CSE3015 Digital Logic Design Project Report **17.01.2021**

Ozan Yerli 150118064

Aykut Kuşkaya 150117854

Zeynep Ferah Akkurt 150119824

Lara Gülsüm Şen 150118071

General Info

In this project, we are supposed to design a basic CPU. Processor will have 16 bits address width and 16 bits data width with 8 registers and supports instructions ADD, AND, ADDI, ANDI, LD, ST, CMP, JUMP, JE, JA, JB, JAE, JBE.

Briefly definition of instructions:

- ADD: this instruction will add two register, and store result into another register
- ADDI: this instruction will add a register value and immediate value, and store the result into another register
- AND: this instruction will and two register, and store result into another register
- ANDI: this instruction will and a register value and immediate value, and store the result into another register
- LD: this instruction will load a value from Data Memory to any register.
- ST: this instruction will store value from a register to Data Memory.
- CMP: Compare instruction will compare two operands, then will update the flag values of zero flag. Example: it will compare registers OP1 and OP2 if they are equal, flag values will be ZF=1, CF=0, if the first operands value is greater flag values will be ZF=0, CF=0, if the first operands value is less than the second operand flag values will be ZF=0,CF=1.
- JUMP: this instruction will set the Program Counter (PC will hold current instruction's address) to the given value in the instruction.
- As in the JUMP instruction, but conditional with the flag values, JE, JA, JB, JAE, JBE jumps to a new destination address.
  JE ADDR: If flag values are (ZF=1, CF=0), PC will be set to ADDR(PC-relative).
  JA ADDR: if flag values are (ZF=0, CF=0), PC will be set to ADDR(PC-relative).
  JB ADDR: if flag values are (ZF=0, CF=1), PC will be set to ADDR(PC-relative).
  JAE ADDR: if flag values are (CF=0), PC will be set to ADDR(PC-relative).
  JBE ADDR: if flag values are (CF=1 or ZF=1), PC will be set to ADDR(PC-relative).

Part 1: Assembler

In this part, first, ISA should be designed, then a code needed as an assembler according to ISA table for converting these instructions to the machine code that our CPU will understand. In this project, java programming language is used and the file for this purpose is given as name "assembler.java". There will be one output file contains the codes of instructions.

Design ISA:

Since there is 8 register, 3 bits is enough and for 13 instructions,4 bits is enough. This way for each register we use 3 bit and for opcode 4 bits is used. This leaves 6 bits for immediate value (IMM) and 12 bits for address.

Instruction Set Architecture:

INSTRUCTION	OPCODE [15:12]	[11:9]	[8:6]	[5:3]	[2:0]
AND	0000	DR	SR1	000	SR2
ANDI	0001	DR	SR1	IMM	
ADD	0010	DR	SR1	000	SR2
ADDI	0011	DR	SR1	IMM	
LD	0100	DR	ADDRESS		
ST	0101	SR	ADDRESS		
CMP	0110	000	OP1	000	OP2
JUMP	1000	ADDRESS			
JE	1001	ADDRESS			
JA	1010	ADDRESS			
JB	1011	ADDRESS			
JBE	1100	ADDRESS			
JAE	1101	ADDRESS			

Part 2: Logisim Component Design

- Adder:
  Full adder is designed, and then 16-bit adder is designed by using 16 full-adder.
- CMP:
  16-bit comparator is designed by using half and full comparator which are also designed by us. And using this comparator, there is CMP circuit which takes four inputs. Two of them is the 16bit input which are came from by reading the values inside 2 registers from the register file (output1 and output2) and compares those 2 register's values then depending on this result it sets the ZF and CF flag values. This flag values will be used in jump condition later.
  The other inputs are the clock and CMP signal. If the opcode in the instruction is match with the comparator operation then this signal will send(from control unit) and of course at each clock rising edge, the operation will work.

- ALU:

