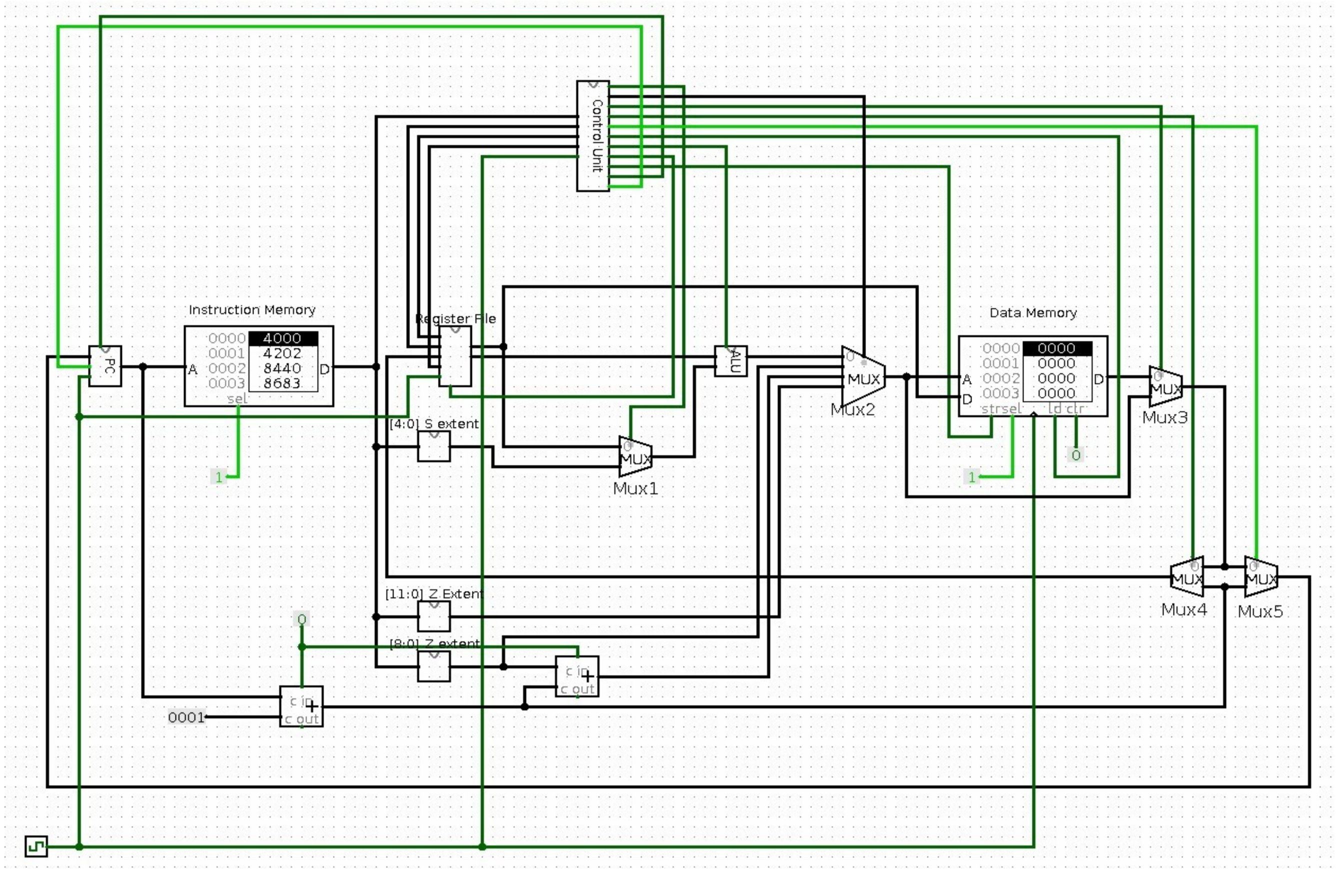
Due Date: 27.12.2013

CSE 325 Term Project

In this project you are expected to design a CPU which supports subset of instructions from architecture known as LC-3 (Little Computer 3). The LC-3 specifies a word size of 16 bits for its registers and uses a 16-bit addressable memory with a 2^{16}-location address space. The register file contains eight registers, referred to by number as R0 through R7. All of the registers are general-purpose in that they may be freely used by any of the instructions that can write to the register file.

