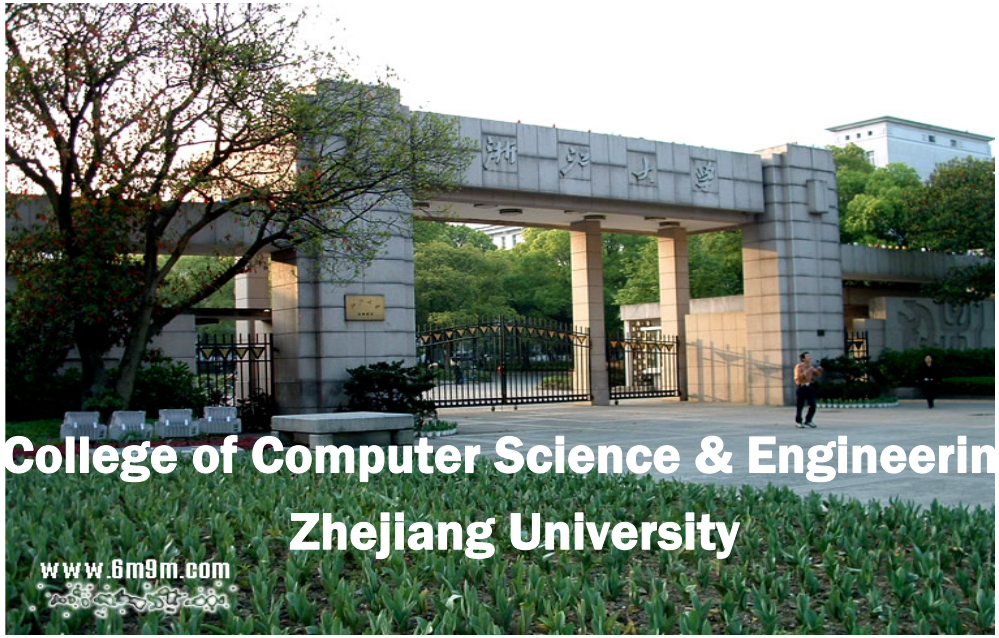
Computer Architecture Experiment

Lab2



Topics

- 0、Lab instroduction
- Lab 1). Warmup Run you 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