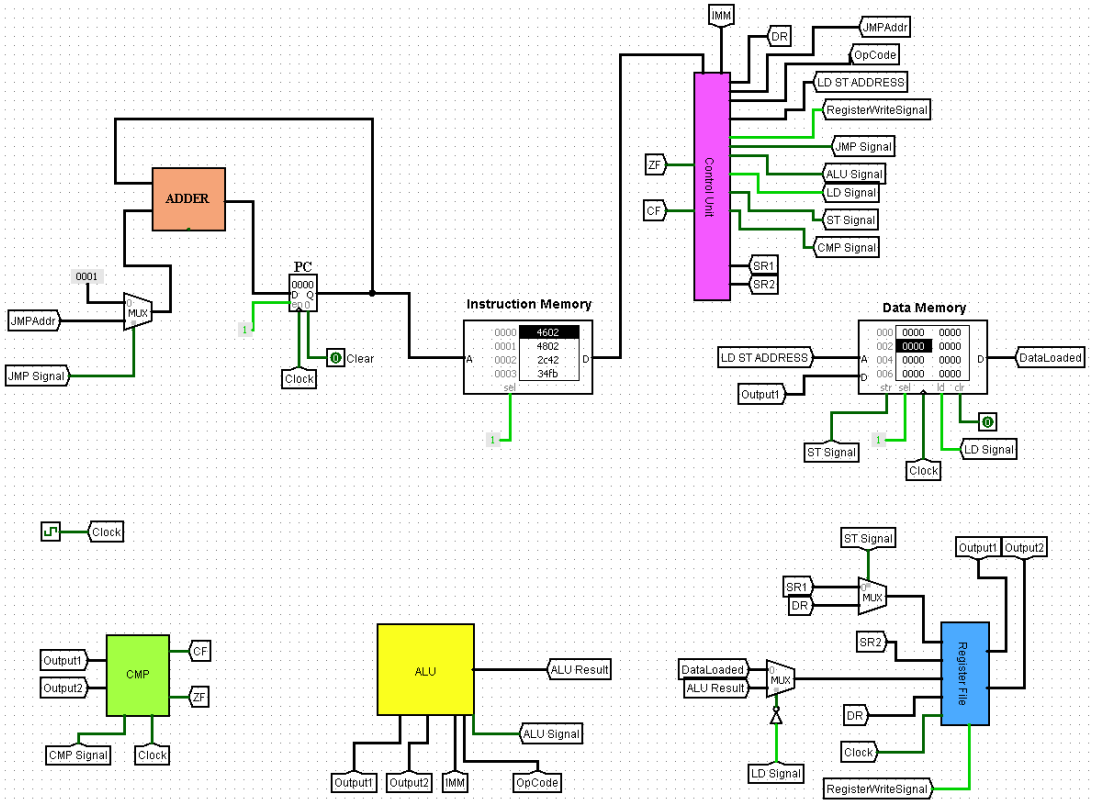
  Arithmetic logic unit gets five inputs. Three of these inputs are the values inside of register 1, register 2 and the immediate value. Fourth one is ALU Signal. The signals are used for identifying every instruction. If the current instruction is add or and operation then the ALU signal is given and the operation will be performed. The last input is the Opcode of the instruction. The Opcode is used for identifying. If the first

bit is ‘0’, this indicates that the second input is register 2 If the first bit is ‘1’, this indicates that the second input is an immediate value. The second bit of the Opcode is for identifying the instruction. If ‘0’ than the result is the result of and operation, if ‘1’ than the result is the result of add operation. All possible operations are done inside ALU. The result is chosen by using multiplexer with the information from opcode.

- RegisterFile:

  Register file has 6 inputs Source1, Source2 to read, a destination register(DR) for write, RegisterWriteSignal, clock, input data which is decided from a mux depending on LD signal.
  Source1 and Source2 are selectors for the registers which we want to read.
  Destination register (DR) is selector for register which we want to write data in it.
  When RegisterWriteSignal is 1, Input data is stored to Destination register. When RegisterWriteSignal is 0, registers preserver their values.
  There are 8 registers have 16-bit data-width.
  1- Input data : 16 bit data input
  2- Clock: shared clock for all system
  3- DR: 3 bit input to select register which to be loaded.
  4 Source1 and Source2: 3 bit input to select registers which to be readed.
  5- RegisterWriteSignal: Signal to enable write operation on registers.

- PC: Program counter is implemented with the built-in register of Logisim. In each clock cycle, if there is no JMP signal from the control unit, the program counter is incremented by one. If there is a JMP signal, the program counter is set to a given PC-relative value.



Part 3: Logisim Control Unit

In the control unit, we first separated instruction in order to get the information needed. This information is provided by the ISA.

[2:0]: the address of second register

[0:5]: immediate value. Since our registers are 16 bits. We sign extended immediate value and stored it.

[8:6]: the address of first register

[8:0]: The address for LD and ST instructions. Since our registers are 16 bits. We extended the address and stored it.

[11:0]: The address for jump instructions. Since our registers are 16 bits. We extended the address and stored it.

[11:9]: the address of directory

[15:12]: Opcode

After separation, we used decoder to identify instructions which are given by Opcode. Then used them to create signals.

Signals:

1) **Register Write Signal:** The instructions (AND, ANDI, ADD, ADDI, LD) writes the results into a register. This is because we created a signal.
2) **ALU Signal:** The instructions (AND, ANDI, ADD, ADDI) require arithmetic operations. If the instruction is one of them than the Alu signal is given and the operation is done inside ALU.
3) **LD Signal:** If the instruction is LD than the signal is given.
4) **ST Signal:** If the instruction is ST than the signal is given.
5) **CMP Signal:** If the instruction is CMP than the signal is given.
6) **JMP Signal:** If the instruction is one of the jump instructions (JUMP, JE, JA, JB, JAE, JBE) and the conditions are true than the signal is given.

Control unit gets two inputs, zero and carry flag, in order to control whether or not the jump conditions are true.

CPU:

There are several components are used in our main CPU design. There are signals come from Control Unit, but we also need component to extract information from the instruction memory.

Data Memory:

- LD and ST instructions use data memory and registers. LD loads data to register from memory, ST stores data to data memory from register. For both, we need a Memory Address that comes with instruction. Memory waits two signals from control unit for ST and LD instruction. For ST instruction Control Unit signal sets the ST signal of data memory, for LD instruction Control Unit signal sets the LD signal of data memory.
  There is a mux on the out of the register file. This mux decides which source (value inside register) of the data to be written to register. If the instruction is an ALU operation then the source will be the output came from ALU result, if the instruction is LD, the source will be the output came from data memory (DataLoaded). This is controlled by LD signal.

Instruction Memory:

- The instruction memory is implemented with the built-in ROM component of the Logisim. It reads the address given by the PC and outputs the data in that address. The data is sent to the control unit to produce signals for the instruction.