Computer Architecture Lab

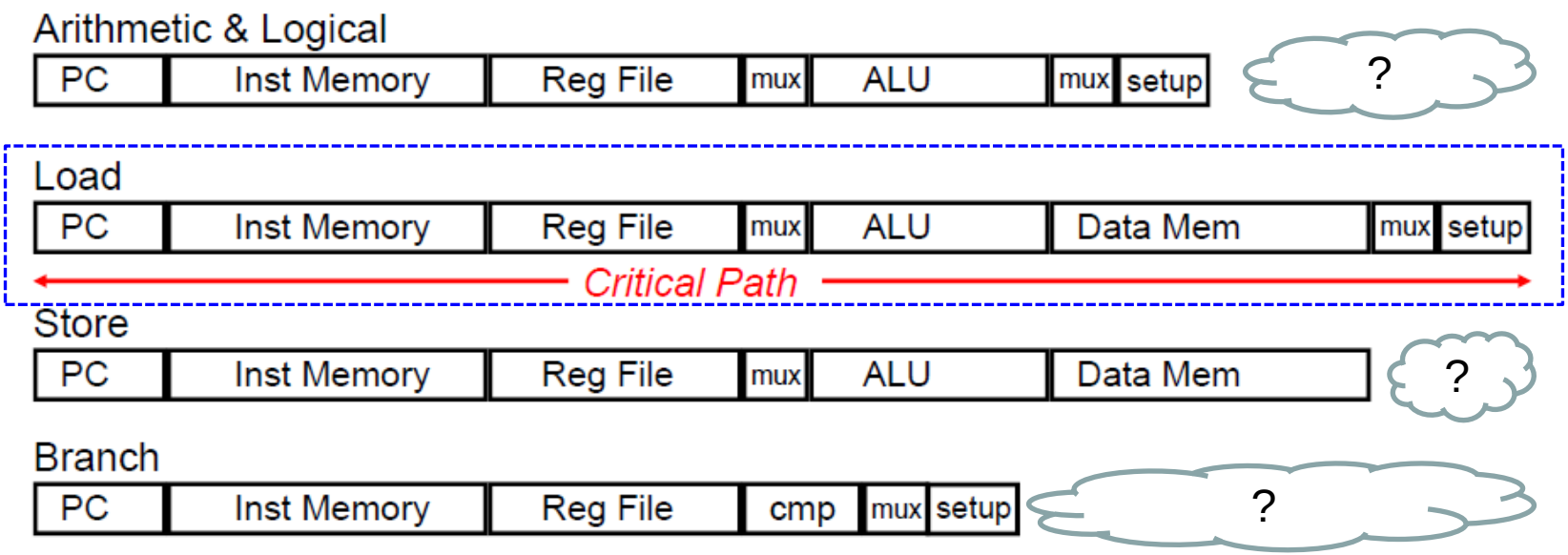
Lab07 – Week #7

MPLAB
Media Processing Laboratory

컴퓨터 공학과 이성원 교수님 연구실

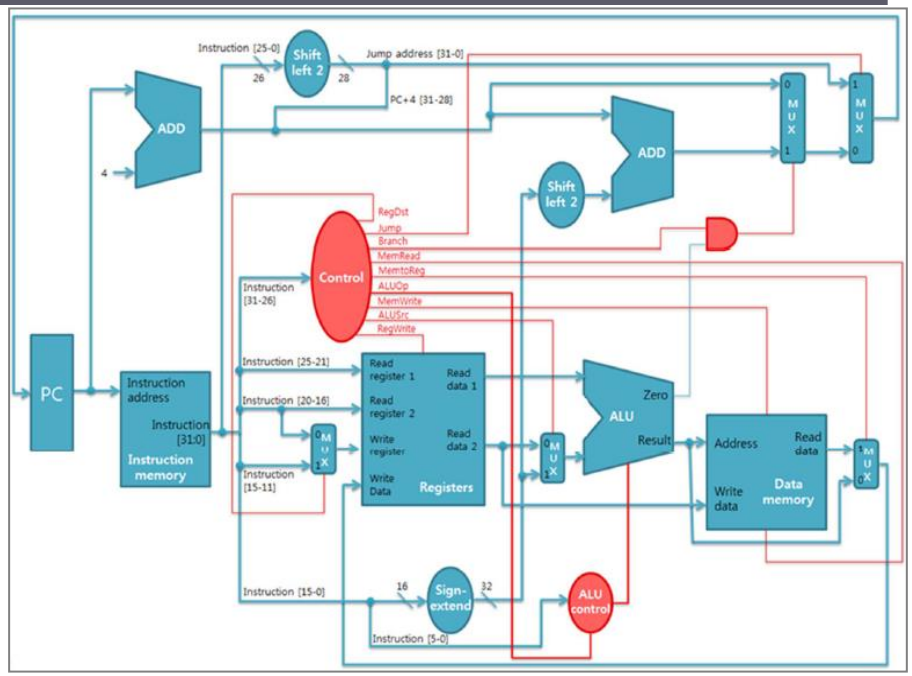
CPI=1 processor's problem

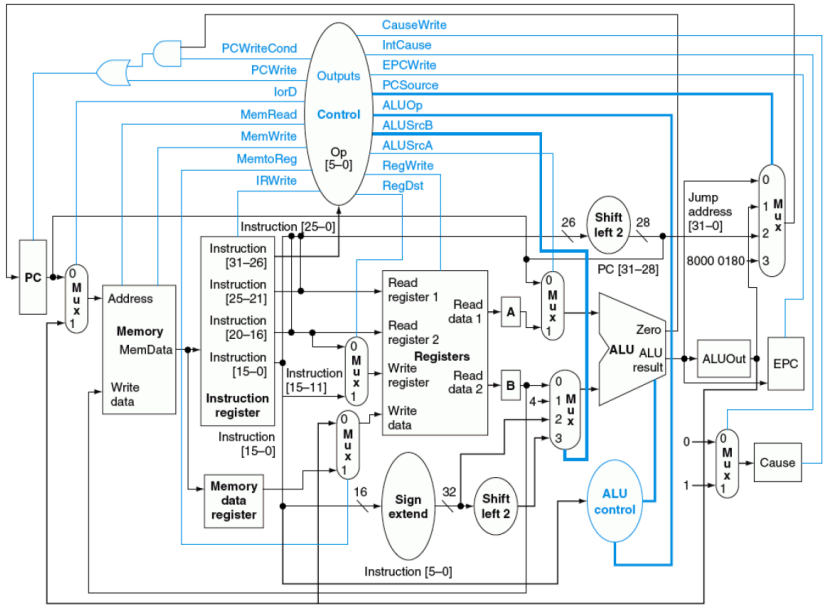
Long Cycle Time (LOAD instr.)
All instructions take as much time as the slowest



Multi-Cycle Approach

- Single memory unit
- Single ALU (2 Adder X)
- Registers after every major functional unit