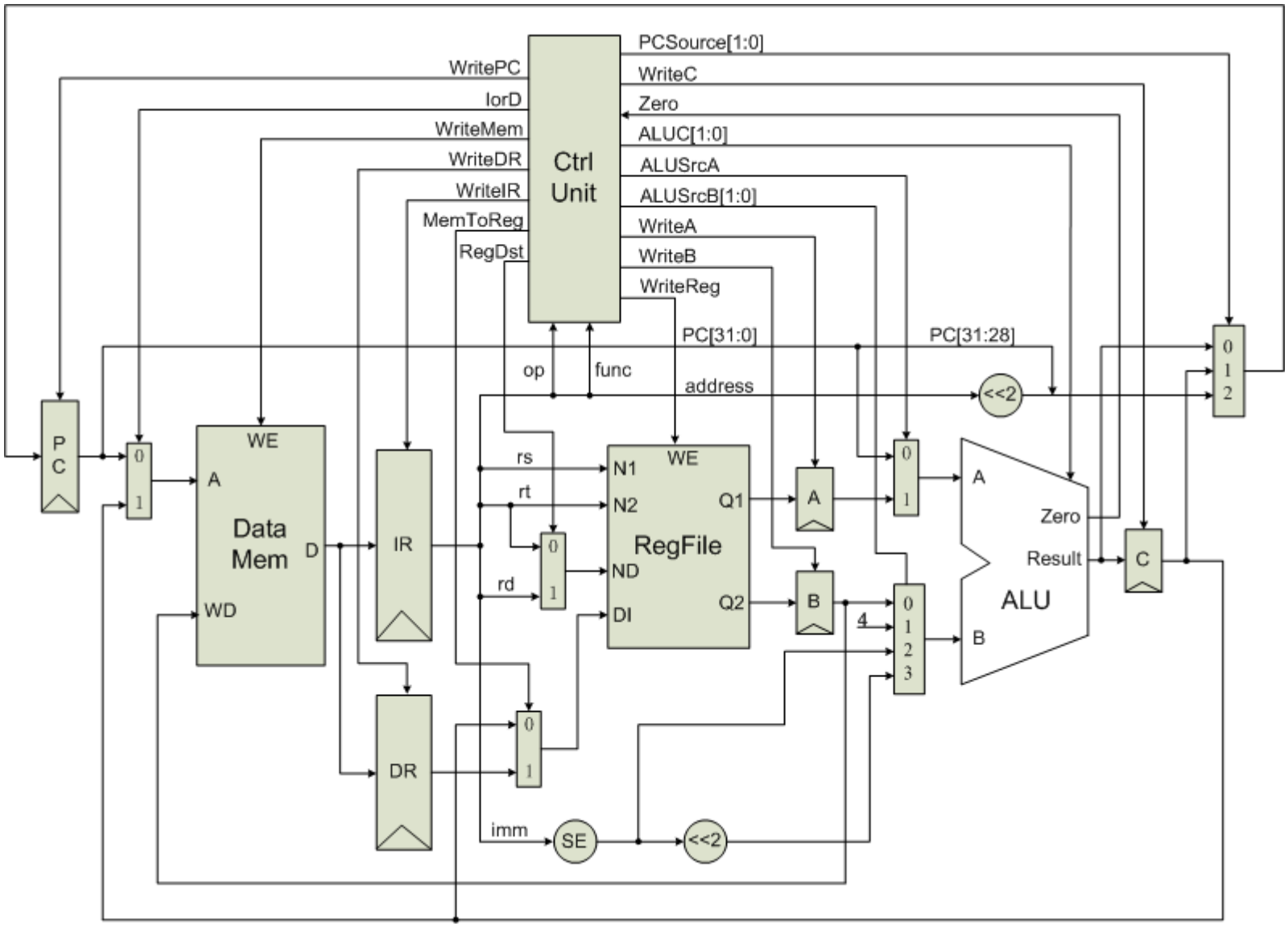
Experiment Task 2

- 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

MIPS Instructions								
Bit #	[31..26]	[25..21]	[20..16]	[15..11]	[10..06]	[05..00]	Operations	
R-type	op	rs	rt	rd	sa	func		
add	000000	rs	rt	rd	00000	100000	rd <-- rs + rt;	PC <-- PC + 4
sub	000000	rs	rt	rd	00000	100010	rd <-- rs - rt;	PC <-- PC + 4
and	000000	rs	rt	rd	00000	100100	rd <-- rs & rt;	PC <-- PC + 4
or	000000	rs	rt	rd	00000	100101	rd <-- rs | rt;	PC <-- PC + 4
sll	000000	00000	rt	rd	sa	000000	rd <-- rt << sa;	PC <-- PC + 4
srl	000000	00000	rt	rd	sa	000010	rd <-- rt >> sa (logical);	PC <-- PC + 4
sra	000000	00000	rt	rd	sa	000011	rd <-- rt >> sa (arithmetic);	PC <-- PC + 4
I-type	op	rs	rt	immediate				
addi	001000	rs	rt	immediate			rt <-- rs + (sign_extend)immediate;	PC <-- PC + 4
andi	001100	rs	rt	immediate			rt <-- rs & (zero_extend)immediate;	PC <-- PC + 4
ori	001101	rs	rt	immediate			rt <-- rs | (zero_extend)immediate;	PC <-- PC + 4
lw	100011	rs	rt	immediate			rt <-- memory[rs + (sign_extend)immediate];	PC <-- PC + 4
sw	101011	rs	rt	immediate			memory[rs + (sign_extend)immediate] <-- rt;	PC <-- PC + 4
beq	000100	rs	rt	immediate			if (rs == rt) PC <-- PC + 4 + (sign_extend)immediate<<2; else	PC <-- PC + 4
bne	000101	rs	rt	immediate			if (rs != rt) PC <-- PC + 4 + (sign_extend)immediate<<2; else	PC <-- PC + 4
J-type	op	address						
j	000010	address					PC <-- (PC+4)[31..28],address<<2	