The schematic view of datapath for simplified LC-3 processor is given below.



Datapath contains 5 basic parts as *PC* (Program Counter), *Instruction Memory*,*ALU*(Arithmetic Logic Unit), *Register File*,*Control Unit* and *Data Memory*. *PC* holds address of instruction to be executed next. This address is sent to *Instruction Memory* to fetch the instruction. Than fetched instruction is sent to *Control Unit* to produce appropriate signals for all parts in datapath. *ALU* handles arithmetic operations(In our reduced LC-3 case only *ADD* and *AND* operations). Data can be stored to or can be read from *Data Memory*.

The design supports *ADD*,*ADD**,*AND*,*AND**,*LD*,*ST*,*JMP* instructions. Details of the instructions are on the following table.

Due Date: 27.12.2013

	[15:12](opcode)	[11:9]	[8:6]	[5]	[4:3]	[2:0]
ADD	1000	DR	SR1	0	00	SR2
ADD*	1000	DR	SR1	1	imm5	
AND	0010	DR	SR1	0	00	SR2
AND*	0010	DR	SR1	1	imm5	
LD	0100	DR	Address9			
ST	1100	SR	Address9			
JMP	1010	Address12				

Remember our instructions have 16 bits. [15:12] specifies the *opcode*(operation code) of the instruction. Notice that every instruction has distinct opcode. *DR* stands for *Destination Register*,[11:9]. *SR1* for first *Source Register*,[8:6].*SR2* for second *Source Register*,[2:0]. *Imm5* is 5 bit signed immediate value. 5Th bit decides the second operand for *ADD* and *AND* instructions. If 5th bit is 0 second operand is register , if 5th bit is 1 second operand is immediate part from instruction. *Address9* and *Address12* are also immediate values for addresses to be used in instructions *LD*,*ST* and *JMP*. Instruction has following assembly language structure:

ADD DR,SR1,SR2	*; DR=SR1+SR2*
ADD DR,SR1,imm5	*; DR=SR1+Sign Extend[imm5](Sign extend 5 bit immediate to 16 bits)*
AND DR,SR1,SR2	*; DR=SR1&SR2 (bitwise AND)*
AND DR,SR1,imm5	*; DR=SR1&Sign Extend[imm5](Sign extend 5 bit immediate to 16 bits)*
LD DR,Address9	*; DR=DataMemory[ZeroExtend[Address9]]*
ST SR,Address9	*; DataMemory[ZeroExtend[Address9]]=SR*
JMP Address12	*; PC=ZeroExtend[Address12]*

Due Date: 27.12.2013

For example lets see what will be the values for following *ADD* instruction.

ADD R0,R1,R2 ; opcode for *ADD* instruction is 1000. *DR* is *R0* so [11:9]=000. *SR1* is *R1* then [8:6]=001. *SR2* is *R2* than [2:0]=010. Since there is no immediate part in instruction [5]=0. [4:3] is redundant for this instruction so [4:3]=XX. Resulting machine code for this instruction is:

1000 000 001 0 XX 010. Lets put 00 to XX for simplicity. The instruction is 8042h.

Lets try another instruction *AND* with immediate value.

AND R3,R7,#12 ; opcode for *AND* instruction is 0010. *DR* is *R3* so [11:9]=011. SR1 is R7 then [8:6]=111. Immediate part is 12=01100. Since there is immediate part in instruction [5]=1. Resulting machine code for this instruction is:

0010 011 111 1 01100. The instruction is 27ECh.

