# Computer Organization, Spring 2020

## Lab 4: Single-Cycle CPU

**Due: 2020/05/21**

## 1. Goal

Based on Lab 3(Simple Single-Cycle CPU), add a memory unit(Data Memory) to implement a Complete Single-Cycle CPU.

## 2. HW Requirement

(1) Please use ModelSim/ISE as you HDL simulator.
(2) Please attach your names and student IDs as comment at the top of each file.
(3) Refer to Lab 3 top module's name and I/O ports.
You may need to add control signals below to Decoder:
- Jump_o
- MemRead_o
- MemWrite_o
- MemtoReg_o

(4) Basic instruction set (60%)

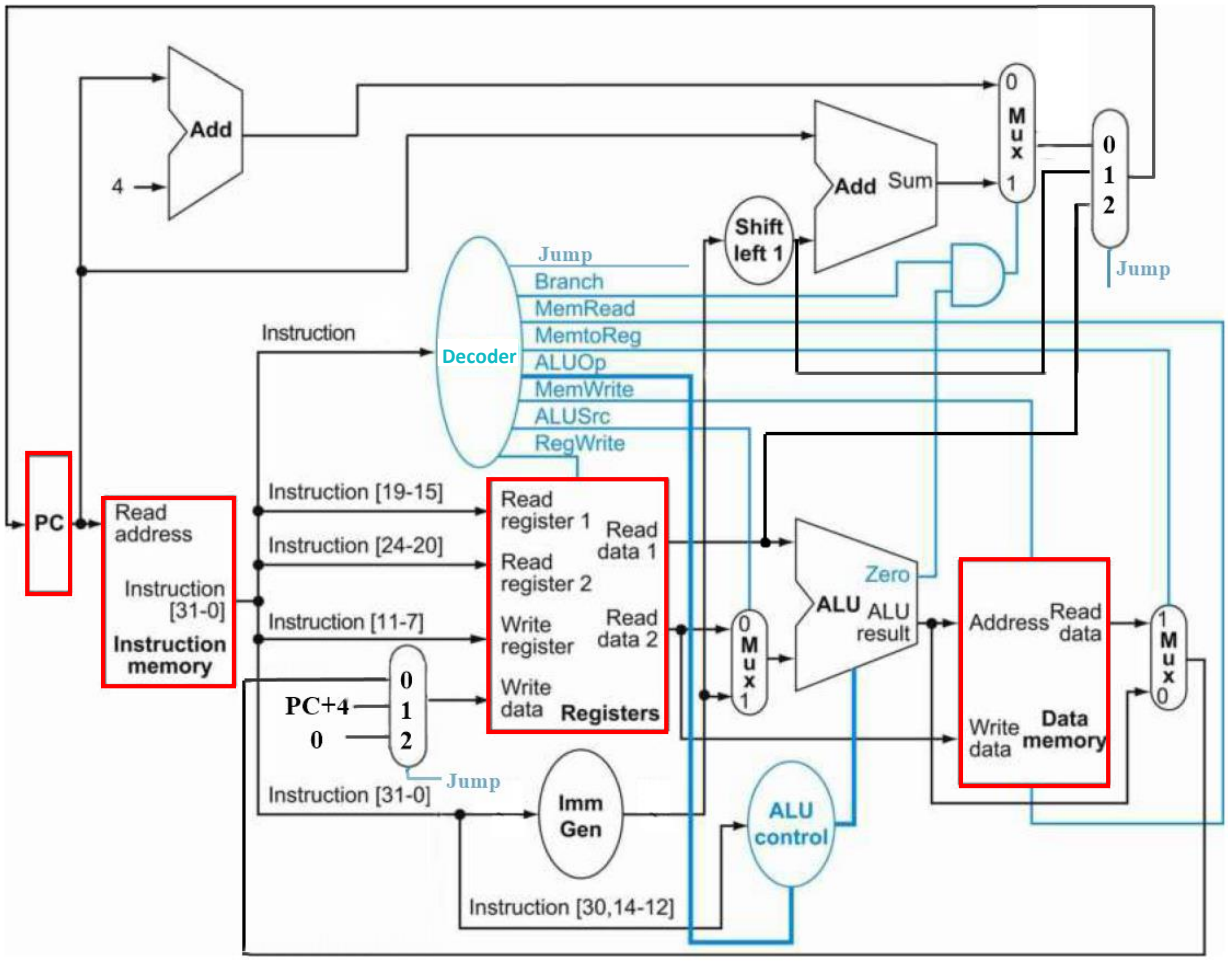
| Instruction | Example | Meaning | Opcode | Funct3 | Funct7 |
|---|---|---|---|---|---|
| Load word | lw r1, 0(r2) | Load r1 from address 0+r2 | 0000011 | 010 | - |
| Store word | sw r1, 0(r2) | Store r1 at address 0+r2 | 0100011 | 010 | - |
| Branch on less than | blt r1, r2, 4 | if(r1 < r2)<br>PC += 4 | 1100011 | 100 | - |
| Branch on greater than | bge r1, r2, 4 | if(r1 >= r2)<br>PC += 4 | 1100011 | 101 | - |
| Jump and link | jal r1, 20 | r1 = PC + 4<br>PC = 20 | 1101111 | - | - |