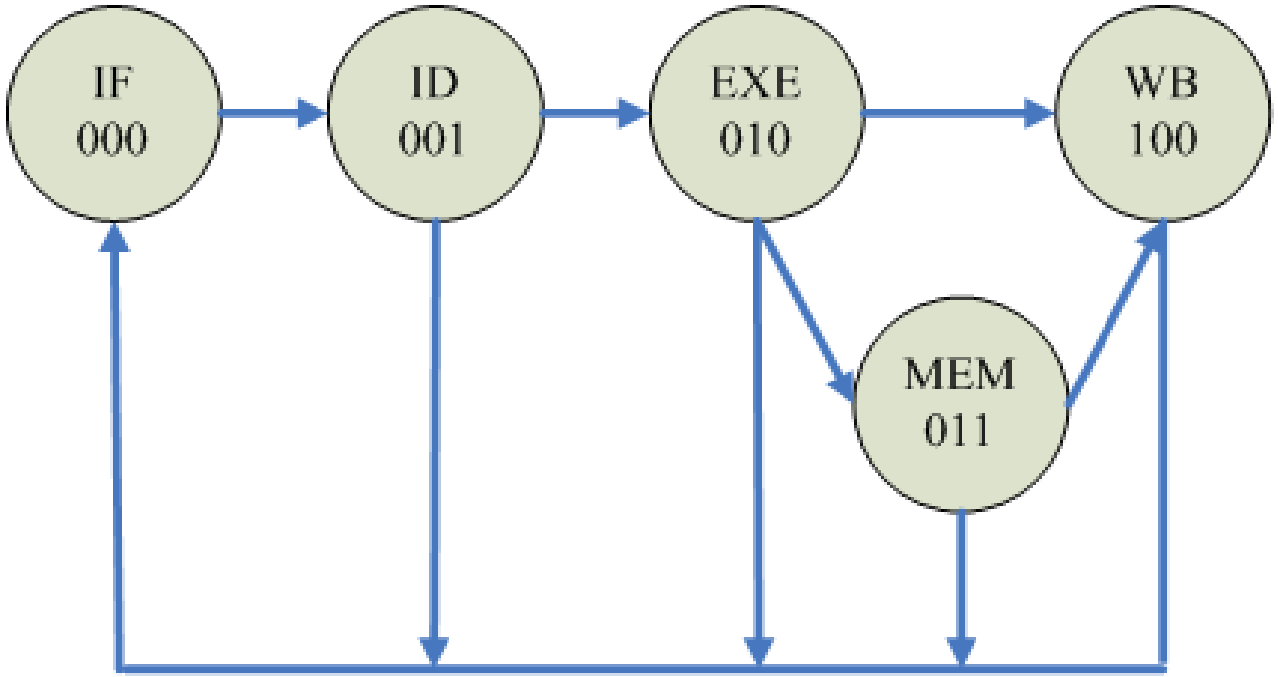
Step1: CPU Controller



Output of CPU Controller

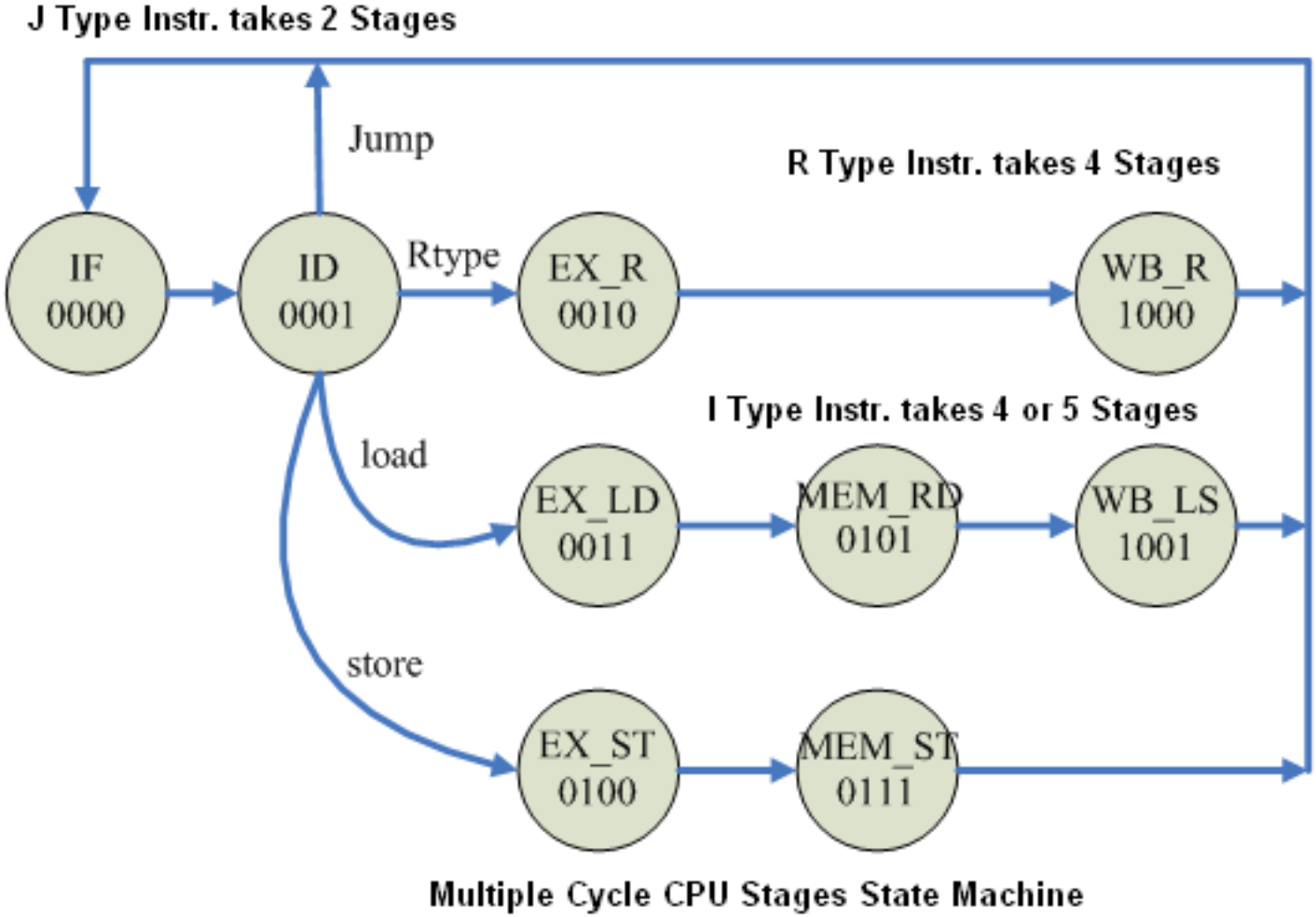
	Output Signal	Meaning When 1	Meaning When 0
1	PCSrc[1:0]	00: PC + 4;01: Branch Instr.;10: jump Instr	
2	WritePC	Write PC	Not Write PC
3	IorD	Instruction Addr	Data Addr.
4	WriteMem	Write Mem.	Not Write Mem.
5	Write DR	Write Data. Reg	Not Write Data. Reg
6	Write IR	Write Instr. Reg	Not Write Instr. Reg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

The principle of CPU Controller(1)

