Control Unit Module Design – Structural Style Module Description

The control module design may be described in a ***behavioral*** or a ***structural*** style. The project can use modules described in both styles and both constructs can be used inside the same module.

1. Behavioral Style (not entirely synthesizable, simulation)

Example:

```
always@(opcode)
begin
    if (opcode == 4'b0011) //Store
    begin
            regWrite = 0;
            memtoReg = 0;
            PCtoReg= 0;
            branchN = 0;
            branchZ = 0;
            jump = 0;
            jumpMem=0;
            memRead = 0;
            memWrite = 1;
            aluOp = 4'b0100;
    end
end
```

2. Structural Style (synthesizable, hardware implementation)

The following module design is based on the DMUX and gates implementation (Figure 1) of the truth table presented in Figure 2 (partially).

```
##################################################################################
```

module ***control_clc***(opcode, regWrite, memtoReg, PCtoReg, branchN, branchZ, jump, jumpMem, ALU, memRead, memWrite);

```
input [3:0] opcode;

output [3:0] ALU;

output regWrite, memtoReg, PCtoReg, branchN, branchZ, jump, jumpMem, memRead, memWrite;

wire nS0, nS1, nS2, nS3;

wire [15:0] OUT;


integer E=1;

always@(opcode) begin

   if(opcode[0]===1'bZ) E=0;

end


DMUX_1_16 dmux_instance(E, opcode, OUT);

CLC_FUNCTIONS clc_functions_instance(.out(OUT), .rW(regWrite), .m2r(memtoReg), .pc2r(PCtoReg),
.bN(branchN), .bZ(branchZ), .jmp(jump), .jmpM(jumpMem), .alu(ALU), .memR(memRead),
.memW(memWrite));

endmodule

##################################################################################################

module DMUX_1_16(E, opcode, out);

input wire E;

input [3:0] opcode;

output [15:0] out;

assign S[0] = opcode[0];

assign S[1] = opcode[1];

assign S[2] = opcode[2];

assign S[3] = opcode[3];


wire [3:0] S;

wire nS0, nS1, nS2, nS3;
```

```
//DMUX 1:16 Selectors
not(nS0, S[0]);
not(nS1, S[1]);
not(nS2, S[2]);
not(nS3, S[3]);

//DMUX 1:16 Outputs
nand(out[0], nS3, nS2, nS1, nS0, E);
nand(out[1], nS3, nS2, nS1, S[0], E);
nand(out[2], nS3, nS2, S[1], nS0, E);
nand(out[3], nS3, nS2, S[1], S[0], E);
nand(out[4], nS3, S[2], nS1, nS0, E);
nand(out[5], nS3, S[2], nS1, S[0], E);
nand(out[6], nS3, S[2], S[1], nS0, E);
nand(out[7], nS3, S[2], S[1], S[0], E);
nand(out[8], S[3], nS2, nS1, nS0, E);
nand(out[9], S[3], nS2, nS1, S[0], E);
nand(out[10], S[3], nS2, S[1], nS0, E);
nand(out[11], S[3], nS2, S[1], S[0], E);
nand(out[12], S[3], S[2], nS1, nS0, E);
nand(out[13], S[3], S[2], nS1, S[0], E);
nand(out[14], S[3], S[2], S[1], nS0, E);
nand(out[15], S[3], S[2], S[1], S[0], E);
endmodule
##################################################################################
module CLC_FUNCTIONS(out, alu, rW, m2r, pc2r, bN, bZ, jmp, jmpM, memR, memW);
input [15:0] out;
output rW, m2r, pc2r, bN, bZ, jmp, jmpM, memR, memW;
output [3:0] alu;
```

```
nand(rW, out[4], out[5], out[6], out[7], out[14], out[15]);
not(m2r, out[14]);
not(pc2r, out[15]);
not(bN, out[11]);
not(bZ, out[9]);
not(jmp, out[8]);
not(jmpM, out[10]);
nand(memR, out[10], out[14]);
not(memW, out[3]);

//ALU control unit, selectors
nand(alu[3], out[1], out[2]);
nand(alu[2], out[0], out[3], out[8], out[9], out[10], out[11], out[14], out[15]);
nand(alu[1], out[6], out[7]);
nand(alu[0], out[5], out[7]);
endmodule
################################################################################################
```