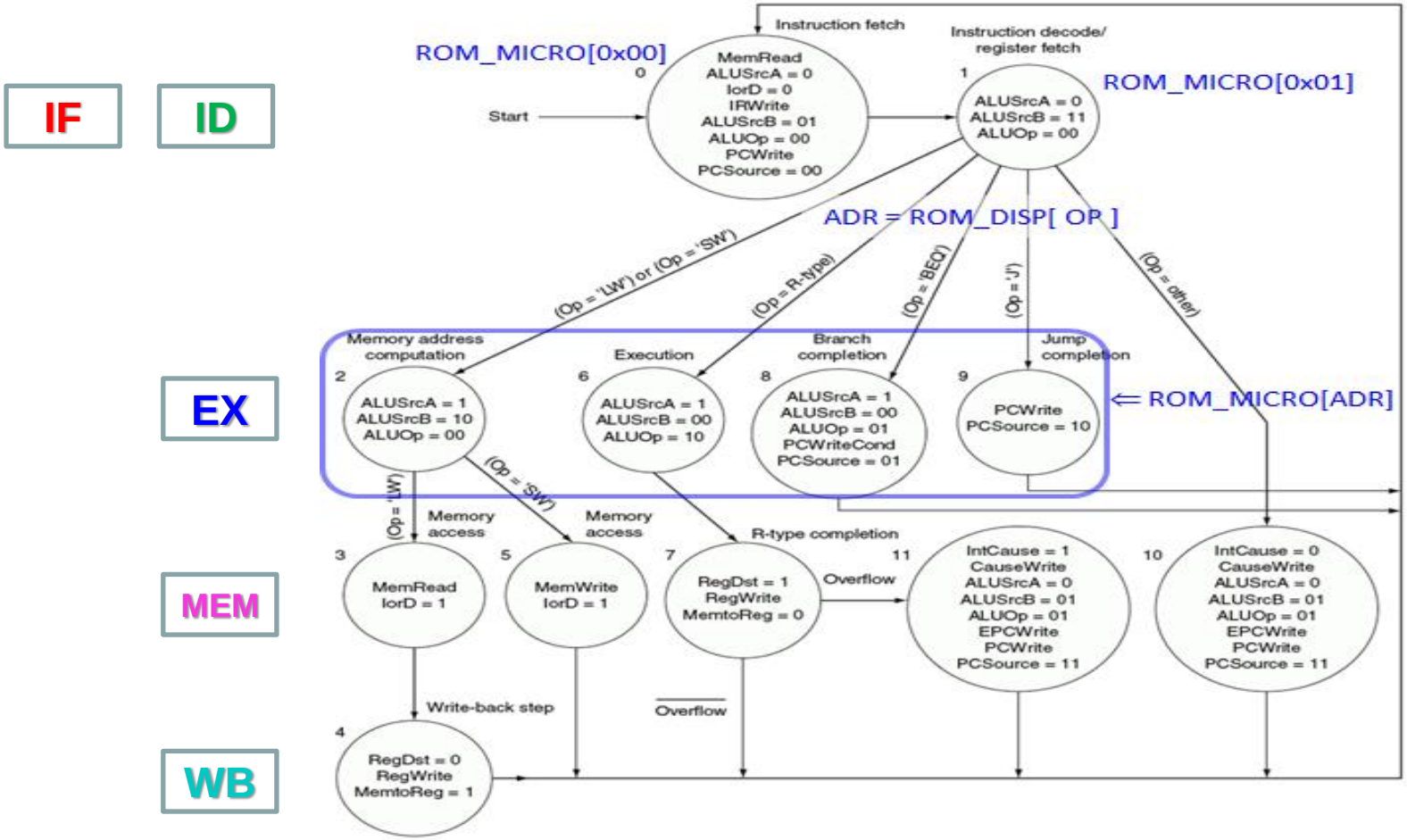


Multi-Cycle

Stage #	Action for R-type instructions	Action for Memory instructions	Action for Branches	Action for Jumps	Remarks
0	IR = Memory[PC] PC = PC + 4				Instr Fetch
1	A = Reg [IR[25:21]] B = Reg [IR[20:16]] ALUOut = PC + (Sign-Extended (IR[15:0]) << 2)				Instr Decode & Register Fetch
2	ALUOut = A op B	ALUOut = A + Sign-Extended (IR[15:0])	If (A==B) then PC = ALUOut	PC = { PC[31:28], (IR[25:0]<<2) }	Execution or Branch/Jump Completion
3	Reg [IR[15:11]] = ALUOut	Load: MDR = Memory[ALUOut] Store: Memory [ALUOut] = B			Memory Access & R-type Completion
4		Load: Reg [IR[20:16]] = MDR			Memory Read Completion

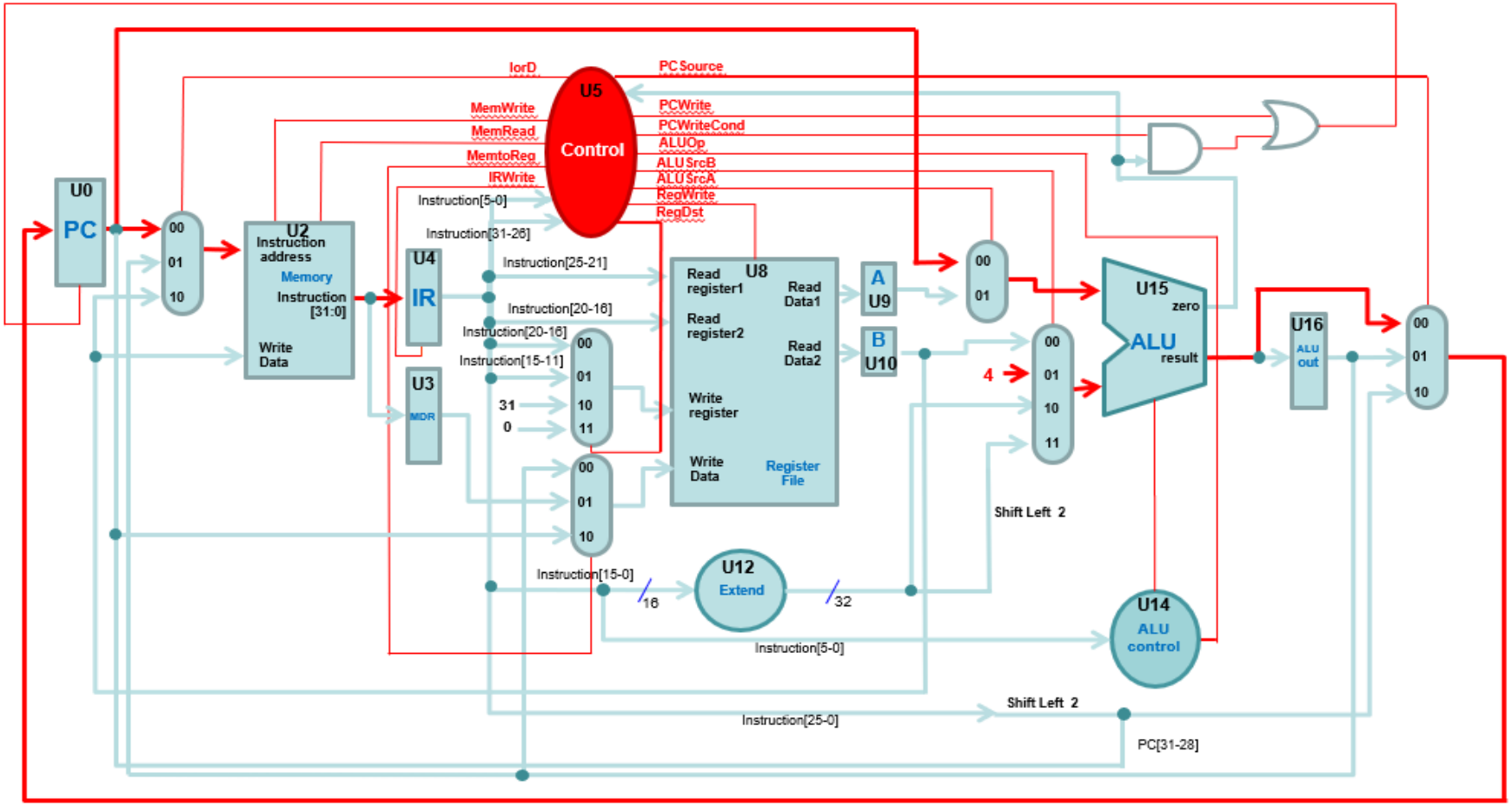
Figure 2 – Example Execution Stages for The multi-cycle datapath