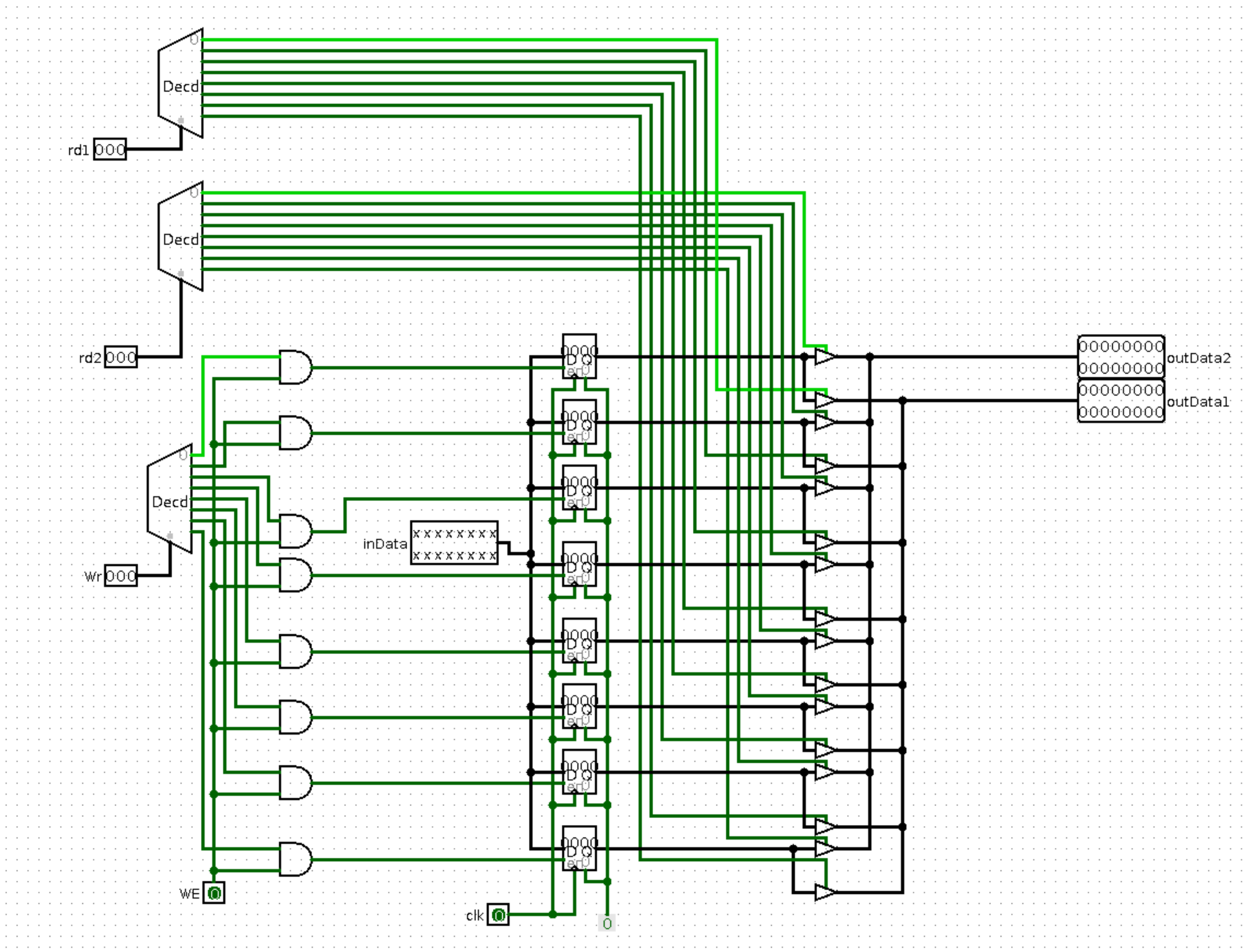
Lets do one of memory instructions *LD*.

LD R5,#3 ; opcode for LD instruction is 0100. *DR* is *R5* so [11:9]=101. Address9=000000011.

0100 101 000000011. The instruction is 4A03h.