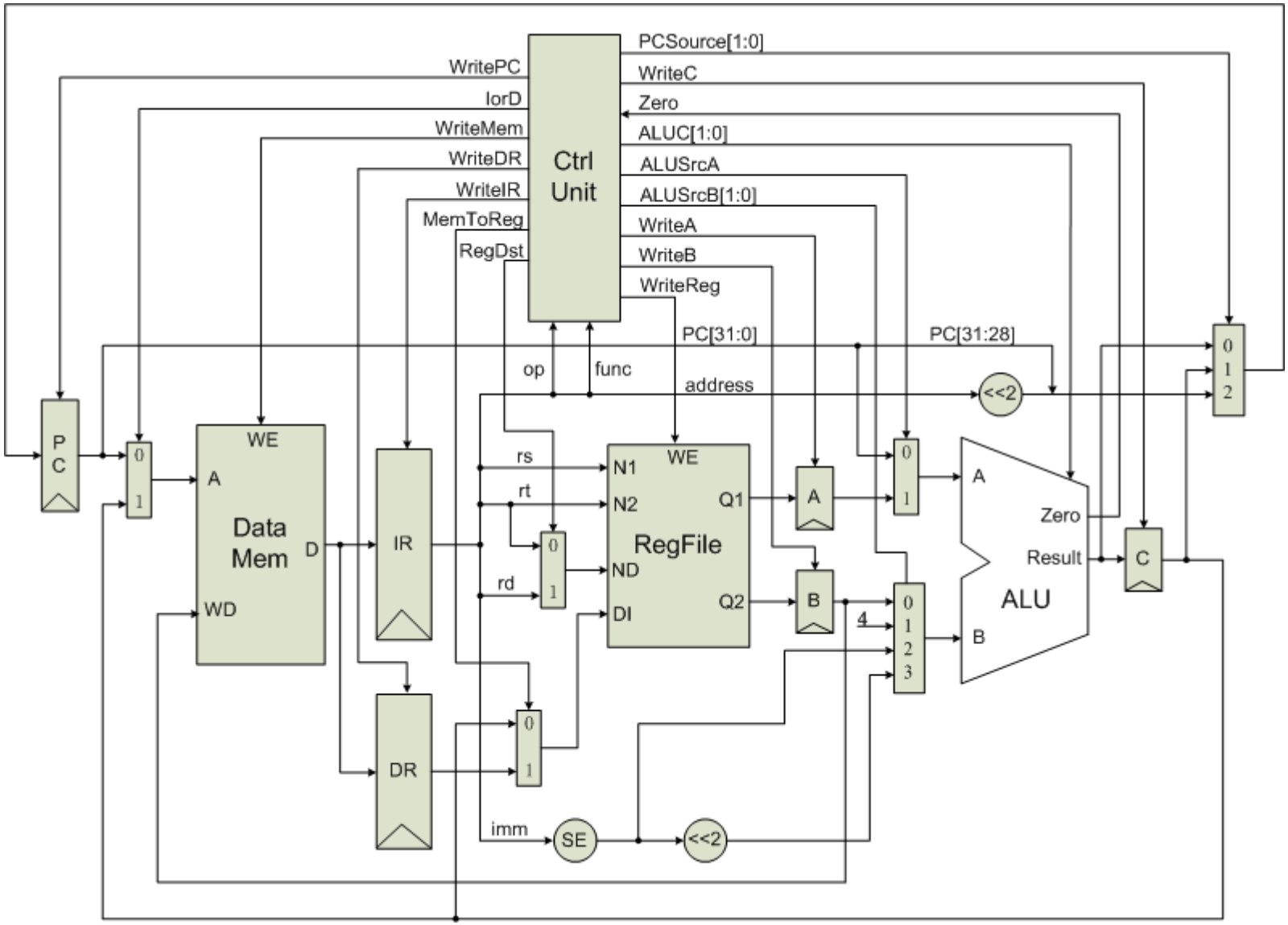


Stages of Multiple-Cycle Execution of Typical MIPS CPU

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

- 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_memory(.addra(raddr),.addrb(waddr),.clka(clk), .clkb(clk),.dinb(b_data), .douta(mem_data),.web(write_mem));

- ctrl x_ctrl(clk, rst, ir_data, zero,write_pc, iord, write_mem, write_dr, write_ir, memtoreg, regdst, pcsource, write_c, alu_ctrl, alu_srcA, alu_srcB, write_a, write_b, write_reg, state_out, insn_type, insn_code, insn_stage);

- pcm x_pcm(clk, rst, alu_out, c_data, ir_data, pcsource, write_pc,pc);

- alu_wrapper x_alu_wrapper(a_data, b_data, ir_data, pc, alu_srcA, alu_srcB, alu_ctrl, zero, alu_out);

- reg_wrapper x_reg_wrapper(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); 0x ac06_0016 State:0,1,4,7 Type:3 Code: 2 (ST)
- <7> J 0; 0x0800_0000 State:0,1 Type:2 Code:7 (JP)

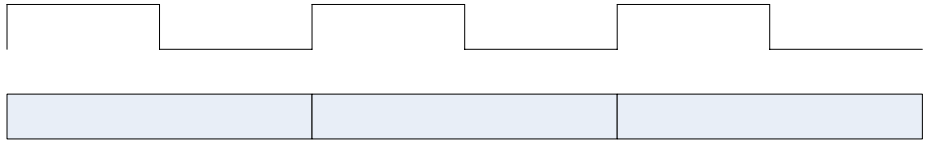
- DataMem(20) = 0xbeef_0000 ;
- DataMem(21) =0x0000_beef ;

Precaution

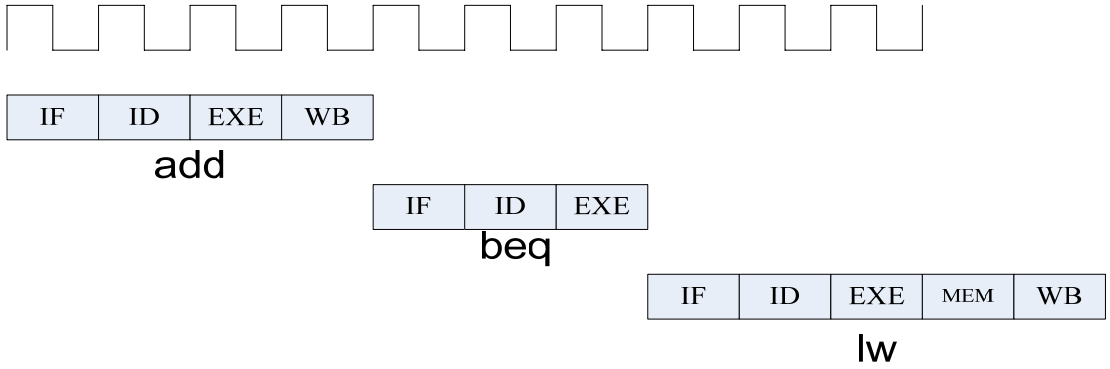
- 1. Add Anti-Jitter
- 2. Finish the State Machine
- 3. Add Stage Status