Multi-Cycle CPU FSM Diagram



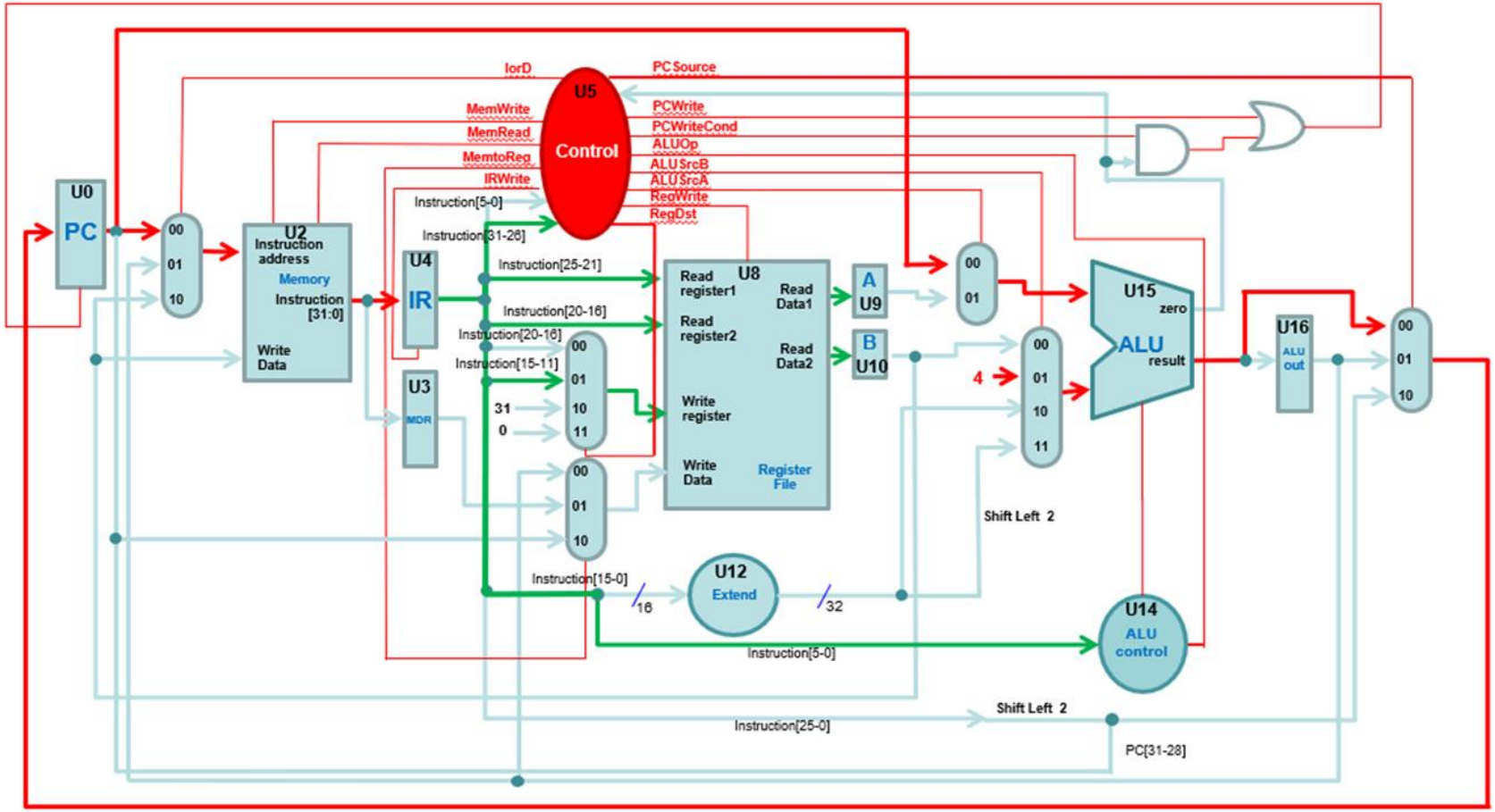
R-type(S0: IF(1))

0	IR = Memory[PC] PC = PC + 4	Instr Fetch



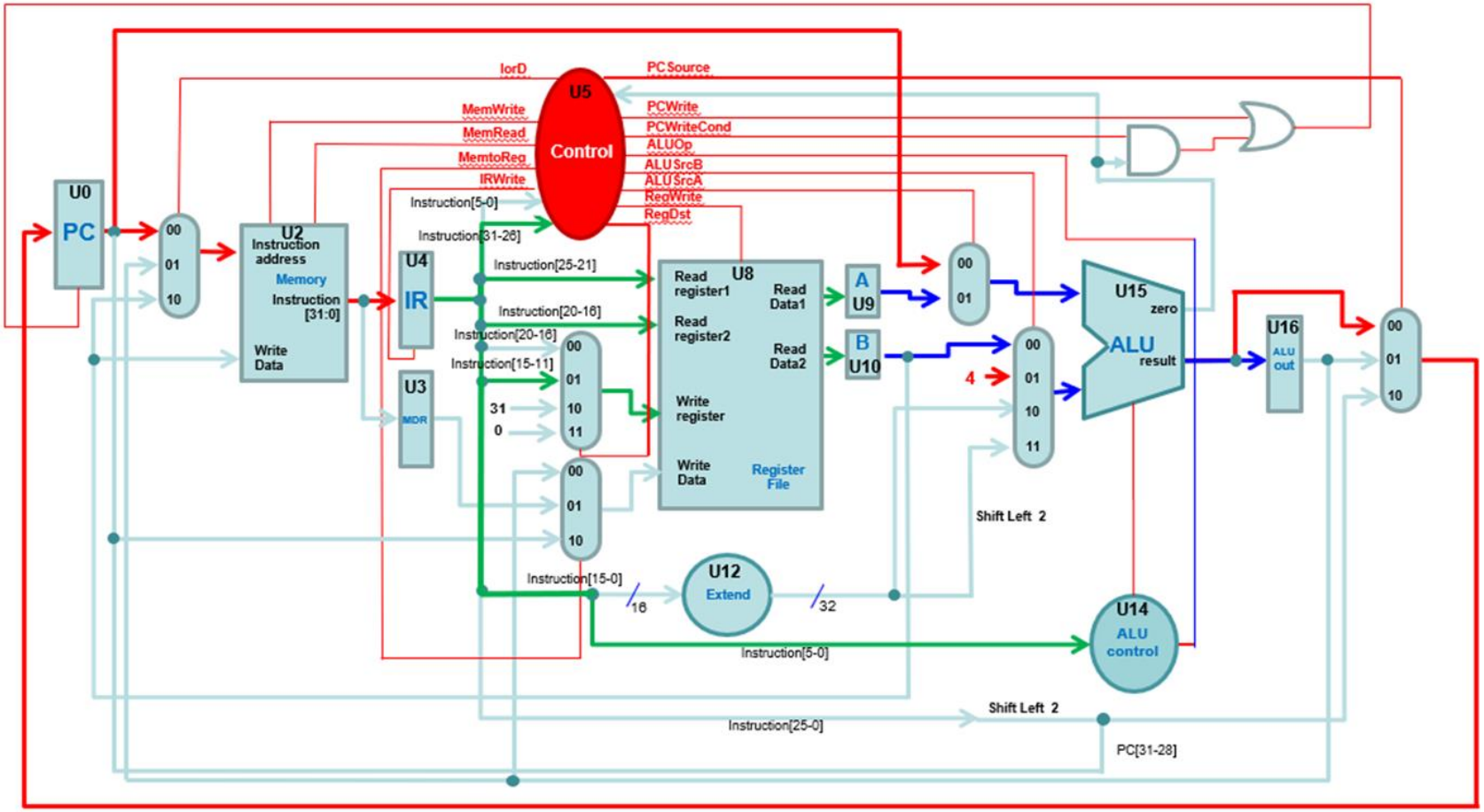
R-type(S1: ID(2))

