Computer Architecture
Experiment

Lab2
Topics

- 0. Lab introduction
- Lab 1). Warmup Run your multiple-cycle CPU on 3E board. Try to add one new branch instruction.
- Lab 2). 5-stage pipelined CPU with 15 MIPS instructions (only required to execute in pipeline).
- Lab 3). Implementing "stall" when have hazards
- Lab 4). Implementing “forwarding paths”
- Lab 5). The whole CPU with 31 instructions.
Outline

- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Precaution
Experiment Purpose 1

- Understand the principles of MC CPU Controller and datapath and master methods of MC CPU Controller and datapath design.
- Understand the principles of Datapath and master methods of Datapath design
- Understand the principles of MC CPU and master methods of MC CPU design
- Master methods of program verification of CPU
Experiment Purpose 2

- Understand the principles of Pipelined CPU
- Understand the basic units of Pipelined CPU
- Understand the working flow of 5-stages
- Master the method of simple Pipelined CPU
- Master methods of program verification of simple Pipelined CPU
Experiment Task 1

- Design the CPU Controller, Datapath, bring together the basic units into Multiple-cycle CPU
- Verify the MC CPU with program and observe the execution of program
Experiment Task 2

- Design the CPU Controller, and the Datapath of 5-stages Pipelined CPU
  - 5 Stages
  - Register File
  - Memory (Instruction and Data)
  - other basic units

- Verify the Pipelined CPU with program and observe the execution of program
### 15 common used MIPS instructions

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>31..26</td>
<td></td>
</tr>
<tr>
<td>25..21</td>
<td></td>
</tr>
<tr>
<td>20..16</td>
<td></td>
</tr>
<tr>
<td>15..11</td>
<td></td>
</tr>
<tr>
<td>10..06</td>
<td></td>
</tr>
<tr>
<td>05..00</td>
<td></td>
</tr>
</tbody>
</table>

#### R-type

<table>
<thead>
<tr>
<th>cp</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>func</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>000000</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>000000</td>
</tr>
<tr>
<td>sub</td>
<td>000000</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>000000</td>
</tr>
<tr>
<td>and</td>
<td>000000</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>000000</td>
</tr>
<tr>
<td>or</td>
<td>000000</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>000000</td>
</tr>
<tr>
<td>sll</td>
<td>000000</td>
<td>00000</td>
<td>rt</td>
<td>rd</td>
<td>sa</td>
</tr>
<tr>
<td>srl</td>
<td>000000</td>
<td>00000</td>
<td>rt</td>
<td>rd</td>
<td>sa</td>
</tr>
<tr>
<td>sra</td>
<td>000000</td>
<td>00000</td>
<td>rt</td>
<td>rd</td>
<td>sa</td>
</tr>
</tbody>
</table>

#### I-type

<table>
<thead>
<tr>
<th>cp</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>addi</td>
<td>001000</td>
<td>rs</td>
<td>rt</td>
</tr>
<tr>
<td>andi</td>
<td>001100</td>
<td>rs</td>
<td>rt</td>
</tr>
<tr>
<td>ori</td>
<td>001101</td>
<td>rs</td>
<td>rt</td>
</tr>
<tr>
<td>lw</td>
<td>100011</td>
<td>rs</td>
<td>rt</td>
</tr>
<tr>
<td>sw</td>
<td>101011</td>
<td>rs</td>
<td>rt</td>
</tr>
</tbody>
</table>

#### J-type

<table>
<thead>
<tr>
<th>cp</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>000010</td>
</tr>
</tbody>
</table>
Step1: CPU Controller
## Output of CPU Controller

<table>
<thead>
<tr>
<th>Output Signal</th>
<th>Meaning When 1</th>
<th>Meaning When 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 PCSrc[1:0]</td>
<td>00: PC + 4; 01: Branch Instr.; 10: jump Instr</td>
<td></td>
</tr>
<tr>
<td>2 WritePC</td>
<td>Write PC</td>
<td>Not Write PC</td>
</tr>
<tr>
<td>3 IorD</td>
<td>Instruction Addr</td>
<td>Data Addr.</td>
</tr>
<tr>
<td>4 WriteMem</td>
<td>Write Mem.</td>
<td>Not Write Mem.</td>
</tr>
<tr>
<td>5 Write DR</td>
<td>Write Data. Reg</td>
<td>Not Write Data. Reg</td>
</tr>
<tr>
<td>6 Write IR</td>
<td>Write Instr. Reg</td>
<td>Not Write Instr. Reg</td>
</tr>
<tr>
<td>7 MemToReg</td>
<td>From Mem. To Reg</td>
<td>From ALUOut To Reg</td>
</tr>
<tr>
<td>8 RegDest</td>
<td>rd</td>
<td>rt</td>
</tr>
<tr>
<td>9 ALUC</td>
<td>ALU Controller Op</td>
<td></td>
</tr>
<tr>
<td>10 ALUSrcA</td>
<td>Register rs</td>
<td>PC</td>
</tr>
<tr>
<td>11 ALUSrcB</td>
<td>Selection: 00: Reg rt; 01: 4; 10: Imm.; 11: branch Address</td>
<td></td>
</tr>
<tr>
<td>12 WriteA</td>
<td>Write A Reg.</td>
<td>Not Write A Reg.</td>
</tr>
<tr>
<td>13 WriteB</td>
<td>Write B Reg.</td>
<td>Not Write B Reg.</td>
</tr>
<tr>
<td>14 WriteC</td>
<td>Write C Reg.</td>
<td>Not Write C Reg.</td>
</tr>
<tr>
<td>15 WriteReg</td>
<td>Write Reg.</td>
<td>Not Write Reg.</td>
</tr>
</tbody>
</table>
The principle of CPU Controller (1)

