[9]),  m10 (A[10],B[10],Select,Out[10]),  m11 (A[11],B[11],Select,Out[11]),  m12 (A[12],B[12],Select,Out[12]),  m13 (A[13],B[13],Select,Out[13]),  m14 (A[14],B[14],Select,Out[14]),  m15 (A[15],B[15],Select,Out[15]);  endmodule  // Multiplexor two 2-bit inputs.  module mux2bitx2x1 (A,B,Select,Out);  input [1:0] A,B;  input Select;  output [1:0] Out;  mux2x1 m0 (A[0],B[0],Select,Out[0]),  m1 (A[1],B[1],Select,Out[1]);  endmodule  //--------------END MULTIPLEXORS------------  //--------------MAIN CONTROL---------------  /\*\*\* 16-bit CPU control source code \*\*\*/  module mainCtrl (op, ctrl);  input [3:0] op;  output reg [9:0] ctrl;  // ctrl bits: RegDst, Beg, Bne, MemtoReg, MemWrite, ALUSrc, RegWrite, ALUCtl(3bits)  always @(op) case (op)  4'b0000: ctrl <= 10'b1000\_001\_010; // ADD  4'b0001: ctrl <= 10'b1000\_001\_110; // SUB  4'b0010: ctrl <= 10'b1000\_001\_000; // AND  4'b0011: ctrl <= 10'b1000\_001\_001; // OR  4'b0100: ctrl <= 10'b1000\_001\_111; // SLT  4'b0101: ctrl <= 10'b0001\_011\_010; // ADDI  4'b0110: ctrl <= 10'b0001\_011\_111; // SLTI  4'b0111: ctrl <= 10'b0001\_011\_000; // ANDI  4'b1000: ctrl <= 10'b0001\_011\_001; // ORI  4'b1001: ctrl <= 10'b0001\_011\_010; // LW  4'b1010: ctrl <= 10'b0000\_110\_010; // SW  4'b1011: ctrl <= 10'b0100\_000\_110; // BEG  4'b1100: ctrl <= 10'b0010\_000\_110; // BNE  endcase  endmodule  //------------END MAIN CONTROL----------------  //------------BRANCH CONTROL---------------  module branchcontrol(Zero,Beg,Bne,Branch);  input Zero,Beg,Bne;  output Branch;  wire invZero,a1,a2;  not not1 (invZero,Zero);  and a1 (a1,Zero,Beg),  a2 (a2,invZero,Bne);  or or1 (Branch,a1,a2);  endmodule  //------------END BRANCH CONTROL---------------  //------------CPU--------------  module CPU (clock,WD,IR,PC);  input clock;  output [15:0] WD,IR,PC;  reg[15:0] PC, IMemory[0:1023], DMemory[0:1023];  wire [15:0] IR,SignExtend,NextPC,RD2,A,B,ALUOut,PCplus4,Target;  wire [1:0] WR;  wire [2:0] ALUCtl;  wire [1:0] ALUOp;  initial begin  // Program: swap memory cells and compute absolute value  IMemory[0] = 16'b1001\_00\_01\_00000000; // lw $1, 0($0)  IMemory[1] = 16'b1001\_00\_10\_00000010; // lw $2, 2($0)  IMemory[2] = 16'b0100\_01\_10\_11\_000000; // slt $3, $1, $2  IMemory[3] = 16'b1011\_11\_00\_00000010; // beq $3, $0, IMemory[6]  //IMemory[3] = 16'b1100\_11\_00\_00000010; // bne $3, $0, IMemory[6]  IMemory[4] = 16'b1010\_00\_01\_00000010; // sw $1, 2($0)  IMemory[5] = 16'b1010\_00\_10\_00000000; // sw $2, 0($0)  IMemory[6] = 16'b1001\_00\_01\_00000000; // lw $1, 0($0)  IMemory[7] = 16'b1001\_00\_10\_00000010; // lw $2, 2($0)  IMemory[8] = 16'b0001\_01\_10\_11\_000000; // sub $3, $1, $2  // Data  DMemory [0] = 16'b0101; // swap the cells and see how the simulation output changes  DMemory [1] = 16'b0111;  end  initial PC = 0;  assign IR = IMemory[PC>>1];  assign SignExtend = {{8{IR[7]}},IR[7:0]}; // sign extension  reg\_file rf (IR[11:10],IR[9:8],WR,WD,RegWrite,A,RD2,clock);  ALU fetch (3'b010,PC,16'b10,PCplus4,Unused1);  ALU ex (ALUCtl, A, B, ALUOut, Zero);  ALU branch (3'b010,SignExtend<<1,PCplus4,Target,Unused2);  // MainCtrl convert Input:4-bit OpCode to Output: 10-bit  mainCtrl MainCtr (IR[15:12],{RegDst, Beg, Bne, MemtoReg, MemWrite, ALUSrc, RegWrite,ALUCtl});  // assign WR  mux2bitx2x1 muxWR (IR[9:8], IR[7:6], RegDst, WR);  // assign WD  mux16bitx2x1 muxWD (ALUOut, DMemory[ALUOut>>1], MemtoReg, WD);  // assign B  mux16bitx2x1 muxB (RD2, SignExtend, ALUSrc, B);  branchcontrol branchCtl(Zero,Beg,Bne,Branch);  mux16bitx2x1 muxBranch (PCplus4, Target, Branch, NextPC);  always @(negedge clock) begin  PC <= NextPC;  if (MemWrite) DMemory[ALUOut>>1] <= RD2;  end  endmodule  // Test module  module test ();  reg clock;  wire [15:0] WD,IR,PC;  CPU test\_cpu(clock,WD,IR,PC);  always #1 clock = ~clock;    initial begin  $display ("clock PC IR WD (Binary) WD (Integer)");  $monitor ("%b %2d %20b %20b %10d", clock,PC,IR,WD, WD);  clock = 1;  #16 $finish;  end  endmodule |
| --- |