Step2 Comparison of three CPUs' work

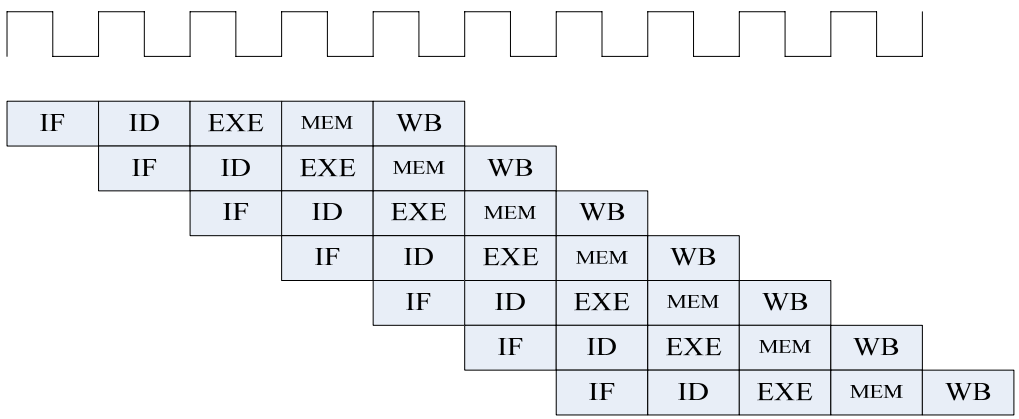
Simple-Cycle CPU



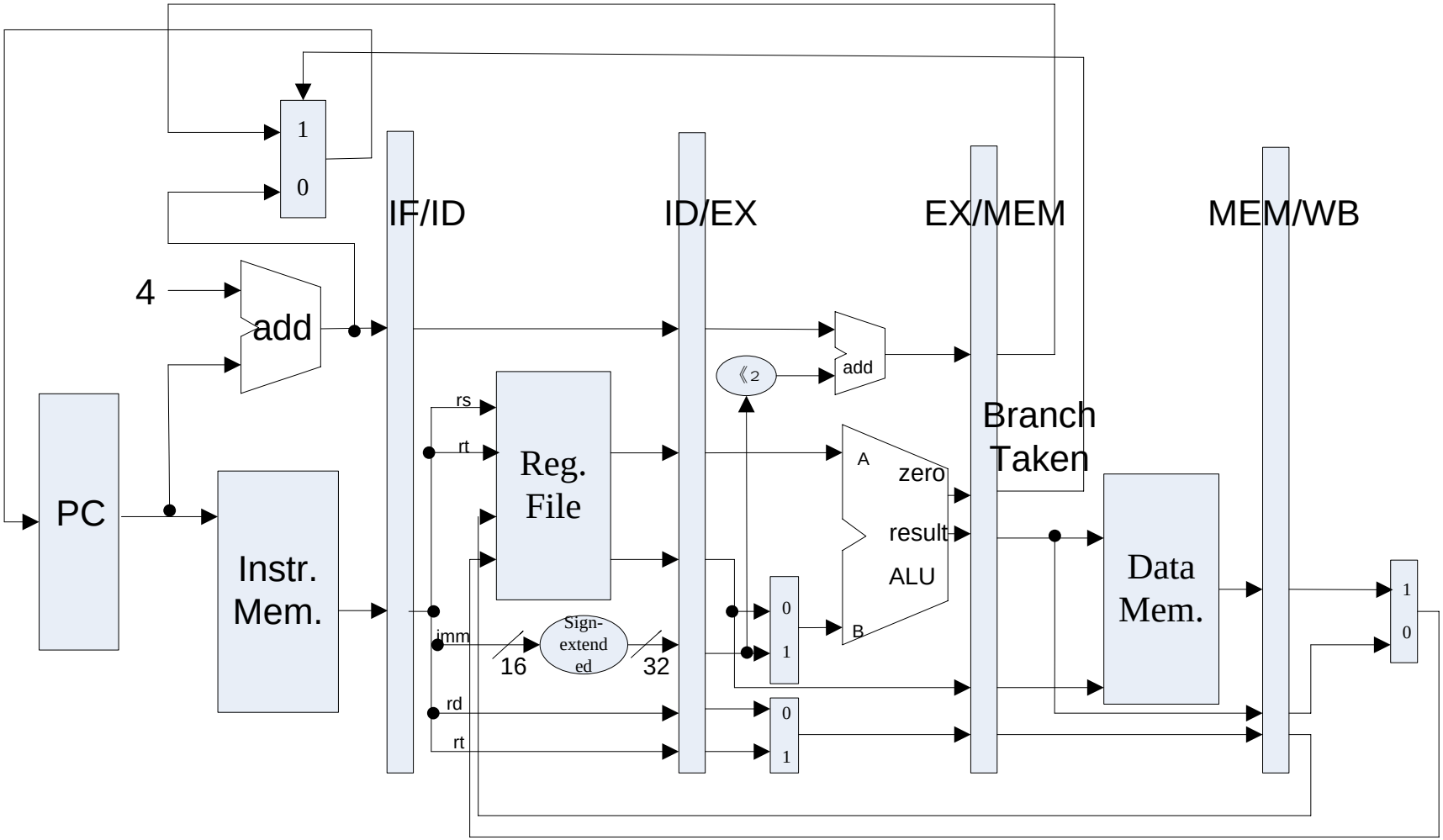
Multiple-Cycle CPU



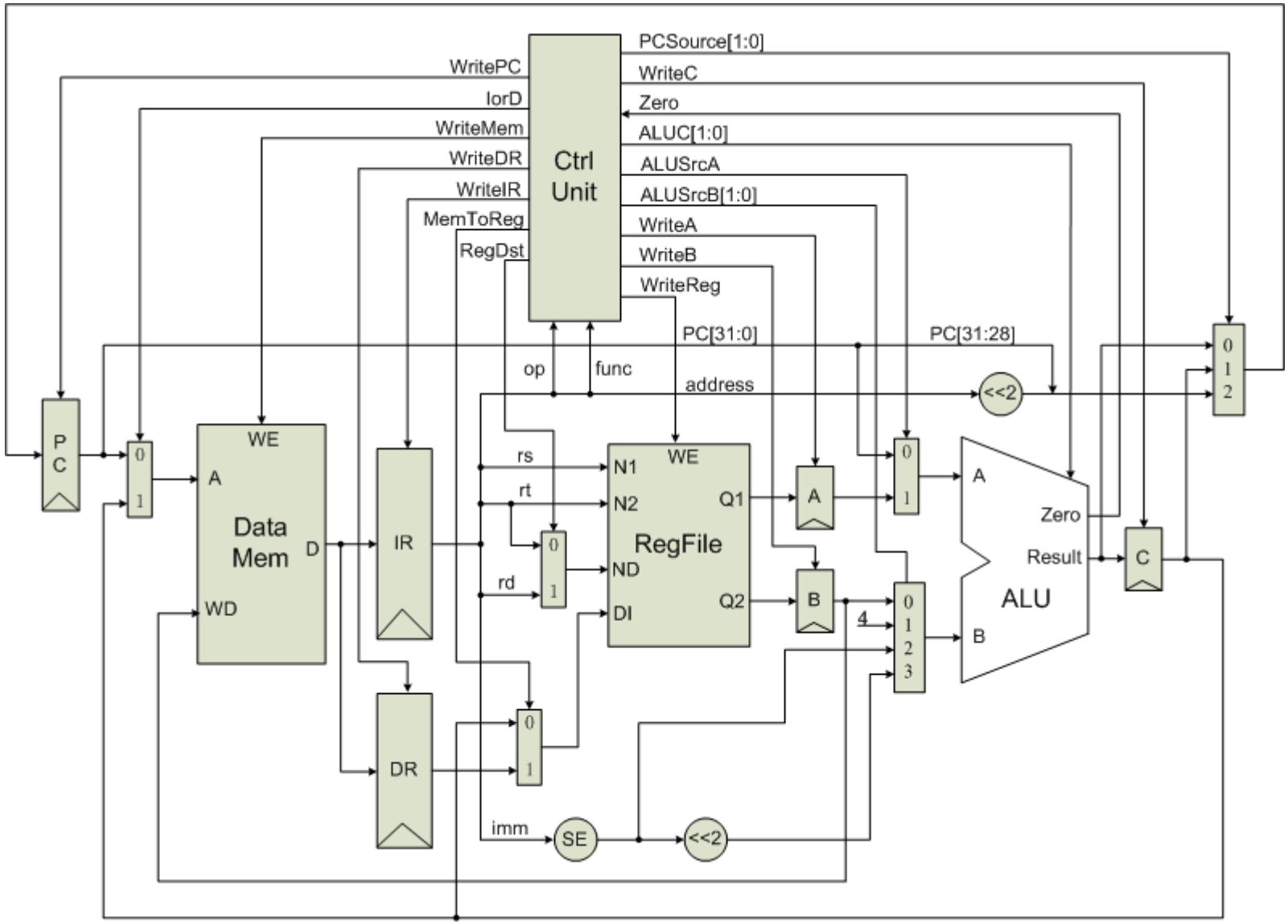
Pipelined CPU



Datapath of 5-stages Pipelined CPU



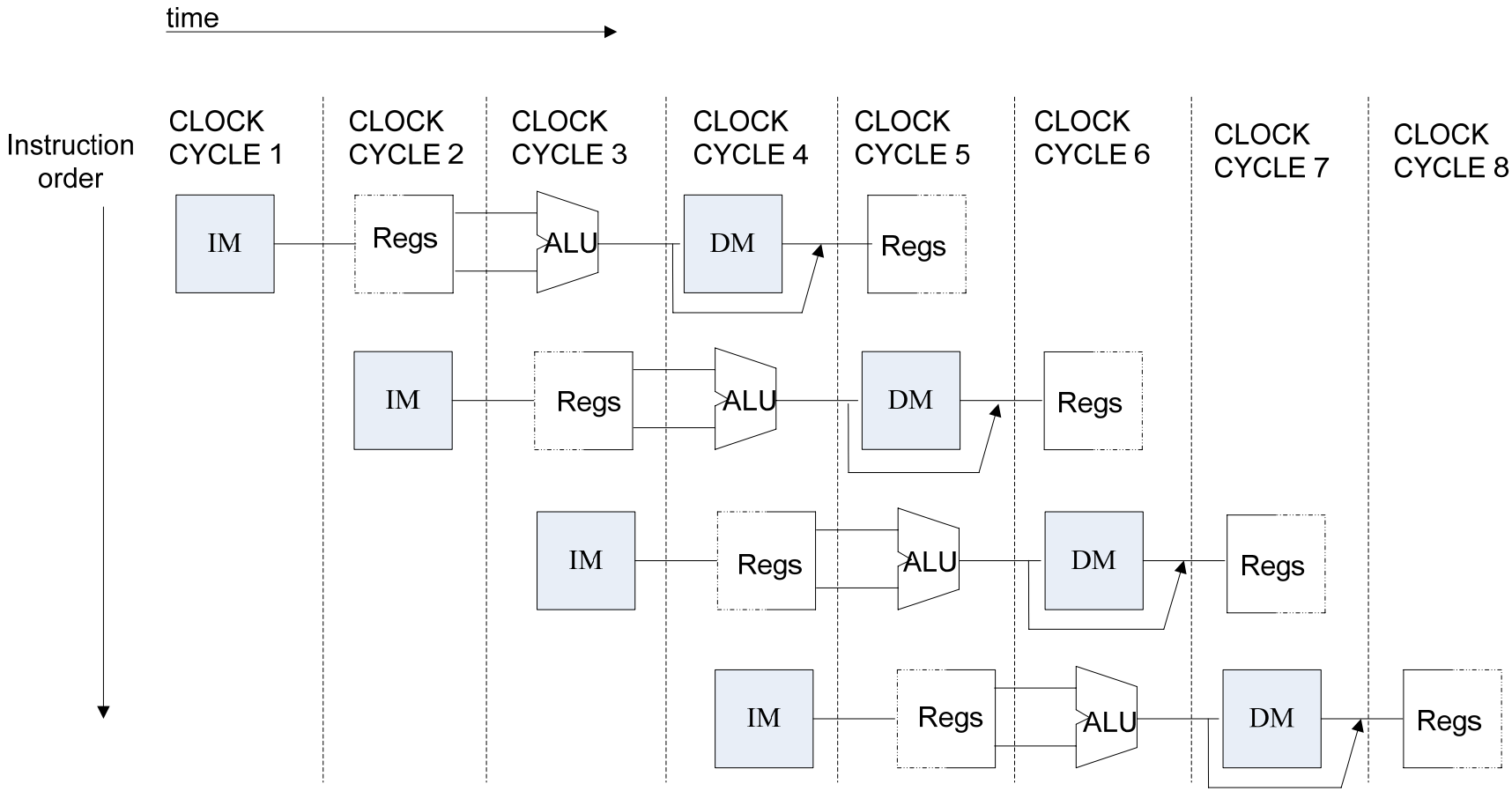
The principle of Multiple-cycle CPU



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



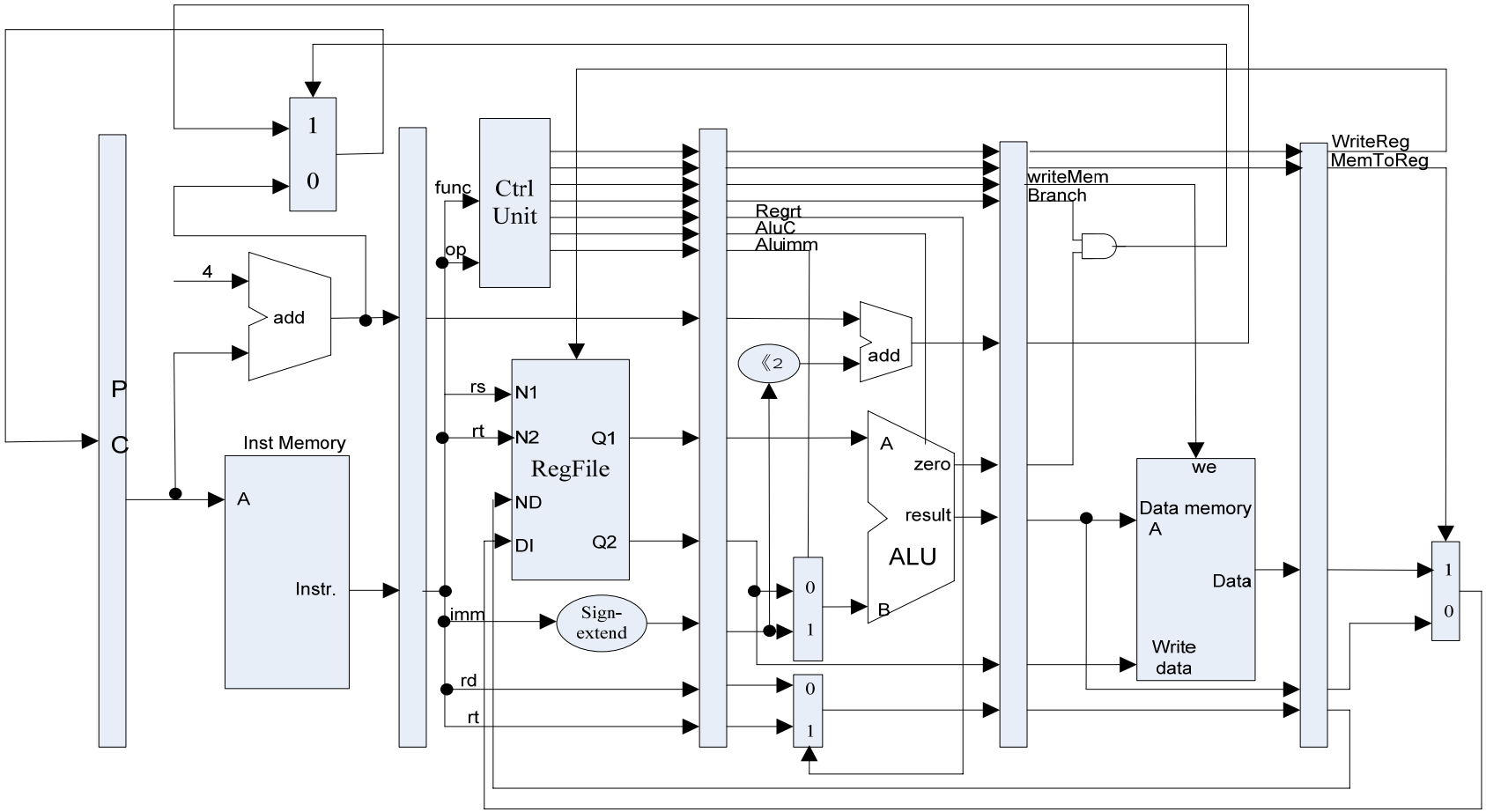
Register File