1	A = Reg [IR[25:21]] B = Reg [IR[20:16]] ALUOut = PC + (Sign-Extended (IR[15:0]) << 2)	Instr Decode & Register Fetch



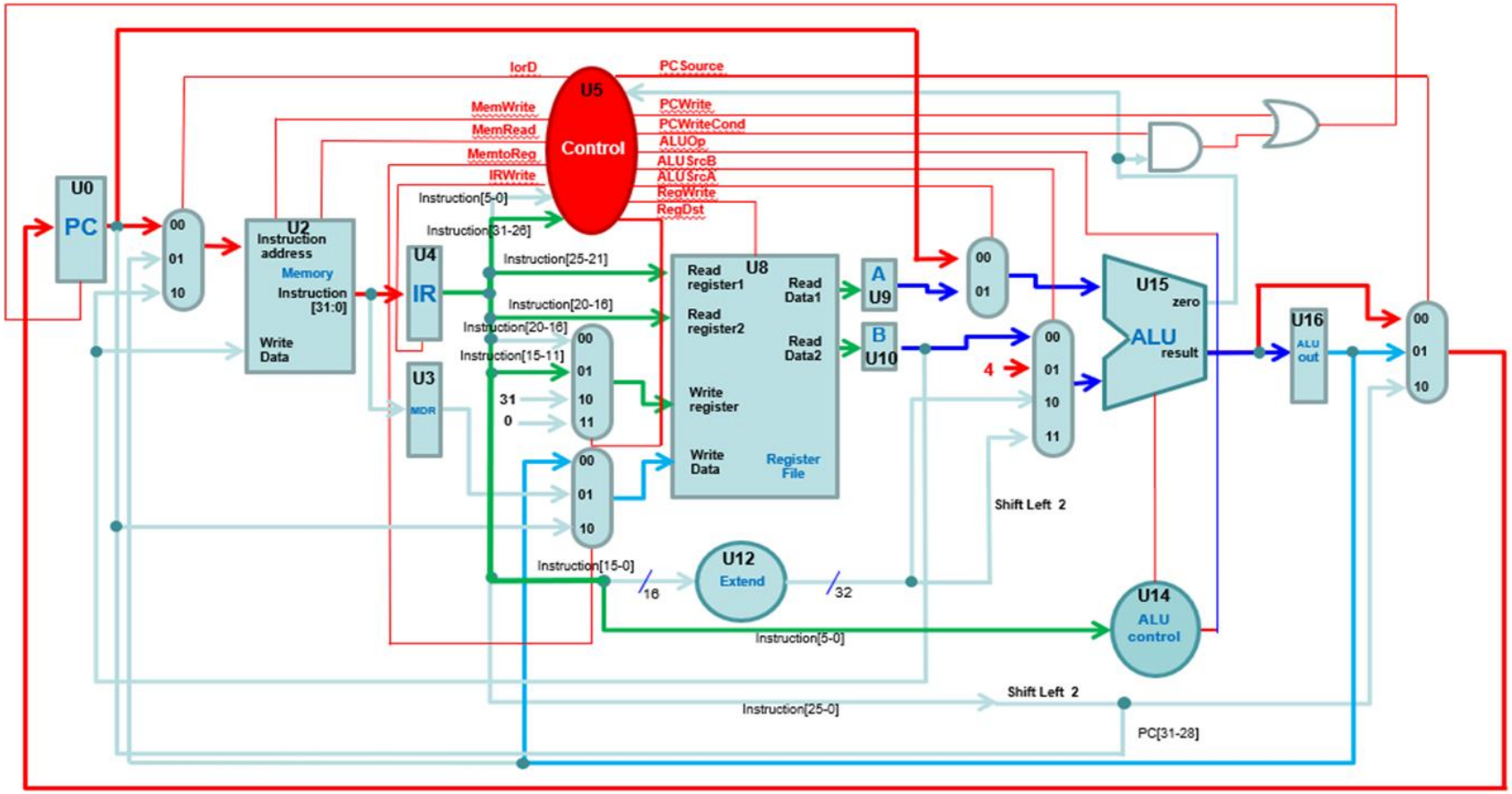
R-type(S6: EX(3))

2	ALUOut = A op B	ALUOut = A + Sign-Extended (IR[15:0])	If (A==B) then PC = ALUOut	PC = { PC[31:28], (IR[25:0]<<2) }	Execution or Branch/Jump Completion



R-type(S7: WB(4))

3	Reg [IR[15:11]] = ALUOut	Load: MDR = Memory[ALUOut] Store: Memory [ALUOut] = B			Memory Access & R-type Completion



Instance name of top module

Instance name	Description
U0_PC	Program counter
U1_MEM	Memory (IM+DM)
U2_IR	Instruction Register
U3_MDR	Memory data register
U4_FSM	Finite State Machine Main Controller
U5_RF	Register File
U6_A	Temporary register for Read data1
U7_B	Temporary register for Read data2
U8_SEU	Sign Extend Unit
U9_ALU	Arithmetic Logical Unit
U10_MUL	Multiplier Unit
U11_ALUO	Temporary register for ALU result

ROM_MICRO.txt

Port name	Classification	Bit	Description
IorD	Output	1-bit	Memory Access for Instruction or Data
MemRead	Output	1-bit	Memory Read Enable Signal
MemWrite	Output	1-bit	Memory Write Enable Signal
DataWidth	Output	3-bit	Memory Data Control signal
IRwrite	Output	1-bit	Instruction Register Write Enable signal
RegDst	Output	2-bit	Register Destination Selection
RegDatSel	Output	3-bit	Register Write Data Selection
RegWrite	Output	1-bit	Register Write Enable signal
ExtMode	Output	1-bit	Extender Control signal
ALUsrcA	Output	3-bit	ALU input A Selection
ALUsrcB	Output	3-bit	ALU input B Selection
ALUop	Output	5-bit	ALU Operation Control signal
ALUctrl	Output	2-bit	ALU Control signal
Branch	Output	3-bit	Branch Address Selection signal
PCsource	Output	2-bit	Next PC Selection
PCwrite	Output	1-bit	PC Write Enable signal
StateSel	Output	2-bit	Next FSM State Number Selection

*** There are an 8-bit reserved field between PCwrite and StateSel. Set them as xxxxxxxx.**

ROM_MICRO.txt

```
0_1_0_000_1_xx_xxx_0_x_011_001_00100_00_000_00_1_xxxxxxxx_11  // 0x00:  FETCH
x_x_0_xxx_0_xx_xxx_0_1_011_100_00100_00_xxx_xx_0_xxxxxxxx_01  // 0x01:  DECODE/REG_READ/BRANCH_ADDR
x_x_x_xxx_x_xx_xxx_x_x_xxx_xxx_xxxxx_xx_xxx_xx_x_xxxxxxxx_xx  // 0x02:
x_x_x_xxx_x_xx_xxx_x_x_xxx_xxx_xxxxx_xx_xxx_xx_x_xxxxxxxx_xx  // 0x03:
x_x_x_xxx_x_xx_xxx_x_x_xxx_xxx_xxxxx_xx_xxx_xx_x_xxxxxxxx_xx  // 0x04:
x_x_x_xxx_x_xx_xxx_x_x_xxx_xxx_xxxxx_xx_xxx_xx_x_xxxxxxxx_xx  // 0x05:
x_x_0_xxx_0_xx_xxx_0_x_000_000_01111_00_xxx_xx_0_xxxxxxxx_11  // 0x06:  sra   $d = $t >>> a
x_x_0_xxx_0_01_000_1_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_00  // 0x07:
```