Stages of Multiple-Cycle Execution of Typical MIPS CPU
The principle of CPU Controller(2)
The Datapath of Multiple-cycle CPU
Basic Units of Multiple-cycle CPU

- CPU Controller
- ALU and ALU Controller
- Register file
- Mem. (Instruction and Data together).
- others: Register, sign-extend Unit, shifter, multiplexor
Memory

- Dual Port Block Memory
- Port A: Read Only, Width: 32, Depth: 512
- Port B: Read and Write, Read After Write
- Rising Edge Triggered
Multiple-cycle CPU Top Module

- memory
  \[ x\text{-}\text{memory}(\text{.addra(raddr)}, \text{.addrb(waddr)}, \text{.clka(clk)}, \text{.clkb(clk)}, \text{.dinb(b_data)}, \text{.douta(mem_data)}, \text{.web(write_mem)}) ];

- ctrl
  \[ x\text{-}\text{ctrl}(\text{clk}, \text{rst}, \text{ir_data}, \text{zero}, \text{write_pc}, \text{iord, write_mem}, \text{write_dr}, \text{write_ir}, \text{memtoreg}, \text{regdst, pcsourse, write_c, alu_ctrl, alu_srcA, alu_srcB, write_a, write_b, write_reg, state_out, insn_type, insn_code, insn_stage}); \]

- pcm
  \[ x\text{-}\text{pcm}(\text{clk}, \text{rst}, \text{alu\_out}, \text{c\_data}, \text{ir\_data}, \text{pcsource, write_pc,pc}); \]

- alu_wrapper
  \[ x\text{-}\text{alu\_wrapper}(\text{a\_data, b\_data, ir\_data, pc, alu\_srcA, alu\_srcB, alu\_ctrl, zero, alu\_out}); \]

- reg_wrapper
  \[ x\text{-}\text{reg\_wrapper}(\text{clk, rst, ir\_data, dr\_data, c\_data, memtoreg, regdst, write\_reg, rdata\_A, rdata\_B, r6out}); \]
Observation Info

- **Input**
  - West Button: Step execute
  - South Button: Reset
  - Slide Button: Address of Register

- **Output**
  - 0-7 Character of First line: Instruction Code
  - 8 of First line: Space
  - 9-10 of First line: Read Address
  - 11 of First line: Space
  - 12-13 of First line: Write Address
  - 0/2/4/6 of Second line: state/type/code/stage
  - 8-9 of Second line: PC
  - 11-14 of Second line: Selected Register Content
Program for verification

- <0>  lw r1, $20(r0); 0x8c01_0014 State:0,1,3,5,9 Type:3 Code:1 (LD)
- <1>  lw r2, $21(r0); 0x8c02_0015 State:0,1,3,5,9 Type:3 Code:1 (LD)
- <2>  add r3, r1, r2; 0x0022_1820 State:0,1,2,8 Type:1 Code:3 (AD)
- <3>  sub r4, r1, r2; 0x0022_2022 State:0,1,2,8 Type:1 Code:4 (SU)
- <4>  and r5, r3, r4; 0x0064_2824 State:0,1,2,8 Type:1 Code:5 (AN)
- <5>  nor r6, r4, r5; 0x0085_3027 State:0,1,2,8 Type:1 Code:6 (NO)
- <6>  sw r6, $22(r0); 0xac06_0016 State:0,1,4,7 Type:3 Code:2 (ST)
- <7>  J 0; 0x0800_0000 State:0,1 Type:2 Code:7 (JP)

- DataMem(20) = 0xbeef_0000 ;
- DataMem(21) = 0x0000_beeff ;
Precaution

- 1. Add Anti-Jitter
- 2. Finish the State Machine
- 3. Add Stage Status
Step 2: Comparison of three CPUs' work

Simple-Cycle CPU

Multiple-Cycle CPU

Pipelined CPU

CA_2013Spring_Lab 1.20
Datapath of 5-stages Pipelined CPU
The principle of Multiple-cycle CPU
Structural hazards — resource conflicts

- Structural hazards arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution.
  - Memory conflicts
  - Register File conflicts
  - Other units conflicts
How to resolve Structural hazards
Register File

- Positive edge for transfer data for stages
- Negative edge for write operation
- Low level for read operation
Memory

- Instruction Memory
  - Single Port Block Memory
  - Read only, Width:32
  - Falling Edge Triggered

