Computer Architecture Experiment

Lab2

College of Computer Science & Engineerin

Zhejiang University

Topics

- 0、Lab instroduction
- Lab 1). Warmup Run you single-cycle CPU on 3E board. Try to implement 15 instructions.
- 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

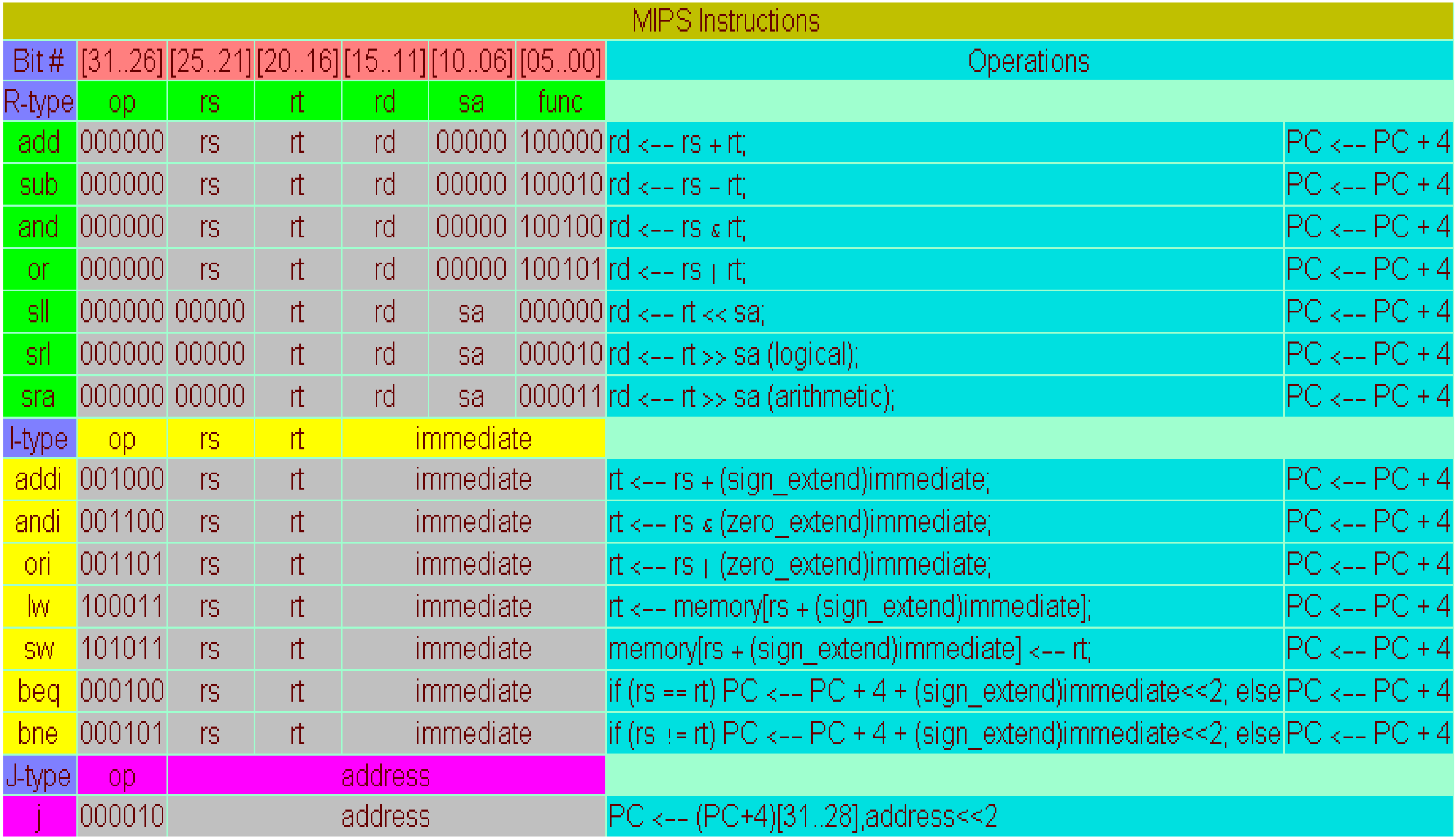
Experiment Purpose 1

- Understand the principles of 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