ROM_DISP.txt

Port name	Classification	Bit	Description
undef	Input	1-bit	Undefined Instruction
raddr	Input	8-bit	Address for Microprogram ROM

```
1_xxxxxxxx  // OP 111001
1_xxxxxxxx  // OP 111010
1_xxxxxxxx  // OP 111011
1_xxxxxxxx  // OP 111100
1_xxxxxxxx  // OP 111101
1_xxxxxxxx  // OP 111110
1_xxxxxxxx  // OP 111111
1_xxxxxxxx  // FN 000000  sll
1_xxxxxxxx  // FN 000001
1_xxxxxxxx  // FN 000010  srl
0_00000110  // FN 000011  sra
```

Change 1 -> 0 undef signal,
enter address for ROM_MICRO.txt
0x06

M_TEXT_SEG.txt

M_TEXT_SEG.txt - Windows 메모장

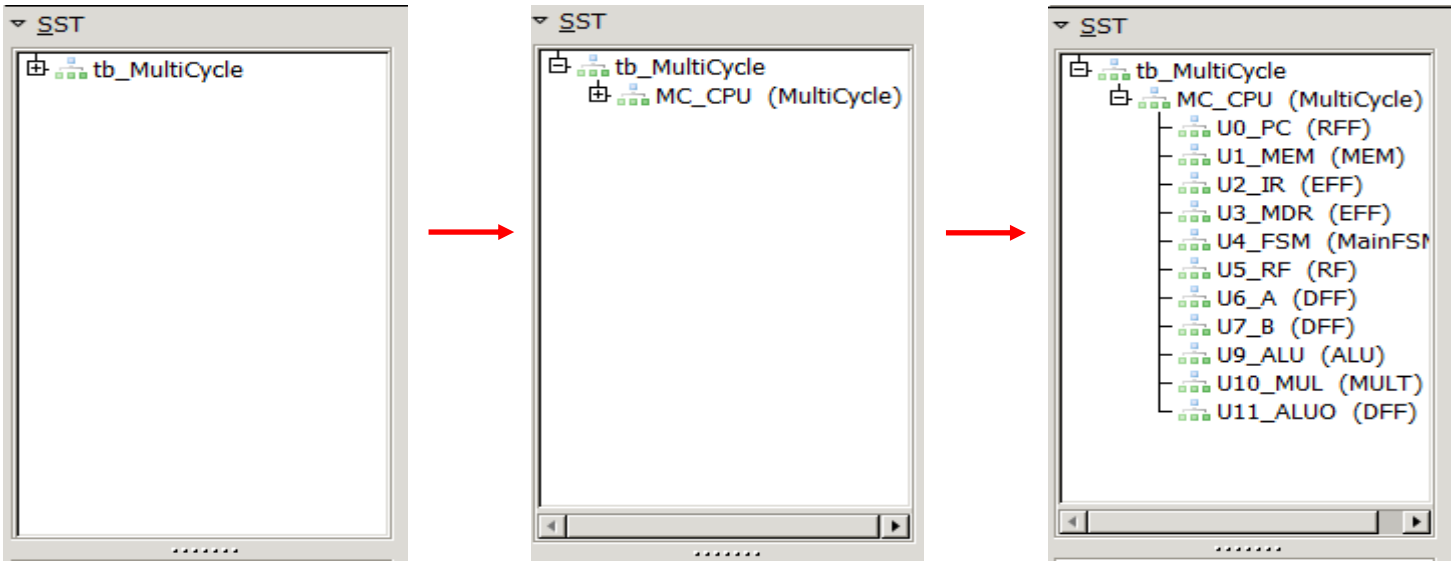
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

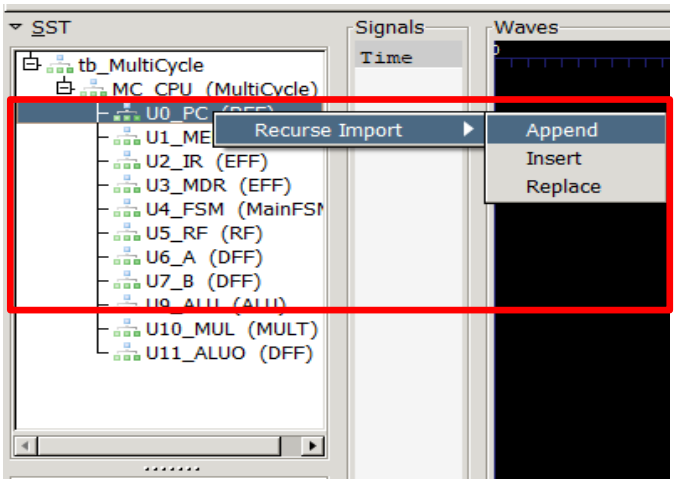
```
001111_00000_00010_0001_0010_0011_0100 //lui $0 $2 0x1234
001101_00010_00011_0101_0110_0111_1000 //ori $2 $3 0x5678
001111_00000_00100_0001_0001_0010_0010 //lui $0 $4 0x1122
001101_00100_00101_0011_0011_0100_0100 //ori $4 $5 0x3344

000000_00011_00101_00110_00000_100011 //subu $3 $5 $6
```

*You can change underbar(_) position for convenience

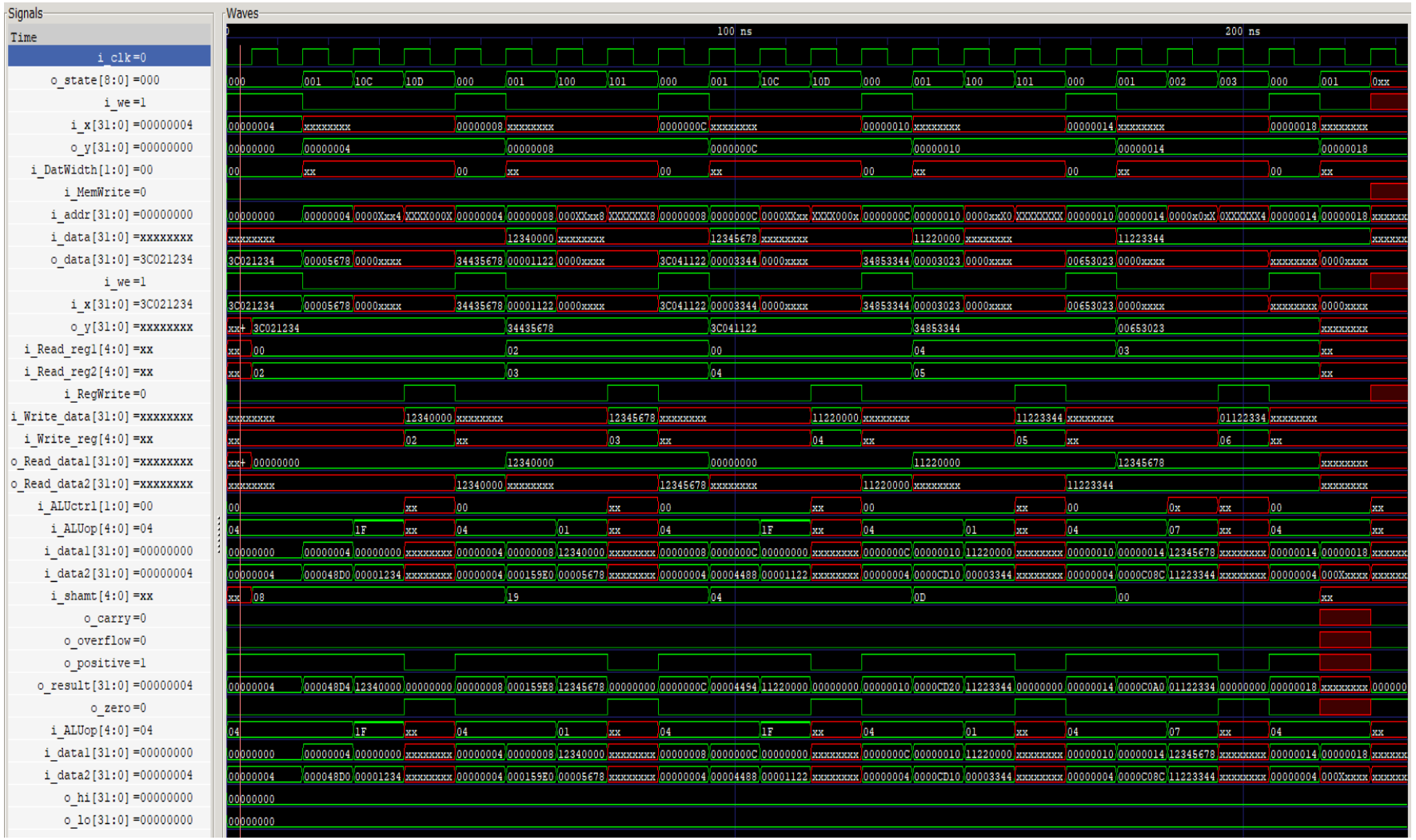
R-Type:	op	$rs	$rt	$rd	shamt	func
I-Type:	op	$rs	$rt	imm16		
J-Type:	op	imm26				