1. **Test result**

***Sample Test 1 : DMemory[0] = 5 DMemory[1] = 7 with Beg instruction***

***lw $1, 0($0) => $1 = 5***

***lw $2, 2($0)=> $2 = 7***

***slt $3, $1, $2=> $3 =1***

***beg $3, $0, IMemory[6]=> branch does not take***

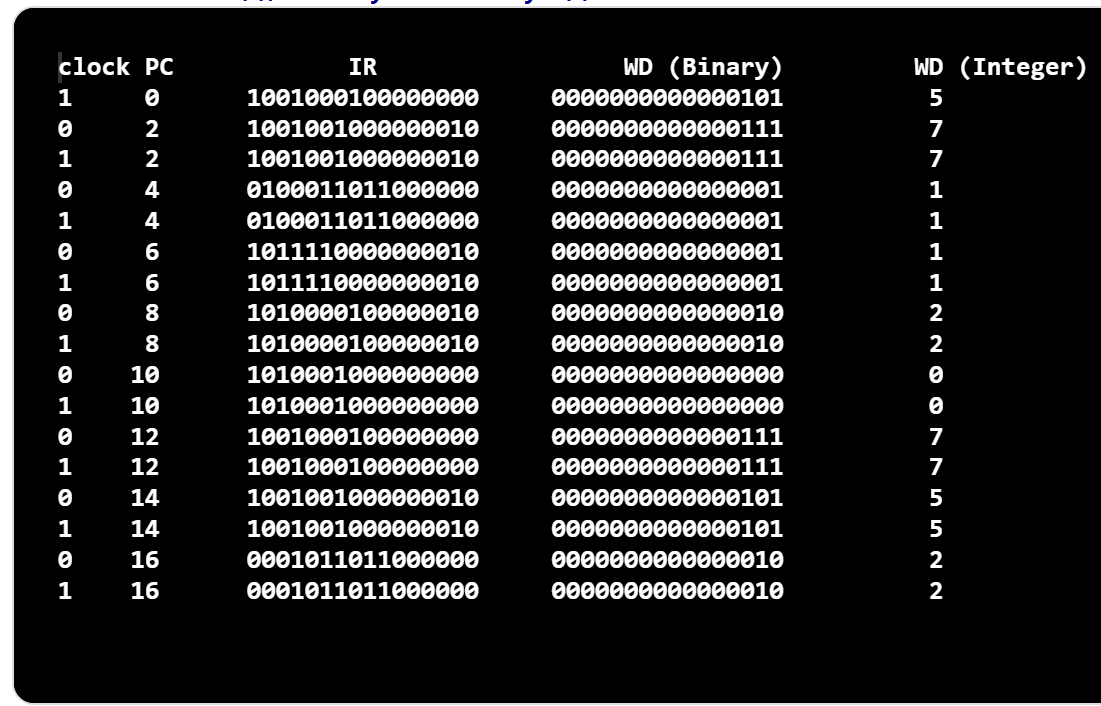
***sw $1, 2($0) => 2($0)=7***

***sw $2, 0($0) => 0($0)=5***

***lw $1, 0($0) => $1=5***

***lw $2, 2($0) => $2=7***

***sub $3, $1, $2 => $3 = 7-5 = 2 in binary 0000\_0000\_0000\_0010***



***Sample Test 2 : DMemory[0] = 7 DMemory[1] = 5 with Beg instruction***

***lw $1, 0($0) => $1 = 7***

***lw $2, 2($0)=> $2 = 5***

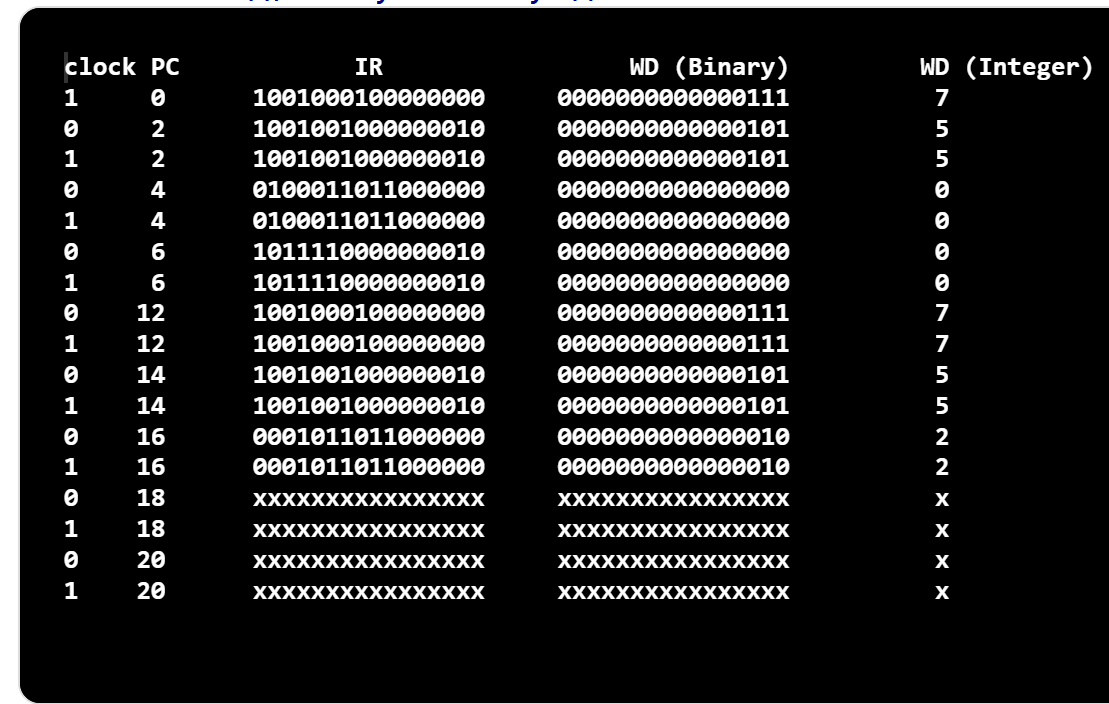
***slt $3, $1, $2=> $3 =0***

***beg $3, $0, IMemory[6]=> branch taken***

***lw $1, 0($0) => $1=7***

***lw $2, 2($0) => $2=5***

***sub $3, $1, $2 => $3 = 7-5 = 2 in binary 0000\_0000\_0000\_0010***

******

***Sample Test 3 : DMemory[0] = 5 DMemory[1] = 7 with Bne instruction***

***lw $1, 0($0) => $1 = 5***

***lw $2, 2($0)=> $2 = 7***

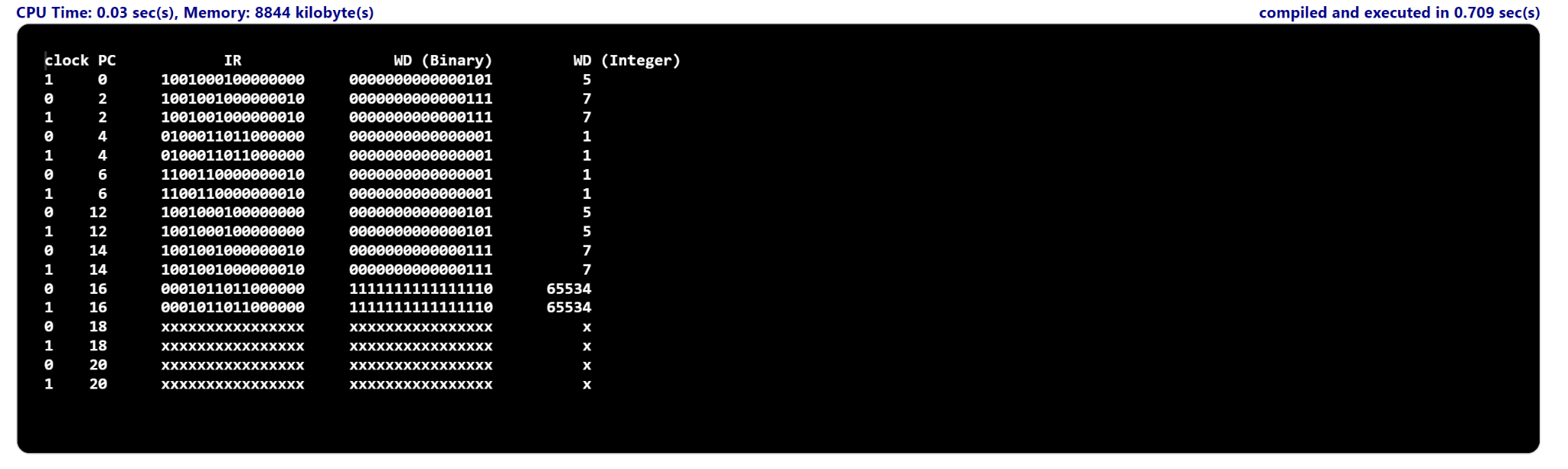
***slt $3, $1, $2=> $3 =1***

***bne $3, $0, IMemory[6]=> branch taken***

***lw $1, 0($0) => $1=5***

***lw $2, 2($0) => $2=7***

***sub $3, $1, $2 => $3 = 5-7 = -2 in binary 1111\_1111\_1111\_1110***

******

***Sample Test 4 : DMemory[0] = 7 DMemory[1] = 5 with Bne instruction***

***lw $1, 0($0) => $1 = 7***

***lw $2, 2($0)=> $2 = 5***

***slt $3, $1, $2=> $3 =0***

***bne $3, $0, IMemory[6]=> branch does not take***

***sw $1, 2($0) => 2($0)=7***

***sw $2, 0($0) => 0($0)=5***

***lw $1, 0($0) => $1=5***

***lw $2, 2($0) => $2=7***

***sub $3, $1, $2 => $3 = 5-7 = -2 in binary 1111\_1111\_1111\_1110***