- 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

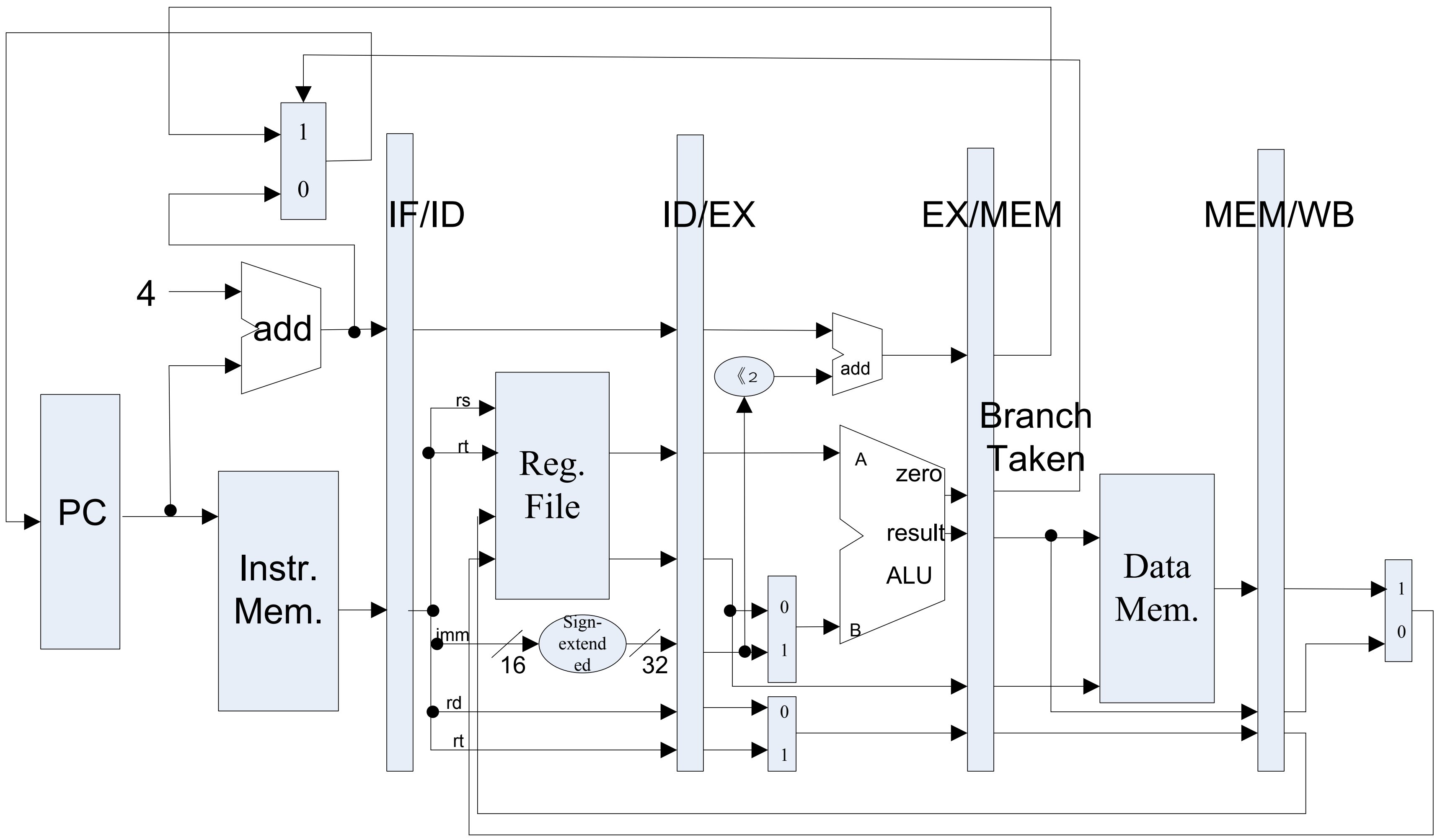


Instr. No	MIPS Instructions								
	Bit #	[31..26]	[25..21]	[20..16]	[15..11]	[10..06]	[05..00]	Operations	
	R-type	op	rs	rt	rd	sa	func		
01	add	000000	rs	rt	rd	00000	100000	rd <-- rs + rt;	PC <-- PC + 4
02	sub	000000	rs	rt	rd	00000	100010	rd <-- rs - rt;	PC <-- PC + 4
03	and	000000	rs	rt	rd	00000	100100	rd <-- rs & rt;	PC <-- PC + 4
04	or	000000	rs	rt	rd	00000	100101	rd <-- rs | rt;	PC <-- PC + 4
05	sll	000000	00000	rt	rd	sa	000000	rd <-- rt << sa;	PC <-- PC + 4
06	srl	000000	00000	rt	rd	sa	000010	rd <-- rt >> sa (logical);	PC <-- PC + 4
07	sra	000000	00000	rt	rd	sa	000011	rd <-- rt >> sa (arithmetic);	PC <-- PC + 4
	I-type	op	rs	rt	immediate				
08	addi	001000	rs	rt	immediate			rt <-- rs + (sign_extend)immediate;	PC <-- PC + 4
09	andi	001100	rs	rt	immediate			rt <-- rs & (zero_extend)immediate;	PC <-- PC + 4
0A	ori	001101	rs	rt	immediate			rt <-- rs | (zero_extend)immediate;	PC <-- PC + 4
0B	lw	100011	rs	rt	immediate			rt <-- memory[rs + (sign_extend)immediate];	PC <-- PC + 4
0C	sw	101011	rs	rt	immediate			memory[rs + (sign_extend)immediate] <-- rt;	PC <-- PC + 4
0D	beq	000100	rs	rt	immediate			if (rs == rt) PC <-- PC + 4 + (sign_extend)immediate<<2; else	PC <-- PC + 4
0E	bne	000101	rs	rt	immediate			if (rs != rt) PC <-- PC + 4 + (sign_extend)immediate<<2; else	PC <-- PC + 4
	J-type	op	address						
0F	j	000010	address					PC <-- (PC+4)[31..28],address<<2	