Simulation


SST
tb_MultiCycle
SST
tb_MultiCycle
MC_CPU (MultiCycle)
SST
tb_MultiCycle
MC_CPU (MultiCycle)
U0_PC (RFF)
U1_MEM (MEM)
U2_IR (EFF)
U3_MDR (EFF)
U4_FSM (MainFSM
U5_RF (RF)
U6_A (DFF)
U7_B (DFF)
U9_ALU (ALU)
U10_MUL (MULT)
U11_ALUO (DFF)


SST
Signals
Waves
Time
tb_MultiCycle
MC_CPU (MultiCycle)
U0_PC
U1_ME
Recurse Import
Append
Insert
Replace
U2_IR (EFF)
U3_MDR (EFF)
U4_FSM (MainFSM
U5_RF (RF)
U6_A (DFF)
U7_B (DFF)
U9_ALU (ALU)
U10_MUL (MULT)
U11_ALUO (DFF)

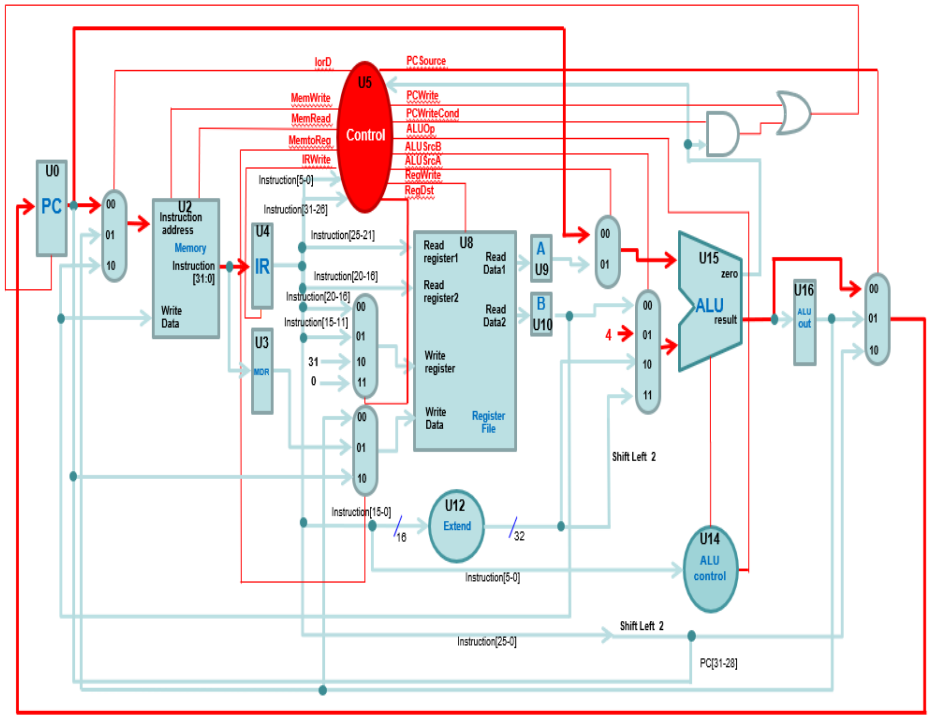
Simulation

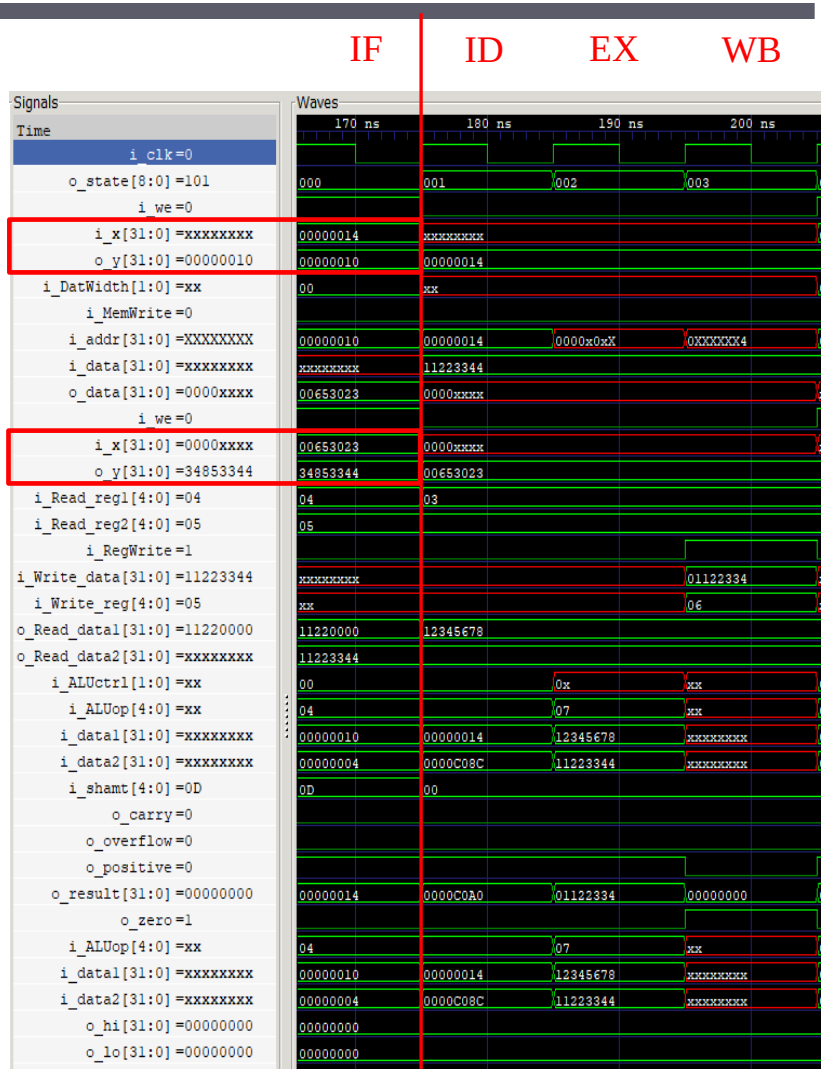


MPLAB
Media Processing Laboratory

Instruction Fetch (subu $3 $5 $6)