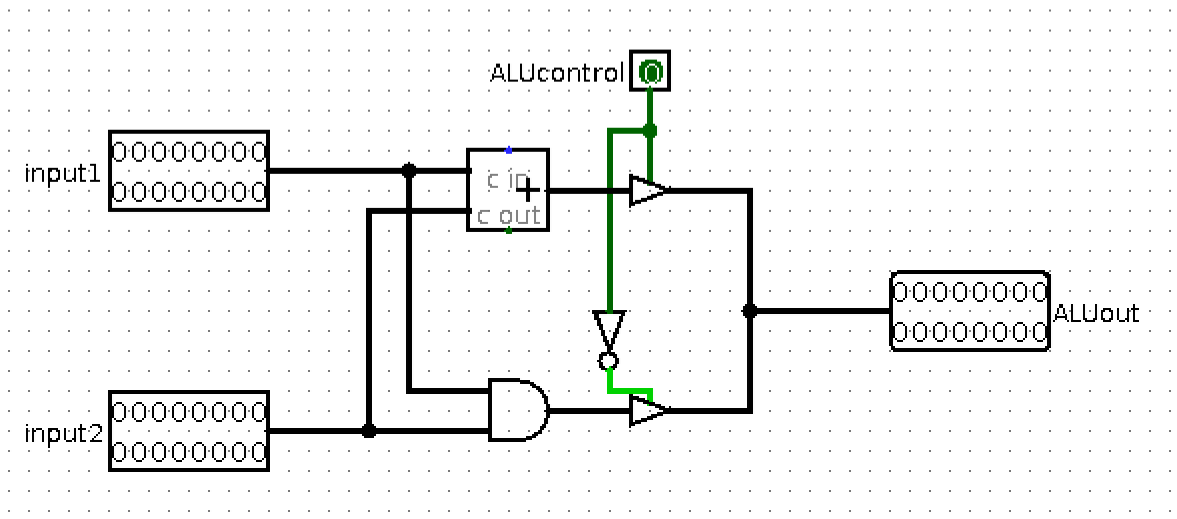
Due Date: 27.12.2013

Register File contains 8 registers. It has 4 input port *SR1*,*SR2*,*DR*,*inData* and 2 output port *OutData1* and *OutData2*. Its controlled by a signal *WE*(write enable). When *WE* is 1 , data at the *inData* port will be stored to *DR*. When *WE* is 0, all registers will preserve their values.



Due Date: 27.12.2013

ALU has 2 inputs *input1* and *input2* which is directed by the values read from *Register File* or immediate part of the instruction. It has one result as *ALUout*. *ALUcontrol* signal decides the operation to be done in *ALU*(*ADD* or *AND* in our case). If the *ALUcontrol* signal is 1 operation is *ADD* , If the *ALUcontrol* signal is 0 operation is *AND*.



Control Unit produces appropriate signals for all parts in datapath. Every operation of instructions are seperated according to their relation with sequential parts in the datapath. For example for *LD* instruction we will spend a clock cycle to read instruction adress from *PC*, 1 clock cycle to read instruction from the *Instruction Memory*, 1 clock cycle to read data from *Data Memory* and finally 1 clock cycle to write the data to *DR*. Load instruction will take 4 clock cycles. For *ADD* instruction again 2 clock cycle to *PC* read and *Instruction Memory* read, 1 clock cycle to store *ALU* result to *DR*. *ADD* instruction take 3 clock cycles. Notice that for every instruction *PC* read and *Instruction Memory* read will be same. Lets name these 2 state as *Fetch1* and *Fetch2* and lets produce signals for every state in the design.

Fetch1 PC read: *Instruction Memory*[read_adress]<=*PC*

Fetch2 InstructionRead: *InstructionMemory*[data_output]<=*InstructionMemory*[read_adress]

LD1 Mux1=X Mux2=01 MemRead=1

LD2 Mux3=0 Mux4=0 WE(Write Signal for RegFile)=1 Mux2=01
PCwrite=1 MemRead=1

Due Date: 27.12.2013

ADD1	Mux1=Instruction[5]	ALUcontrol=0	Mux3=1	Mux4=0	WE=1	PCwrite=1
AND1	Mux1=Instruction[5]	ALUcontrol=1	Mux3=1	Mux4=0	WE=1	PCwrite=1
ST1	Mux1=X	Mux2=01	MemWrite(Store signal to Data Memory)=1	PCwrite=1		
JMP1	Mux1=X	Mux2=11	Mux3=1	Mux5=0	PCwrite=1	

According to those signals lets produce inputs for components(select bits for muxes).