Precaution

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

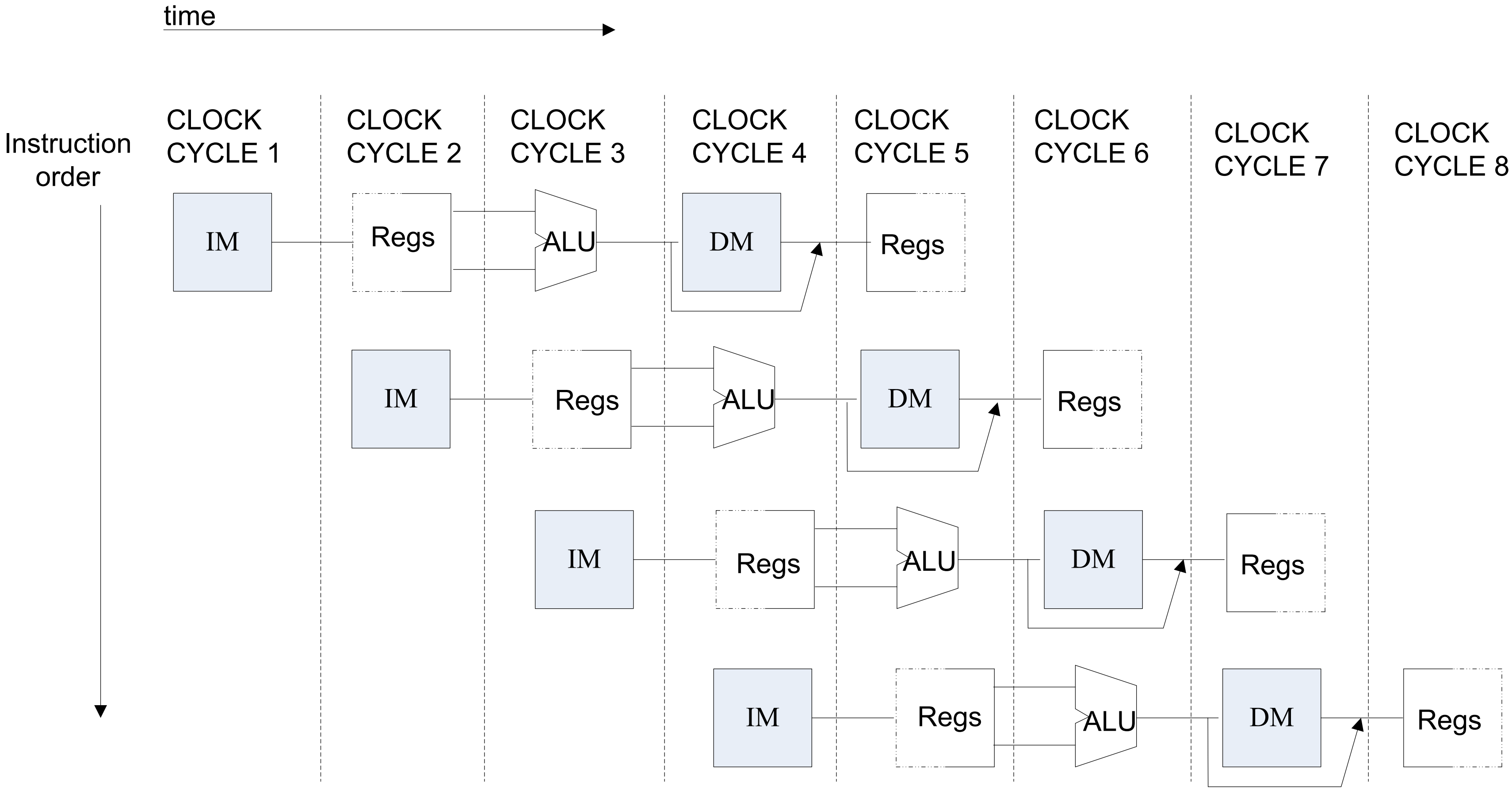
Datapath of 5-stages Pipelined CPU



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



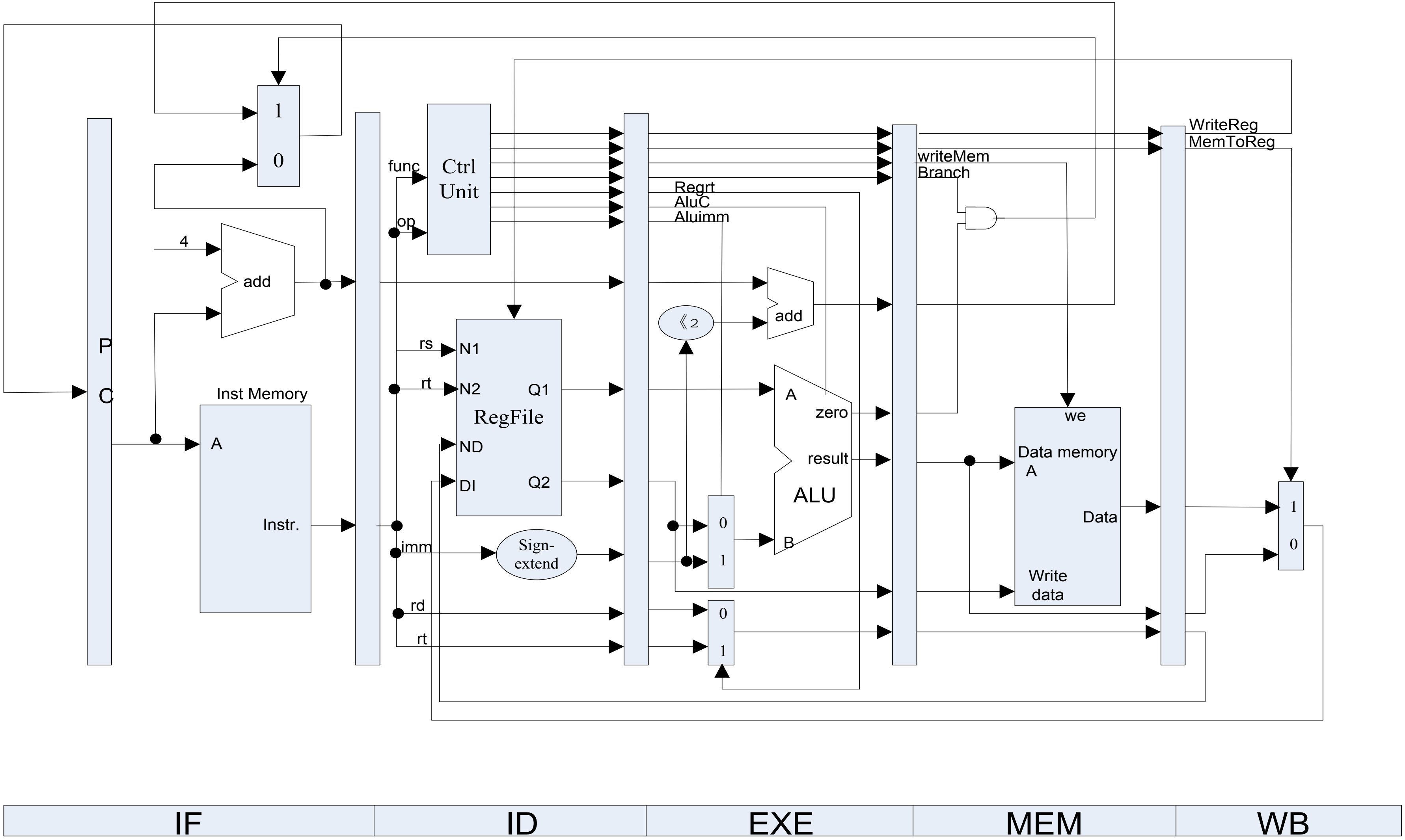
Register File

- 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

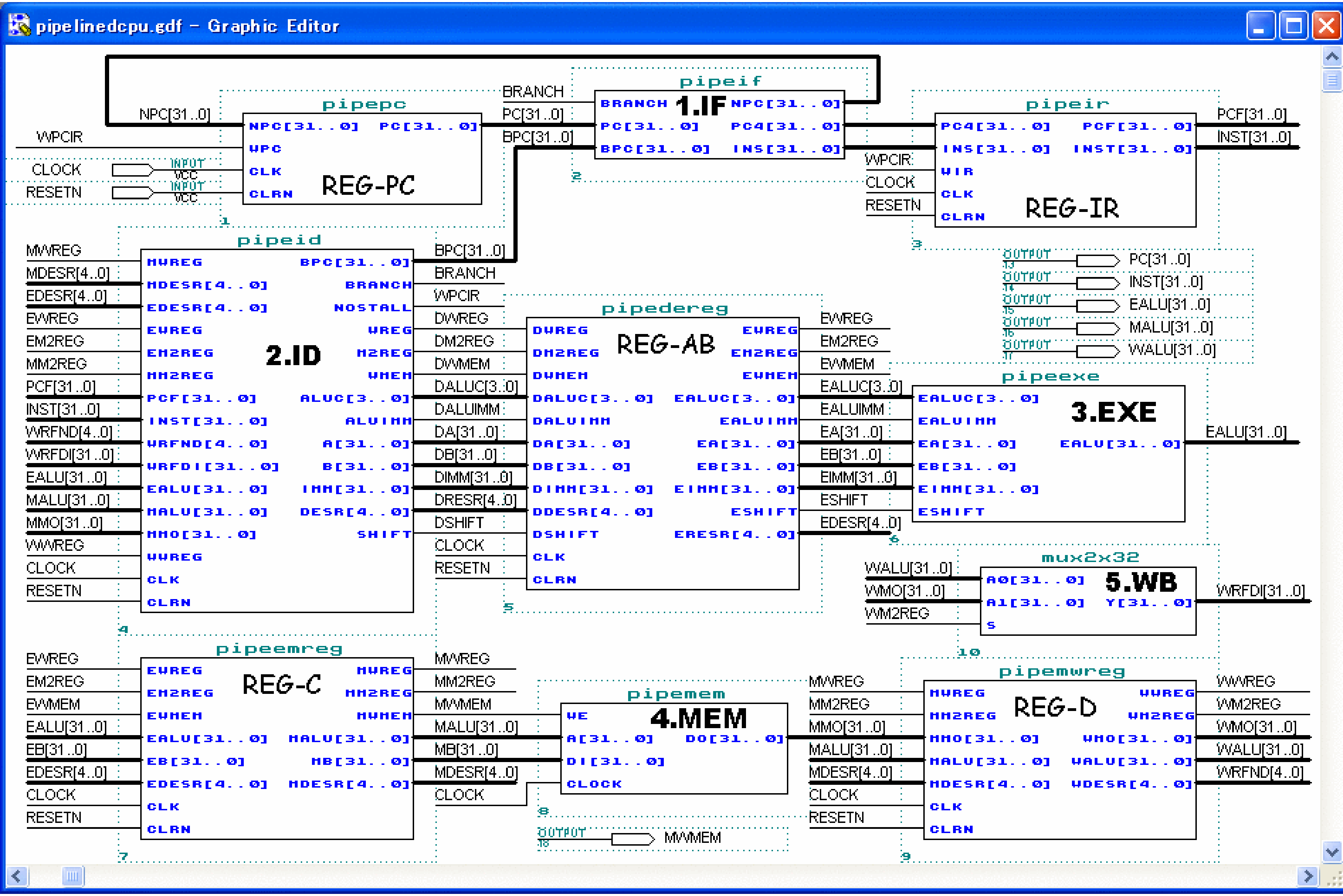


Output of CPU Controller(not limited)

	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
NPC[31..0]
PC[31..0]
WPC
CLK
CLRN
NPC[31..0]
WPCIR
CLOCK
RESETN
INPUT
VCC
1
pipeif
1.IF
BRANCH
PC[31..0]
BPC[31..0]
NPC[31..0]
PC4[31..0]
INS[31..0]
2
pipeir
REG-IR
PC4[31..0]
INS[31..0]
WIR
CLK
CLRN
PCF[31..0]
INST[31..0]