0	IR = Memory[PC] PC = PC + 4	Instr Fetch

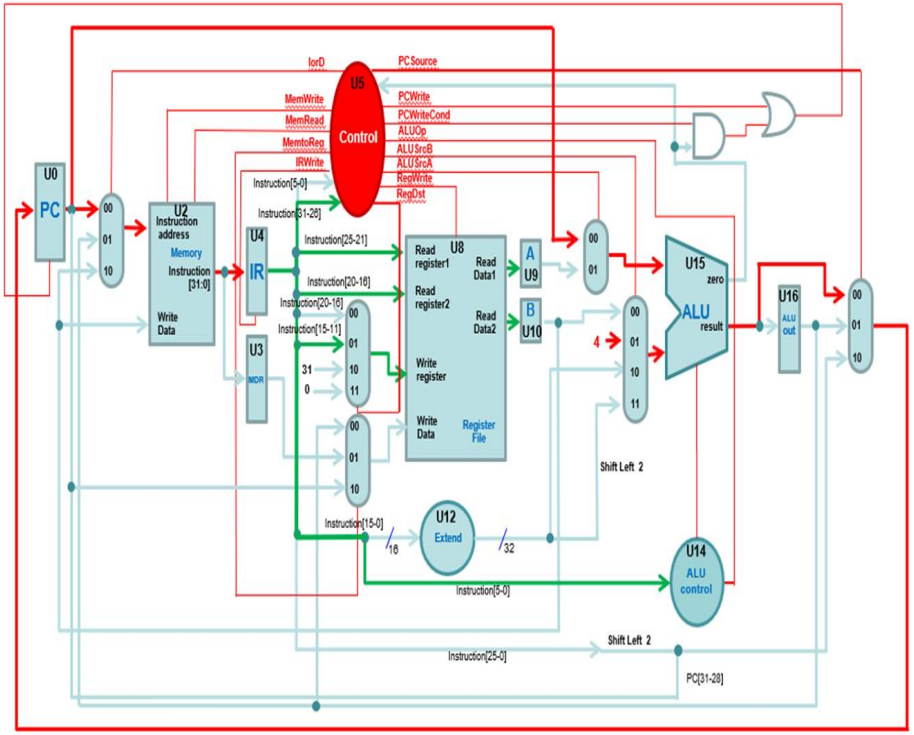




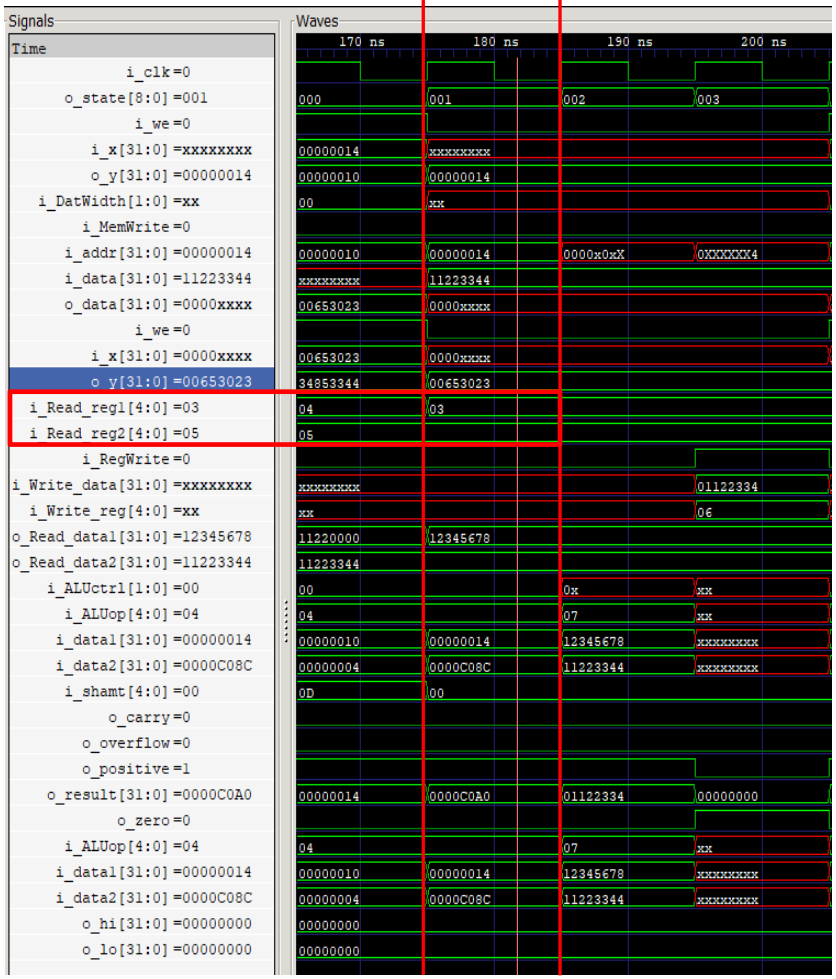
MPLAB
Media Processing Laboratory

Instruction Decode (subu $3 $5 $6)

