

单周期CPU设计

单时钟周期的CPU设计简图

每个框一个module

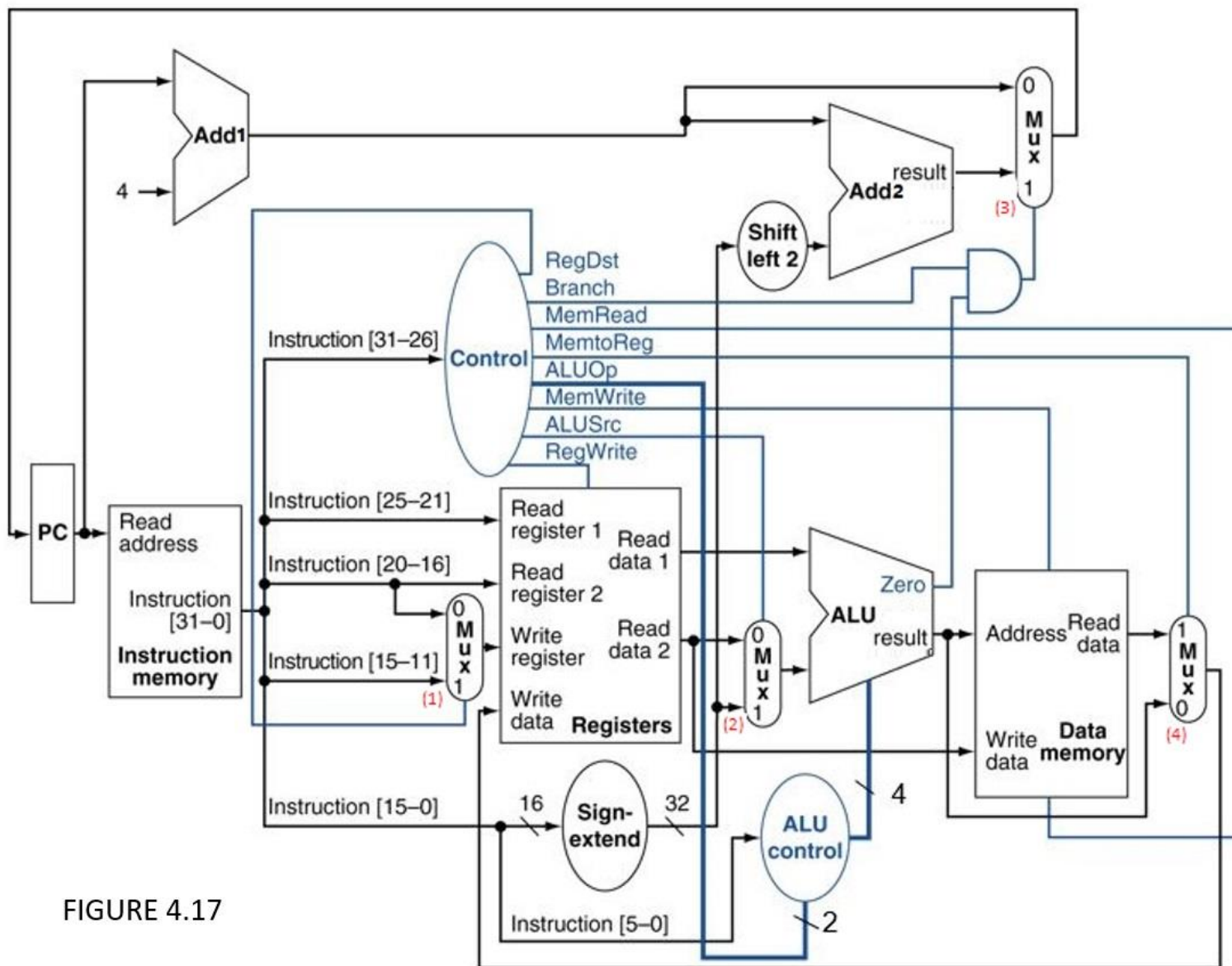


FIGURE 4.17

单时钟周期 -- Jump的控制信号和数据通路

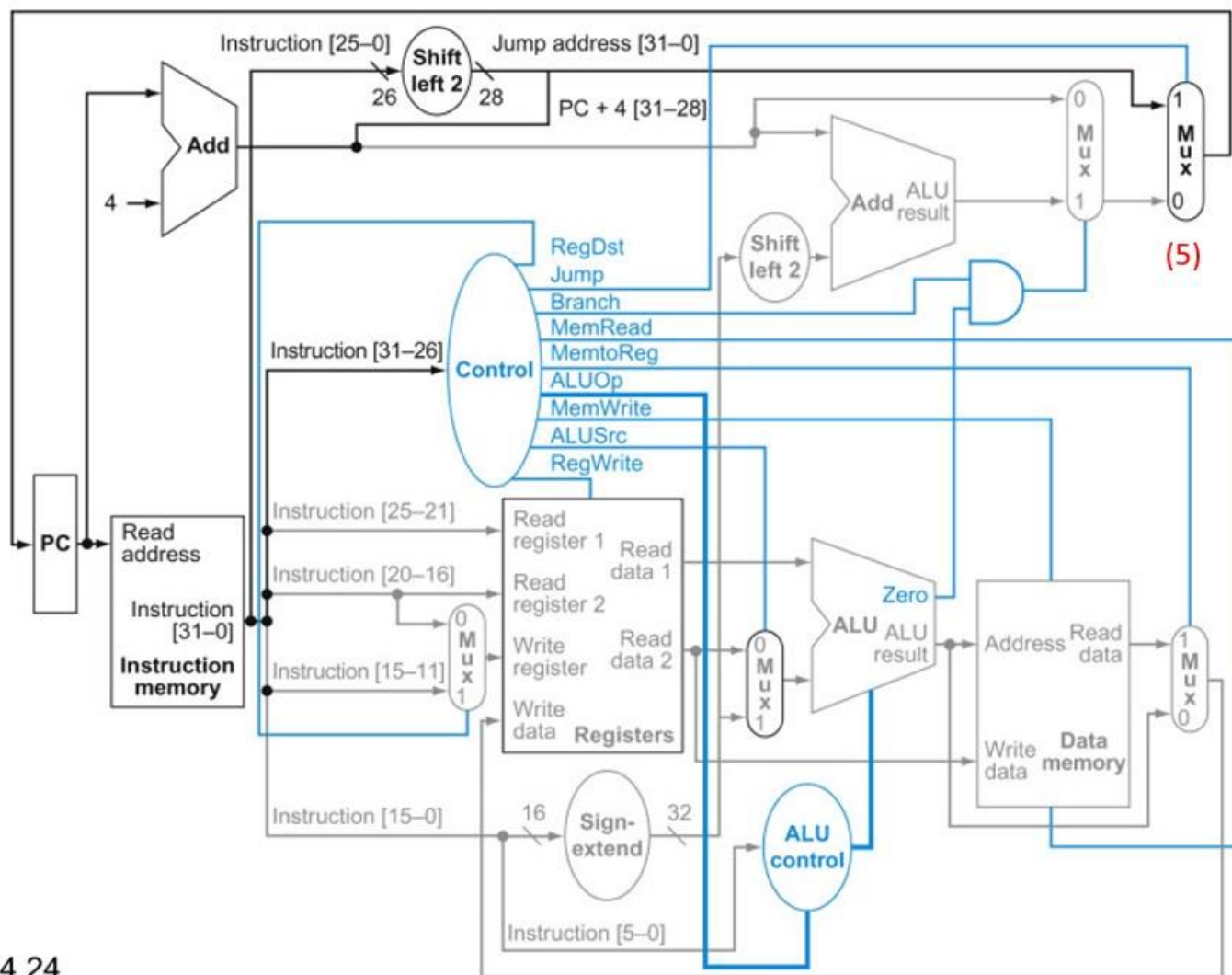



FIGURE 4.24

ALUOp和ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10  3位	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

注意事项

- 时序逻辑电路(PC、寄存器、数据存储器)的always使用posedge:
always @(posedge clk)
- 组合逻辑电路(寄存器)的always不使用posedge:
always @(clk)
- 时序逻辑电路一般采用非阻塞赋值: <=
- 组合逻辑电路一般采用阻塞赋值: =
- 要在always之外使用assign
assign PC4 = PC4 + offset;
- 存储器定义:
reg[31:0] mem[255:0];
- 先做模拟, 再下载到板
- 如果结果有问题, 可以反推方式检查上一级的输出, 可以仿真单个模进行检查
- 从PC加4开始设计

- 模块定义:

```
module Registers(reg, regRead, readData);
```

```
    input reg;
```

```
    input regRead;
```

```
    output readData;
```

```
    wire[5:0] reg;
```

```