- 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



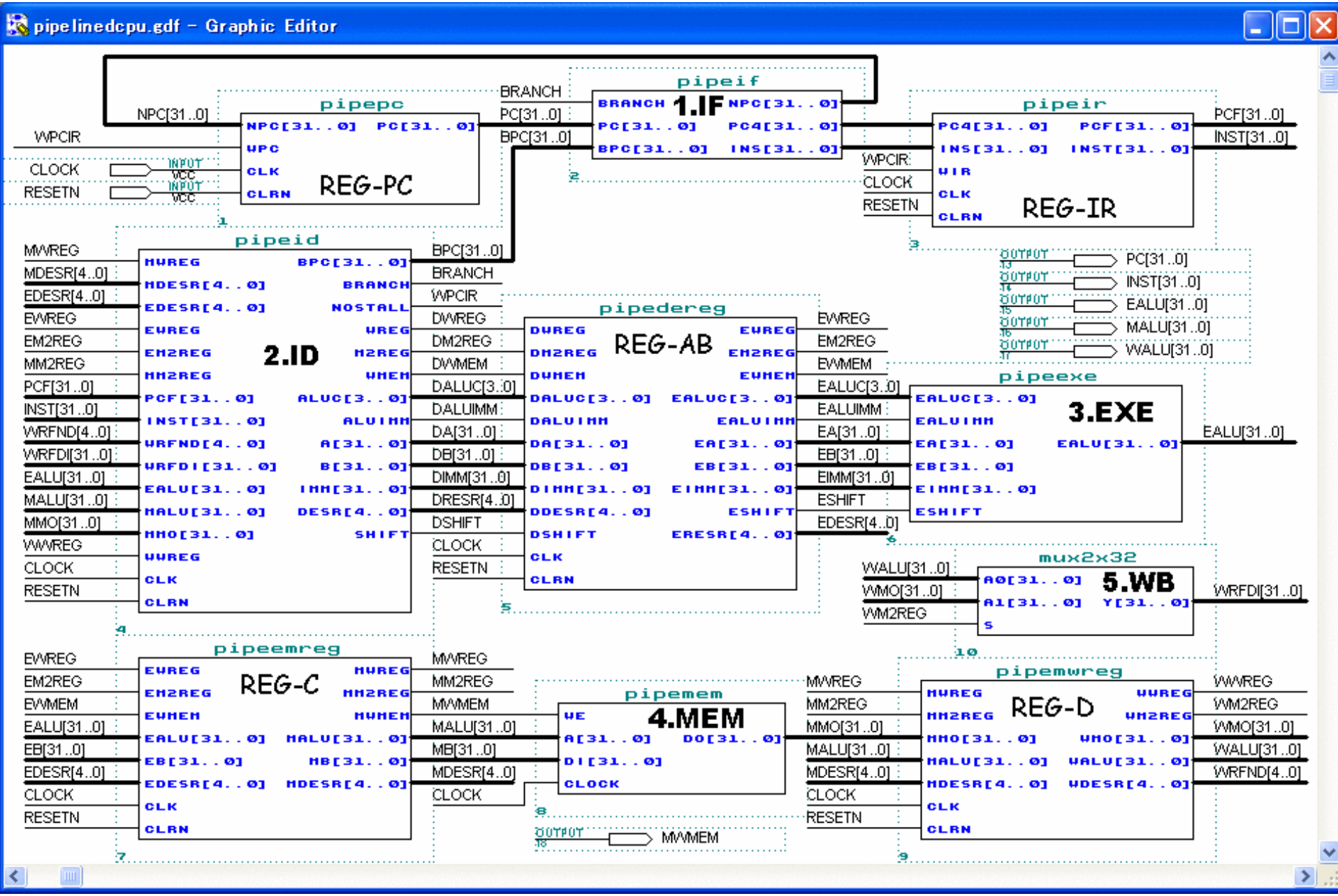
IF	ID	EXE	MEM	WB

Output of CPU Controller

	Output Signal	Meaning When 1	Meaning When 0
1	Cu_branch	Branch Instr.	Non-Branch Instr.
2	Cu_shift	sa	Register data1
3	Cu_wmem	Write Mem.	Not Write Mem.
4	Cu_Mem2Reg	From Mem. To Reg	From ALUOut To Reg
5	Cu_sext	Sign-extend the imm.	No sign extended the imm.
6	Cu_aluc	ALU Operation	
7	Cu_aluimm	Imm.	Register data2
8	Cu_wreg	Write Reg.	Not Write Reg.