(5) Advanced set(20%)

| Instruction | Example | Meaning | Opcode | Funct3 | Funct7 |
|---|---|---|---|---|---|
| Exclusive or immediate | xori r1, r2, 10 | r1 = r2 ⊕ 10 | 0010011 | 100 | - |
| Jump and link register | jalr r0, 0(r1) | //Return from subroutine<br>PC = r1 | 1100111 | 000 | - |

(6) Report(20%)

# 3. Architecture Diagram



# 4. Testbench

CO_test_data1.txt tests the basic instructions (60 points).
CO_test_data2.txt tests the advance instructions (20 points).
The default pattern is the Test data 1.
Please edit the line 19 in the file “Instr_Memory.v” to test the other cases.
Line 19: $readmemb("CO_test_data1.txt", instruction_file);

```
16  initial begin
17      for ( i=0; i<32; i=i+1 )
18          instruction_file[i] = 32'b0;
19      $readmemb("CO_test_data1.txt", instruction_file);
20  end
```

The following are the assembly code for the test pattern:

| Test data 1 | Test data 2 |
|---|---|
| addi r5, r0, 1 | addi r5, r0, 1 |
| addi r6, r0, 2 | xori r6, r5, 6 |
| addi r7, r0, 3 | jal r1, 16 |
| jal r1, 24 | jal r1, 32 |
| bge r7, r5, 44 | add r7, r7, r6 |
| addi r6, r6, 31 | addi r6, r6, -1 |
| addi r2, r2, -12 | bge r6, r5, -8 |
| sw r5, 0(r2) | jalr r0, r1, 0 |
| sw r6, 4(r2) | addi r2, r2, -4 |
| sw r7, 8(r2) | sw r7, 0(r2) |
| addi r5, r5, 3 | addi r2, r2, 4 |
| addi r6, r6, 3 | |
| addi r7, r7, 3 | |
| blt r6, r7, -36 | |
| addi r5, r5, 31 | |
| lw r8, 0(r2) | |
| lw r9, 4(r2) | |
| lw r10, 8(r2) | |
| addi r2, r2, 12 | |

Final result for Test data 1:

```
# PC =         76
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          1,          2,          3
# Registers
# R0 =          0, R1 =         16, R2 =        128, R3 =          0, R4 =          0, R5 =          4, R6 =          5, R7 =          6
# R8 =          1, R9 =          2, R10 =          3, R11 =          0, R12 =          0, R13 =          0, R14 =          0, R15 =          0
# R16 =          0, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =          0, R22 =          0, R23 =          0
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =          0, R30 =          0, R31 =          0
```

Final result for Test data 2:

```
# PC =         44
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,          0
# Data Memory =          0,          0,          0,          0,          0,          0,          0,         28
# Registers
# R0 =          0, R1 =         16, R2 =        128, R3 =          0, R4 =          0, R5 =          1, R6 =          0, R7 =         28
# R8 =          0, R9 =          0, R10 =          0, R11 =          0, R12 =          0, R13 =          0, R14 =          0, R15 =          0
# R16 =          0, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =          0, R22 =          0, R23 =          0
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =          0, R30 =          0, R31 =          0
```

## 5. Grade

(1) Basic instructions score: 60 points.
(2) Advance instructions score: 20 points.
(3) Report: 20 points – format is in CO_Report.docx.
(4) Late submission: 10 percent penalty per day
(5) No plagiarism, or you will get 0 point.