3
OUTPUT
PC[31..0]
INST[31..0]
EALU[31..0]
MALU[31..0]
WALU[31..0]
pipeid
2.ID
MWREG
MDESR[4..0]
EDESR[4..0]
EWREG
EM2REG
MM2REG
PCF[31..0]
INST[31..0]
WRFND[4..0]
WRFDI[31..0]
EALU[31..0]
MALU[31..0]
MMO[31..0]
WWREG
CLK
CLRN
BPC[31..0]
BRANCH
NOSTALL
WREG
M2REG
WMEM
ALUC[3..0]
ALUIMM
A[31..0]
B[31..0]
IMM[31..0]
DESR[4..0]
SHIFT
4
pipedereg
REG-AB
DWREG
DM2REG
DWMEM
DALUC[3..0]
DALUIMM
DA[31..0]
DB[31..0]
DIMM[31..0]
DDESR[4..0]
DSHIFT
EWREG
EM2REG
EWMEM
EALUC[3..0]
EALUIMM
EA[31..0]
EB[31..0]
EIMM[31..0]
ESHIFT
ERESR[4..0]
EDESR[4..0]
5
pipeexe
3.EXE
EALUC[3..0]
EALUIMM
EA[31..0]
EB[31..0]
EIMM[31..0]
ESHIFT
EALU[31..0]
6
mux2x32
5.WB
A0[31..0]
A1[31..0]
S