Mux1[0]=Instruction[5]

Mux2[0]= LD1 | ST1 | JMP1 | LD2

Mux2[1]= JMP1

Mux3[0]= ADD1 | AND1 | JMP1

Mux4[0]= 0

Mux5[0]= (JMP1)'

ALUcontrol= AND1

WE= ADD1 | AND1 | LD2

MemWrite= ST1

PCwrite= LD2 | JMP1 | ADD1 | AND1 | ST1

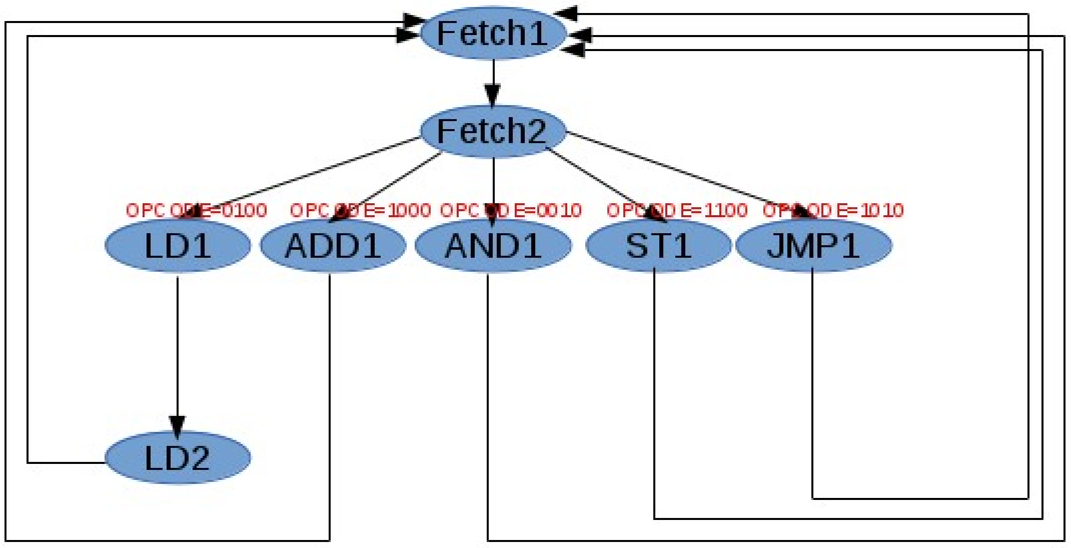
MemRead= LD1 | LD2

InstrRead=1

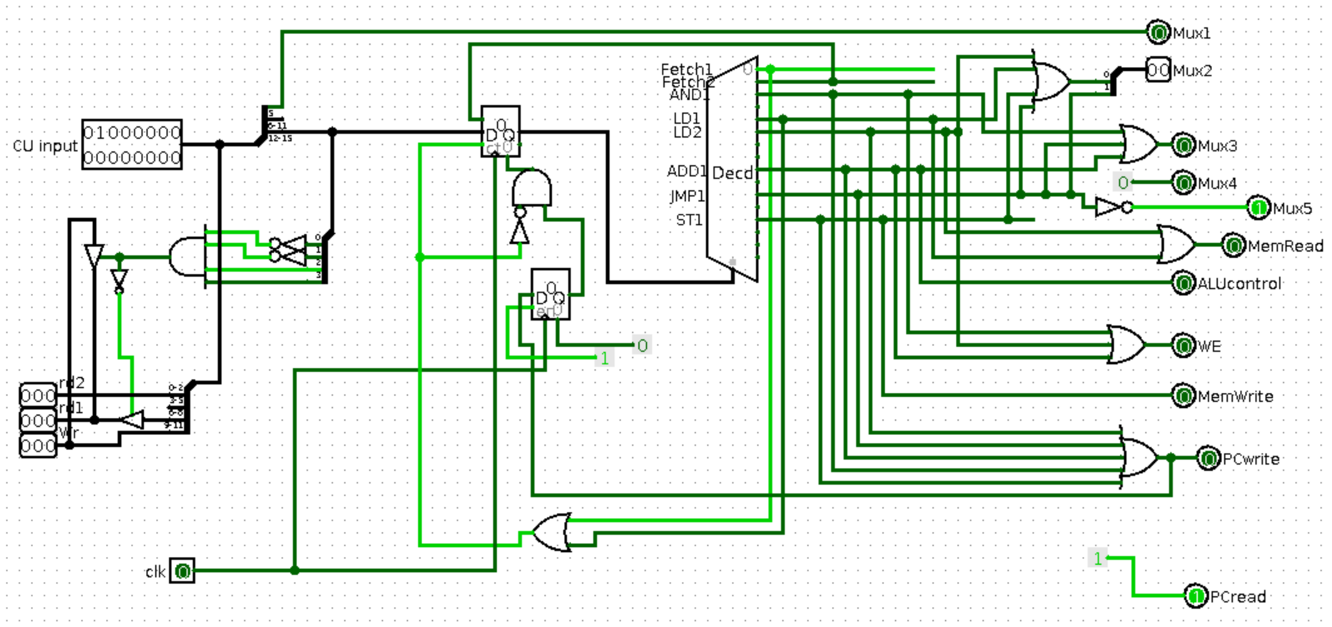
PCRead=1

Due Date: 27.12.2013

Control Unit state diagram as follows.



Control Unit is implemented by a counter and decoder. Decoder outputs represent states for Instructions.



Due Date: 27.12.2013

What to Submit?

You are required to implement above design in Logisim(25pts) and Verilog(50pts). You are also required to write an assembler(25pts). Assembler input will be a code sequence of assembly language defined above. Output of the assembler will be machine code in two different formats. One is for your logisim design and the other is for your Verilog design. Lets see how it should look like in two different files. Given example code below:

LD R0,#0

LD R1,#2

ADD R2,R1,R0

ADD R3,R2,R3

ST R3,#10

ADD R3,R3,#12

AND R4,R0,R1

AND R0,R1,#3

JMP #0

Generated machine code should be:

4000

4202

8440

8683

C60A

86EC

2801

2063

A000

Due Date: 27.12.2013

Logisim memory file with above machine code looks like:

v2.0 raw

4000 4202 8440 8683 c60a 86ec 2801 2063

a000