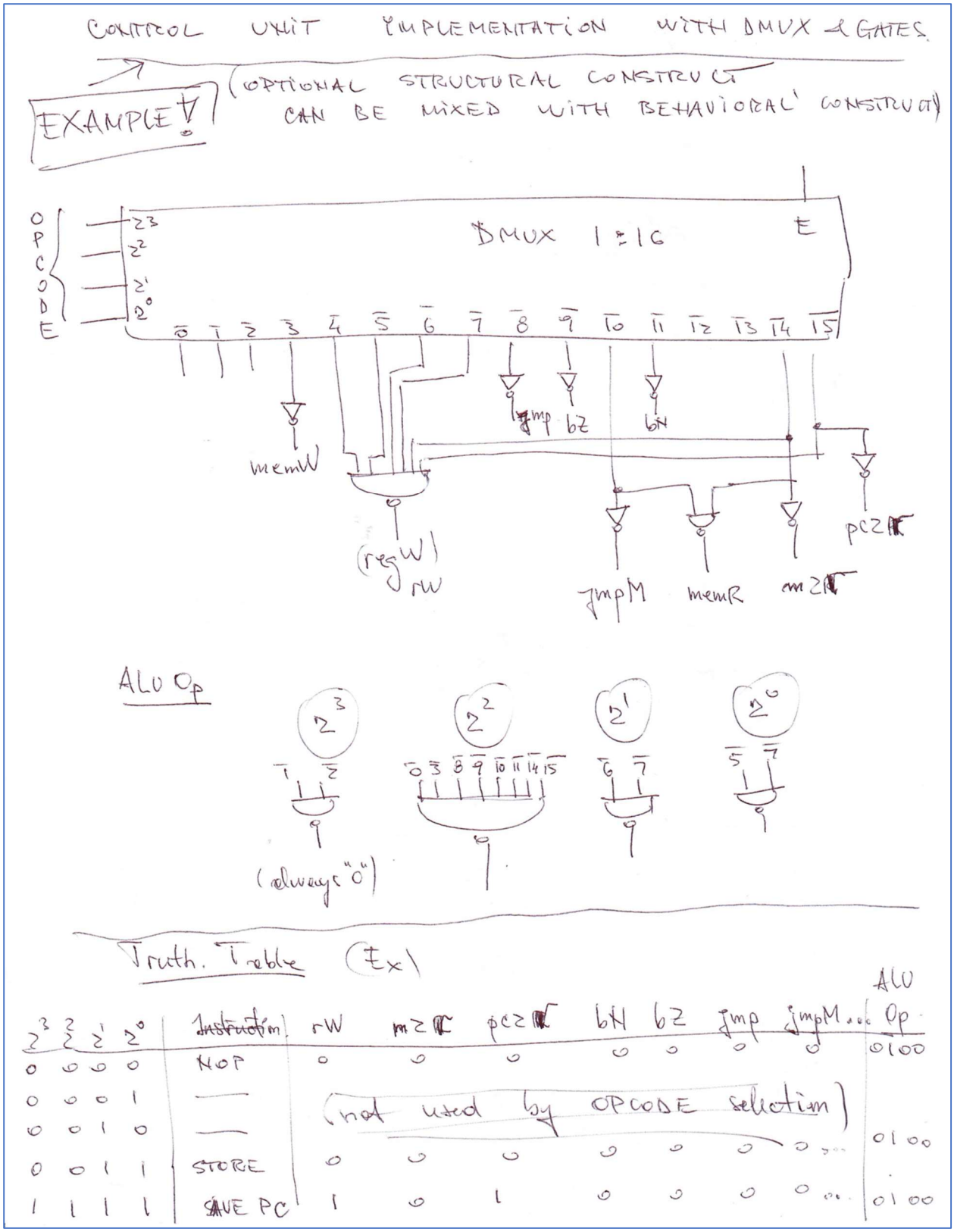

Figure 1: CLC Implementation of the Control Unit (Figure 2 Truth Table, with DMUX & Gates)

Instruction	Op-code	regWrite	memtoReg	PCtoReg	branchN	branchZ	jump	jumpMem	memRead	memWrite	aluOp
NOP	0000	0									0100
Save PC	1111	1									0100
Load	1110	1									0100
Store	0011	0	0	0	0	0	0	0	0	1	0100
Add	0100	1	0	0	0	0	0	0	0	0	0000
Increment	0101	1									0001
Negate	0110	1									0010
Subtract	0111	1									0011
Jump	1000	0									0100
Branch if Z	1001	0									0100
Jump Mem	1010	0									0100
Branch if N	1011	0									0100

Figure 2: Truth Table Example