- Data Memory
  - Single Port Block Memory
  - Read and write, Width:32
  - Falling Edge Triggered
The principle of Pipelined CPU—with CPU controller
## Output of CPU Controller

<table>
<thead>
<tr>
<th>Output Signal</th>
<th>Meaning When 1</th>
<th>Meaning When 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cu_branch</td>
<td>Branch Instr.</td>
<td>Non-Branch Instr.</td>
</tr>
<tr>
<td>2 Cu_shift</td>
<td>sa</td>
<td>Register data1</td>
</tr>
<tr>
<td>3 Cu_wmem</td>
<td>Write Mem.</td>
<td>Not Write Mem.</td>
</tr>
<tr>
<td>4 Cu_Mem2Reg</td>
<td>From Mem. To Reg</td>
<td>From ALUOut To Reg</td>
</tr>
<tr>
<td>5 Cu_sext</td>
<td>Sign-extend the imm.</td>
<td>No sign extended the imm.</td>
</tr>
<tr>
<td>6 Cu_aluc</td>
<td>ALU Operation</td>
<td></td>
</tr>
<tr>
<td>7 Cu_aluimm</td>
<td>Imm.</td>
<td>Register data2</td>
</tr>
<tr>
<td>8 Cu_wreg</td>
<td>Write Reg.</td>
<td>Not Write Reg.</td>
</tr>
<tr>
<td>9 Cu_regrt</td>
<td>rt</td>
<td>rd</td>
</tr>
</tbody>
</table>
Units of Pipelined-cycle CPU

- IF Stage (Instr. Mem.)
- ID Stage (CPU Ctl. And R.F.)
- EX Stage (ALU)
- Mem Stage (Data Mem.)
- WB Stage
Pipelined CPU Top Module

```verilog
module top (input wire CCLK, BTN3, BTN2, input wire [3:0]SW, output wire LED, LCDE, LCDRS, LCDRW, output wire [3:0]LCDDAT);

assign pc [31:0] = if_npc[31:0];

if_stage x_if_stage(BTN3, rst, pc, mem_pc, mem_branch, ...
  IF_ins_type, IF_ins_number,ID_ins_type,ID_ins_number);

id_stage x_id_stage(BTN3, rst, if_inst, if_pc4, wb_destR,...
  ID_ins_type, ID_ins_number, EX_ins_type, EX_ins_number..);

ex_stage x_ex_stage(BTN3, id_imm, id_inA, id_inB, id_wreg, ..
  EX_ins_type, EX_ins_number, MEM_ins_type, MEM_ins_number);

mem_stage x_mem_stage(BTN3, ex_destR, ex_inB, ex_aluR, ...
  MEM_ins_type, MEM_ins_number, WB_ins_type, WB_ins_number);

wb_stage x_wb_stage(BTN3, mem_destR, mem_aluR, ...
  WB_ins_type, WB_ins_number,OUT_ins_type, OUT_ins_number);
```

CA_2013Spring_Lab
Observation Info

- **Input**
  - West Button: Step execute
  - South Button: Reset
  - 4 Slide Button: Register Index

- **Output**
  - 0-7 Character of First line: Instruction Code
  - 8 of First line: Space
  - 9-10 of First line: Clock Count
  - 11 of First line: Space
  - 12-15 of First line: Register Content
  - Second line: “stage name”/number/type
    - stage name: 1-”f”, 2-”d”, 3-”e”, 4-”m”, 5-”w”
## Program for verification

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Bin Code</th>
<th>Address</th>
<th>Inst. Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw r1, $20(r0)</td>
<td>0x8c01_0014</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>lw r6, $21(r0)</td>
<td>0x8c06_0015</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>add r3, r0, r0</td>
<td>0x0000_1820</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>add r4, r0, r0</td>
<td>0x0000_2020</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>add r5, r0, r0</td>
<td>0x0000_2820</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>add r2, r2, r1</td>
<td>0x0041_1020</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>sub r3, r3, r1</td>
<td>0x0061_1822</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>and r4, r4, r1</td>
<td>0x0081_2024</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>nor r5, r5, r1</td>
<td>0x00a1_2827</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>beq r2, r1, -8</td>
<td>0x1041_ff8</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>
Precaution

1. Add Anti-Jitter and display for “A-F”.
2. Finish the blank.
3. Debug method: Output whatever signal to LCD Display.
4. Understand the principle of pipelined CPU and check the logic of circuit carefully, understand the sample code, then write code and synthesize the project, because it takes you a few minutes…
Something Important !!!

1. The number and type tells the information of the instruction that is to be executed in the stage.

2. How to verify the result? Pls. check the result of WB stage for R-type and LW instructions, while check the result of EXEC stage for BEQ instruction.

3. Why there are some NONE instructions following BEQ? How many NONE instructions? 3, because the condition of BEQ is generated in MEM stage.

4. Why the initial value of PC is FFFFFFFF, not 0?

5. Why we should pull the slide button after step execution to refresh the result? And instruction refresh is delayed by 1 clock-cycle? How to refresh automatically?
Thanks!