Part 1. Preliminary Work / Preliminary Design Report (50 points)

CS224
Section No.: 05
Spring 2018
Lab No.: 04
Asuman Aydın/21502604

1. **[5 pts]** What does it mean for a processor to be a *single-cycle*? What are the differences between single-cycle, multi-cycle, and pipelined architectures? Briefly explain.

⇨ It means for a processor to be a *single-cycle that* processor executes all instructions in one cycle. It has simple structure than other architectures. One of the differences is regardless of what is the instruction; single cycle does every instruction in the same amount of time. For example, load instructions may be longer than arithmetic instructions but they are all executed in the same amount of time. Thus, pipelined architecture is composed of the methodology of multi-cycle and single-cycle. While using the multi-cycle method, it decreases the amount of time which single-cycle wastes. When all processors compared, they do the same thing at different times. From slower to faster: single-cycle, multi-cycle, pipelined.

2. **[10 pts]** In this lab, we will work on a MIPS architecture that executes instructions in a single cycle. A single-cycle processor consists of different units that can be divided into two main parts: the datapath and the control unit. *List* the modules that constitute each of these parts. *Explain* the functions of each module with your own words [Do not copy/paste the definitions in the textbook. You may omit the adders and multiplexers in the data path for this part, and focus on the main units].

⇨ **Datapath**:
1. **Instruction Memory**: It contains the instruction address
2. **Register File**: It includes the rs, rt, and destination register and has control of registers with enable of the RegWrite control signal. It helps to read the base address and helps signExtension to decide if the instruction is positive or negative to execute.
3. **Sign Extend**: It determines the sign of instruction whether positive or negative according to a most significant bit of instr15:0. It takes the value and sends it to ALU.
4. **ALU**: It takes the signExtension result and rs register value from RD1 and add them to compute the memory address.
5. **Data Memory**: It takes the data that ALU computed and reads in to memory and transfers it to destination register head back to register file.

⇨ **The control Unit**:
1. **Main Decoder**: It has the control of clock cycle and execution of specific instructions. It makes it happen in proper way. The modules that it has every functional to control the instruction's execution. It also controls keeping the values in memory and makes connection between processor and registers.

- Opcode
- MemToReg
- MemWrite
- Branch
- ALUSrc
- RegDst
- RegWrite

2. **ALU Decoder**: It determines which function ALU will execute so it produces the signal for it. It looks at function and opcode part to specify the signal. For example, for and function ALU signal would be 000.
    - Funct
    - ALUControl

3. **[15 pts]** Looking at the complete processor (see the textbook, p.383) might be intimidating at first. That is why it is important that you isolate each module and analyze them individually.
What are the inputs and outputs of each module? Write a descriptive signature for each module that contains its inputs and outputs, as well as how many bits each input/output has.

For example:

```
ExampleModule(input a, input[32] b, output[32] c)
```

means that the ExampleModule takes two inputs: a and b, consisting of 1 and 32 bits respectively; and an output c, which has 32 bits. [Accomplishing this is the key to be able to write SystemVerilog modules. Whenever you want to write a SystemVerilog module, treat it as a blackbox and ask the following questions: if this module was doing what it is supposed to be doing, (1) what would be its inputs and outputs, (2) how many bits each of them would have?

- ⇨ InstructionMemory(input logic[31:0] a, output logic[31:0] rd)
- ⇨ Pc(input logic[31:0] pcpre, output logic[31:0] pc)
- ⇨ RegisterFile(input logic[4:0] a1, a2, a3, input logic[31:0] wd3, input logic clk, input logic we3, output logic[31:0] rd1, rd2)
- ⇨ SignExtend(input logic[15:0] inst, output logic[31:0] signImm)
- ⇨ DataMemory(input logic[31:0] a, wd, input clk, input we, output logic[31:0] rd)
- ⇨ PCPlus4(input logic[31:0] pc, input logic[3:0] four, output logic[31:0] pcplus4)
- ⇨ ALu(input logic[31:0] srcA, srcB, output logic zero, output logic[31:0] ALUResult)
- ⇨ PcBranch(input logic signImm, input logic[31:0] pcplus4, output logic[31:0] pcBranch)
- ⇨ MuxPc(input logic PCSrc, input logic[31:0] pcBranch, input logic[31:0] pcplus4, output logic logic[31:0] pcpre)
- ⇨ muxWriteReg(input logic RegDst, input logic[4:0] a2, a3, output logic[4:0] a3final)
- ⇨ muxRd2(input logic[31:0] rd2, signImm, output logic[31:0] srcB)

⇨ muxReadData(input logic memToReg, input logic[31:0] ALUResult, readData, output logic[31:0] result)
⇨ mainDecoder(input logic[5:0] opcode, output logic, memtoreg ,memwrite ,branch ,ALUsrc , regdst, regwrite, output logic[1:0] ALUop)
⇨ ALUDecoder(input logic[5:0] Funct, input logic[1:0] ALUop, output logic[2:0] ALUcontrol)

4. **[20 pts]** Determine the assembly language equivalent of the machine codes given in the imem module in the "imem.txt" file posted on Unilica for this lab. In the given SystemVerilog module for imem, the hex values are the MIPS machine language instructions for a small test program. Dis-assemble these codes into the equivalent assembly language instructions and give a 3-column table for the program, with one line per instruction, containing its location, machine instruction (in hex) and its assembly language equivalent. [Note: you may dis-assemble by hand or use a program for this purpose]

INSTRUCTION	**ADDRESS**	**MACHINE INST. IN HEX**
addi $v0, $zero, 5	8'h00	0x20020005
addi $v1, $zero, 12	8'h04	0x2003000c
addi $a3, $v1, 65527	8'h08	0x2067fff7
or $a0, $a3, $v0	8'h0c	0x00e22025
and $a1, $v1, $v0	8'h10	0x00642824
add $a1, $a1, $a0	8'h14	0x00a42820
beq $a1, $a3,10	8'h18	0x10a7000a
slt $a0, $v1, $a0	8'h1c	0x0064202a
beq $a0, $zero, 1	8'h20	0x10800001
addi $a1, $zero, 0	8'h24	0x20050000
slt $a0, $a3, $v0	8'h28	0x00e2202a
add $a3, $a0, $a1	8'h2c	0x00853820
sub $a3, $a3, $v0	8'h30	0x00e23822
sw $a3, 68, $v1	8'h34	0xac670044
lw $v0, 80, $zero	8'h38	0x8c020050
J 0x0000011	8'h3c	0x08000011
addi $v0, $zero, 1	8'h40	0x20020001
sw $v0, 84, $zero	8'h44	0xac020054
J 0x0000012	8'h48	0x08000012
END		