    wire regRead;
```

```
    reg[31:0] readData;
```

```
    ...
```

```
endmodule
```

```
module Registers(reg, regRead, readData);
```

```
    input wire[5:0] reg;
```

```
    input regRead; ————— 可以省略wire
```

```
    output reg[31:0] readData;
```

```
    ...
```

```
endmodule
```

```
module Registers(input wire[5:0] reg, input wire regRead, output reg[31:0] readData);
```

```
    ...
```

```
endmodule
```

```
module Registers(input [5:0] reg, input regRead, output reg[31:0] readData);
```

```
    ...
```

```
endmodule
```

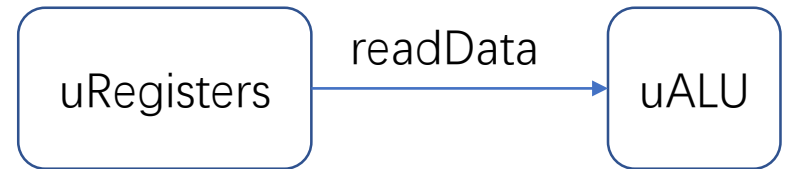
可以省略wire

- 模块连线:

```
wire[5:0] reg;  
wire regRead;  
wire[31:0] readData, data1;  
assign readData = data1;  
Registers uRegisters(reg, regRead, readData);  
ALU uALU(data1, data2, result);
```