Y[31..0]
WALU[31..0]
WMO[31..0]
WM2REG
WRFDI[31..0]
10
pipeemreg
REG-C
EWREG
EM2REG
EWMEM
EALU[31..0]
EB[31..0]
EDESR[4..0]
MWREG
MM2REG
MWMEM
MALU[31..0]
MB[31..0]
MDESR[4..0]
7
pipemem
4.MEM
WE
A[31..0]
DI[31..0]
CLOCK
DO[31..0]
8
MWMEM
pipemwreg
REG-D
MWREG
MM2REG
MMO[31..0]
MALU[31..0]
MDESR[4..0]
WWREG
WM2REG
WMO[31..0]
WALU[31..0]
WDESR[4..0]
WRFND[4..0]
9

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
  - One Button: Step execute
  - One Button: Reset
  - One Button + 4 Switches : Register Index
- Output (Led screen)
  - 0-7 Character of First line: Instruction Code at ID-stage
  - 8-15 of Character of First line : Register Value
  - Second line : F D06 | E03 | M07 | W0A (06/03/07/0A：inst. No.)

Prepare for checking

- Test code for /Lab2 can be downloaded from the material directory on course website.

- Do understand what you have implemented.
  - Prepare to answer questions on your results (Verilog code, logical graph, UCF…).

Precaution

- 1. Add Anti-Jitter for buttons.
- 2. Modified your code based on your lab1 code.
- 3. Debug method: Output whatever signal to Led lamp or 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、If you want to execute the test code and get the right result on your pre-pipelined CPU, how should you modify the test code ?
- 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!