1	A = Reg [IR[25:21]] B = Reg [IR[20:16]] ALUOut = PC + (Sign-Extended (IR[15:0]) << 2)	Instr Decode & Register Fetch



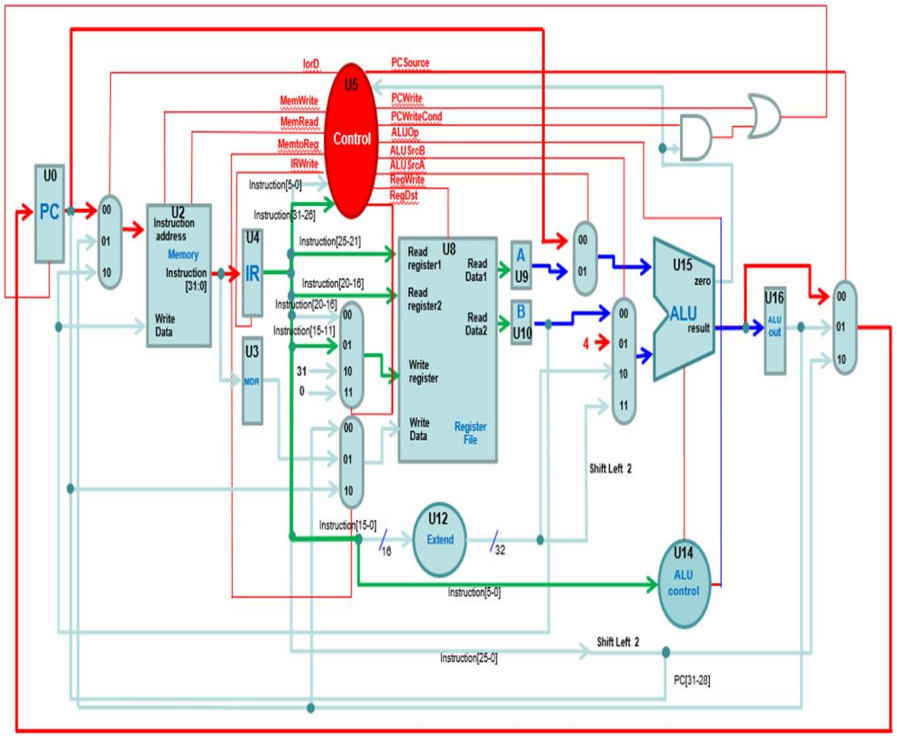
IF ID EX WB

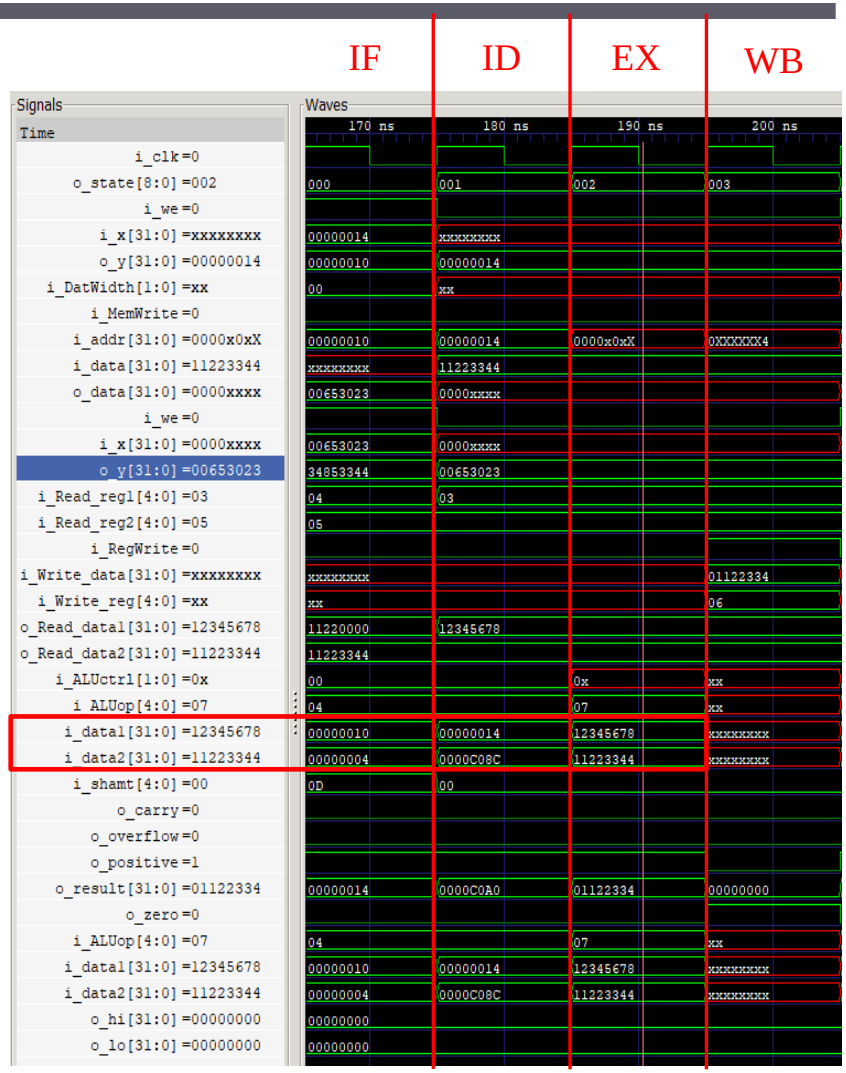


MPLAB
Media Processing Laboratory

Execution (subu $3 $5 $6)

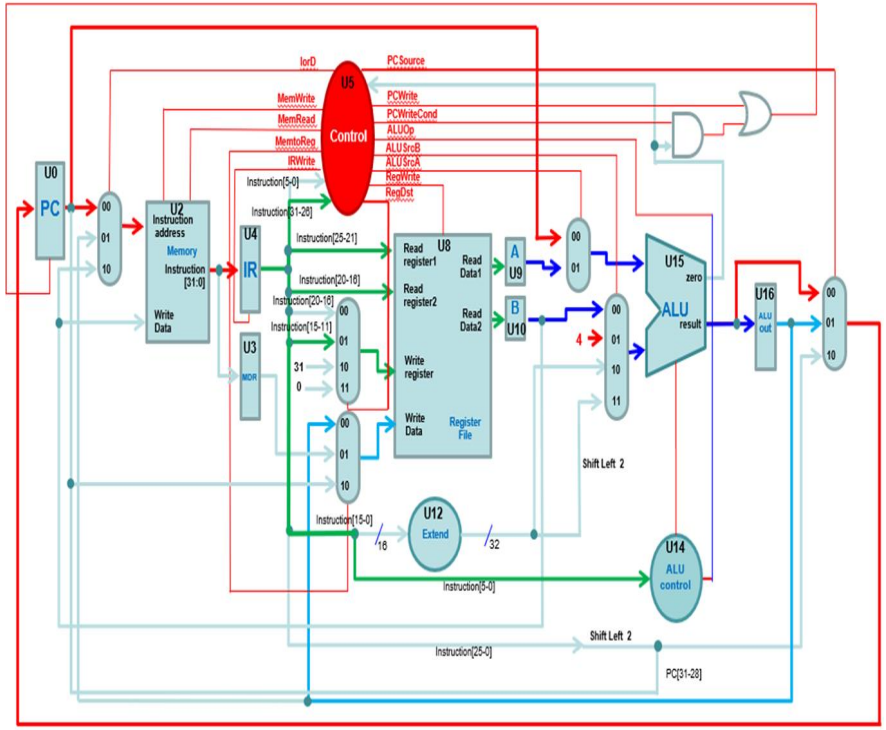
2	ALUOut = A op B	ALUOut = A + Sign-Extended (IR[15:0])	If (A==B) then PC = ALUOut	PC = { PC[31:28], (IR[25:0]<<2) }	Execution or Branch/Jump Completion

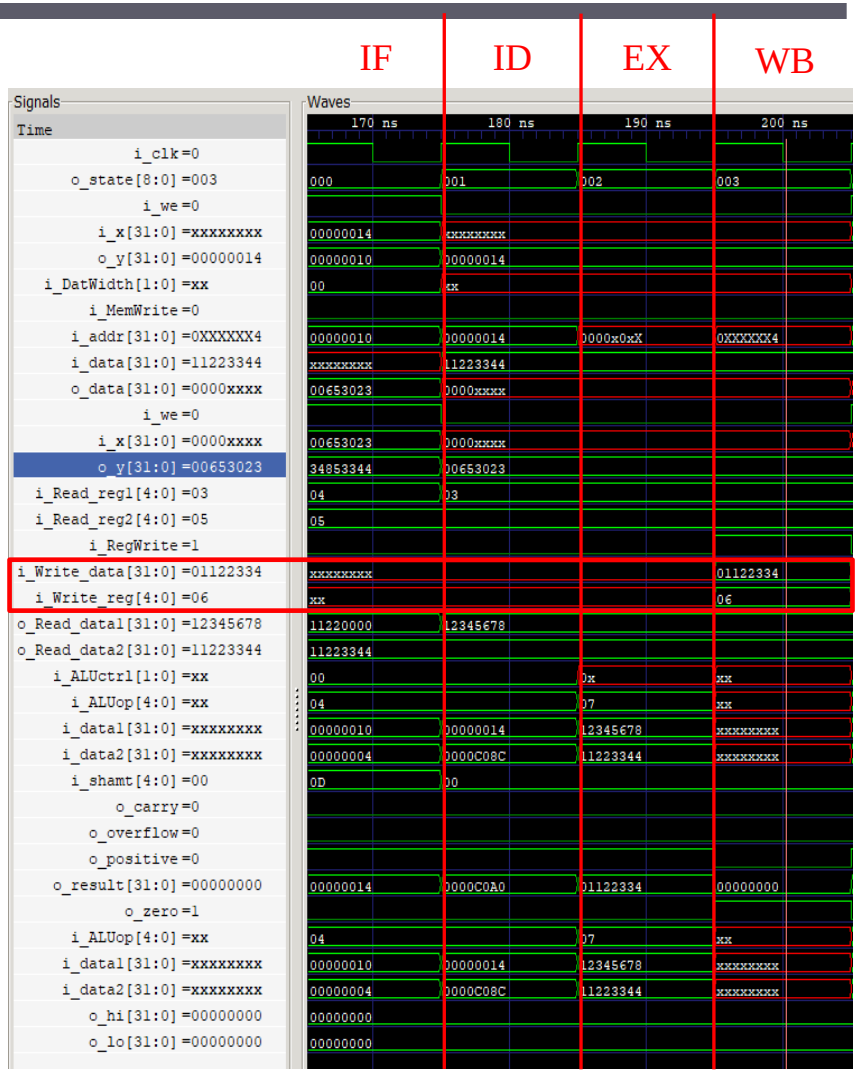




Write Back (subu $3 $5 $6)

3	Reg [IR[15:11]] = ALUOut	Load: MDR = Memory[ALUOut] Store: Memory [ALUOut] = B			Memory Access & R-type Completion





MPLAB
Media Processing Laboratory

Thank You

Multi-Cycle CPU FSM Diagram