或

```
Registers uRegisters(reg, regRead, readData);  
ALU uALU(readData, data2, result);
```



- Mux:

```
module Mux1(input sel,input[4:0] rt, input[4:0] rd,output [4:0] rg);  
    assign rg = (sel==0)?rt:rd;  
endmodule
```

- RTL ANALYSIS:

RTL ANALYSIS

Open Elaborated Design

- Report Methodology
- Report DRC
- Report Noise
- Schematic

(1)

ELABORATED DESIGN - xc7a35tcbg236-1 (active)

Elaborated Design is out-of-date. Design sources were modified. [details](#) [Reload](#)

Sources Netlist x ? _ □ ↗

- CPU
 - Nets (392)
 - Leaf Cells (18)
 - uALU (ALU)
 - uALUControl (ALUcontrol)
 - uControl (Control)
 - uDataMemory (DataMemory)
 - uInstructionMemory (InstructionMemory)
 - uMux1 (Mux1)
 - uMux2 (Mux2)
 - uMux4 (Mux4)
 - uPC (PC)

Source File Properties ? _ □ ↗

Mux1.v

Enabled

Location: C:\vivado2\cpu1\cpu1.srcs/sources_1/new

Type: Verilog

Library: xil_defaultlib

General Properties

(2)

Project Summary x Schematic x Schematic (2) x

29 Cells 17 I/O Ports

clk

uDataMemory

addr[31:0]

clk

memRead

memWrite

writeData[31:0]

DataMemory

readData[31:0]

uMux1

rd[4:0]

rt[4:0]

sel

Mux1

uMux4

aluResult[31:0]

memToReg

readData[31:0]

Mux4

uSignExt

lineIn[15:0]

SignExt

- 调试:

结果: q[15:0]-writeData
 仿真: always #5 clk=~clk;
 实现:q~led, clk-left button
 (也可以把结果q输出到数码管)

```
module CPU(
  input clk,
  output wire[15:0] q
);
  wire[31:0] pc;
  wire[31:0] ins;
  wire[31:0] immOrOffset;
  wire RegDst, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
  wire[31:0] writeData, readData1, readData2, operand2, aluResult;
  wire Zero, Branch, Jump;
  wire[2:0] ALUOp;
  wire[4:0] rg;
  wire[3:0] aluControl;
  wire[31:0] readData;
  wire[31:0] branchOffset;

  assign q[7:0] = pc[7:0];
  assign q[15:8] = writeData[7:0];

  PC uPC(clk, Branch, Zero, branchOffset, Jump, ins[25:0], pc);
  InstructionMemory uInstructionMemory(pc, ins);
  Control uControl(clk, ins[31:26], ins[5:0], RegDst, Branch,
    MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite, Jump);
  Mux1 uMux1(RegDst, ins[20:16], ins[15:11], rg);
  Registers uRegisters(clk, ins[25:21], ins[20:16], rg, writeData, RegWrite, readData1, readData2);
  SignExtend uSignExtend(ins[15:0], immOrOffset);
  Mux2 uMux2(ALUSrc, readData2, immOrOffset, operand2);
  ALUControl uALUControl(ins[5:0], ALUOp, aluControl);
  ALU uALU(readData1, operand2, aluControl, aluResult, Zero);
  DataMemory uDataMemory(.clk(clk), .addr(aluResult), .writeData(readData2),
    .memWrite(MemWrite), .memRead(MemRead), .readData(readData));
  Mux4 uMux4(MemtoReg, aluResult, readData, writeData);
  ShiftLeft2Addr uShiftLeft2Addr(immOrOffset, branchOffset);
endmodule
```

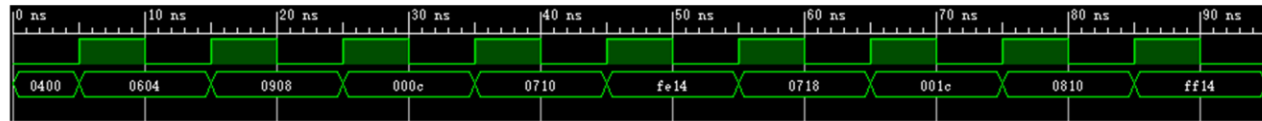
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF] (去抖动)

- 指令存储器预存的指令：

```

    addi $t0,$zero,4
    lw $s1,0($t0)
    lw $s2,4($t0)
    add $s3,$zero,$zero
loop:
    addi $s1,$s1,1
    beq $s1,$s2,exit
    add $s3,$s3,$s1
    j loop
exit:
    sub $s3,$s3,$s2
    sw $s3,8($t0)
ll:
    lw $s3,8($t0)
    j ll

```



(只有部分指令，有些指令的 writeData 可能不同)。

截屏（可以多幅，截到最后的 ll 循环执行 3 次）：

- 预存数据（地址分别为4和8）：

```

32' h00000006
32' h00000009

```

(选做) 在上一步的基础上增加所需指令(slt,slti,sll,jal,jr), 实现冒泡排序算法。

对8个数进行排序, 排序结束后按最左边按钮, 每按一次在数码管上显示结果的1个数。

截止日期：2020年12月5日23: 00（周六）