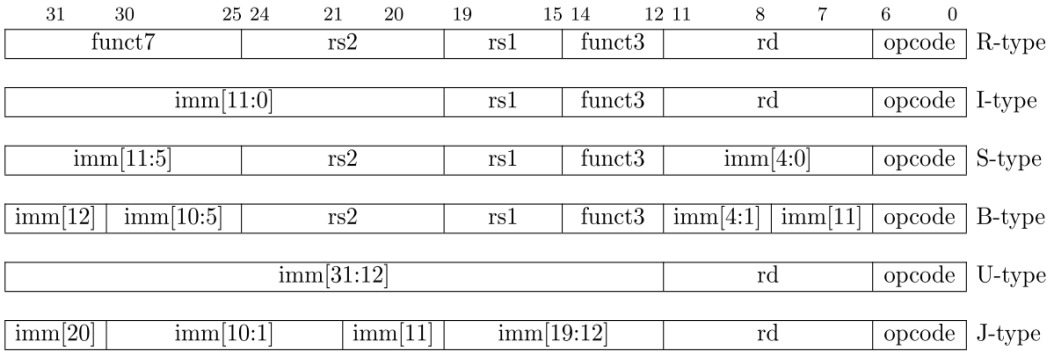
## 6. Hand in

(1) Zip your folder and name it as "GID_ID1_ID2.zip" (e.g. G1_0816001_0816002.zip) before uploading to newe3. Other filenames and formats such as *.rar and *.7z are NOT accepted! Multiple submissions are accepted, and the version with the latest time stamp will be graded.
(2) Please include ONLY Verilog source codes (*.v) and your report (*.docx or *.pdf) in the zipped folder.

## 7. Q&A

For any questions regarding Lab 4, please contact
張祐銘 yumingchang.cs03@g2.nctu.edu.tw
賴柏宏 bhbruce.cs07g@nctu.edu.tw
鄭俊賢 petertay1996.cs08g@nctu.edu.tw

# 8. References




| Instruction opcode | ALUOp | Operation | Funct7 field | Funct3 field | Desired ALU action | ALU control input |
|---|---|---|---|---|---|---|
| ld | 00 | load doubleword | XXXXXXX | XXX | add | 0010 |
| sd | 00 | store doubleword | XXXXXXX | XXX | add | 0010 |
| beq | 01 | branch if equal | XXXXXXX | XXX | subtract | 0110 |
| R-type | 10 | add | 0000000 | 000 | add | 0010 |
| R-type | 10 | sub | 0100000 | 000 | subtract | 0110 |
| R-type | 10 | and | 0000000 | 111 | AND | 0000 |
| R-type | 10 | or | 0000000 | 110 | OR | 0001 |

| Register | ABI Name | Description | Saver |
|---|---|---|---|
| r0 | zero | Hard-wired zero | — |
| r1 | ra | Return address | Caller |
| r2 | sp | Stack pointer | Callee |
| r3 | gp | Global pointer | — |
| r4 | tp | Thread pointer | |
| r5 | t0 | Temporary/alternate link register | Caller |
| r6-7 | t1–2 | Temporaries | Caller |
| r8 | s0/fp | Saved register/frame pointer | Callee |
| r9 | s1 | Saved register | Callee |
| r10-11 | a0–1 | Function arguments/return values | Caller |
| r12-17 | a2–7 | Function arguments | Caller |
| r18-27 | s2–11 | Saved registers | Callee |
| r28-31 | t3–6 | Temporaries | Caller |

| ALUOp | | Funct7 field | | | | | | | Funct3 field | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ALUOp1 | ALUOp0 | I[31] | I[30] | I[29] | I[28] | I[27] | I[26] | I[25] | I[14] | I[13] | I[12] | Operation |
| 0 | 0 | X | X | X | X | X | X | X | X | X | X | 0010 |
| X | 1 | X | X | X | X | X | X | X | X | X | X | 0110 |
| 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0010 |
| 1 | X | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0110 |
| 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0000 |
| 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0001 |

**FIGURE 4.13 The truth table for the 4 ALU control bits (called Operation).**

| Instruction | ALUSrc | Memto-Reg | Reg-Write | Mem-Read | Mem-Write | Branch | ALUOp1 | ALUOp0 |
|---|---|---|---|---|---|---|---|---|
| R-format | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| ld | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| sd | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 |
| beq | 0 | X | 0 | 0 | 0 | 1 | 0 | 1 |

**FIGURE 4.18 The setting of the control lines is completely determined by the opcode fields of the instruction.**