Verilog memory file doesn't require any header like in logisim(v2.0 raw). It will be in `.*hex*` file format and content will be same as the generated machine code. Verilog code to read memory from a .hex file:

```
module  memory();
 reg [15:0] InstructionMemory [15:0];

      initial begin
            $readmemh("AssemblerOutput.hex", InstructionMemory);
      end
endmodule
```

Data Memory can be read the same way for both Verilog and Logisim design. Data will be entered to *Data Memory* files manually.

$$BONUS$$

Make necessary upgrades to your datapath so that it supports some more instruction from LC-3 instruction set. *STI* (Store Indirect),*LDI* (Load Indirect) and *BR* (Branch) instructions. You will get 10 pts bonus for each instruction.

	[15:12](opcode)	[11:9]			[8:0]
STI	1011	SR			Address9
LDI	1000	DR			Address9
BR	0010	N	Z	P	Address9

LDI DR,Address9   *; DR=DataMemory[DataMemory[ZeroExtend[Address9]]]*

STI SR,Address9   *; DataMemory[DataMemory[ZeroExtend[Address9]]]=SR*

Due Date: 27.12.2013

BRn Address9	*;Branch to DataMemory[Address9] if the last result was negative*
BRz Address9	*;Branch to DataMemory[Address9] if the last result was zero*
BRp Address9	*;Branch to DataMemory[Address9] if the last result was positive*

Submission

You can submit the project as group of at most three students. You should submit your all work under file name `` `*StudentID1_StudentID2_StudentID3*.zip` `` and send an e-mail to the address: lokman.altin@gmail.com before the midnight of 27th December.