9	Cu_regrt	rt	rd

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


pipelinedcpu.gdf - Graphic Editor
pipepc
REG-PC
pipeif
1.IF
pipeir
REG-IR
pipeid
2.ID
pipedereg
REG-AB
pipeexe
3.EXE
mux2x32
5.WB
pipeemreg
REG-C
pipemem
4.MEM
pipemwreg
REG-D
NPC[31..0]
WPCIR
CLOCK
RESETN
BRANCH
PC[31..0]
BPC[31..0]
PCF[31..0]
INST[31..0]
EALU[31..0]
MALU[31..0]
WALU[31..0]
WRFDI[31..0]
MWMEM

Pipelined CPU Top Module

- 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);

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

	Instruction	Bin Code	Address	Inst. Type
1	lw r1, $20(r0)	0x8c01_0014	0	6
2	lw r6, $21(r0)	0x8c06_0015	1	6
3	add r3,r0,r0	0x0000_1820	2	1
4	add r4,r0,r0	0x0000_2020	3	1
5	add r5,r0,r0	0x0000_2820	4	1
6	add r2,r2,r1	0x0041_1020	5	1
7	sub r3, r3, r1	0x0061_1822	6	2
8	and r4, r4, r1	0x0081_2024	7	3
9	nor r5, r5, r1	0x00a1_2827	8	5
10	beq r2, r1, -8	0x1041_fff8	9	8

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!