计算机系统结构实验Lab03:
单周期CPU部件设计（一）

常烁晨 521021910369

2023.3.31

摘要

lab03 的实验目标是完成第一部分类MIPS单周期处理器功能部件的设计。在本实验中需要设计的三个功能部件分别是主控制器部件（Ctr）、ALU控制器部件（ALUCtr）、ALU部件。在上一个实验 lab02 中，我们完成了四位加法器的设计与实现，此部分则是直接利用模块编程，完成CPU中硬件的编写。

从本实验开始，实验说明书将不会给出完整的实验代码，因此实验所需的代码需要自己根据所学知识（如MIPS指令的编码逻辑、ALU运算单元的工作模式）等来自己编写。此外，从本实验开始的四个实验都没有必做的上板验证要求，因此实验报告只展示源代码和激励文件的编写。

目录

1、实验目的

（1）首先理解单周期CPU的主控部件、ALU控制单元以及ALU单元的原理，了解MIPS指令的编码格式。

（2）分别完成主控制器部件（Ctr）、ALU控制器部件（ALUCtr）、ALU部件模块的编写。

（3）使用 Vivado ，分别进行三个功能部件的仿真，确保仿真结果与实验报告要求的一致。

2、源代码实现

2.1 Ctr

主控制单元 Ctr 输入为指令的最高六位（opCode）字段，根据32位指令的高六位，可以确定ALUCtr、Data Memory、Registers等部件的工作方式。Ctr的工作方式就是接收指令的高六位作为输入，并输出相关的控制信号给其他单元。

实验指导书给出了Ctr的编码逻辑。

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

图 3. 主控制模块的真值表

其中，Jump 指令的高六位编码是 000010，此时控制器发出的 Jump 信号为 1，其余的输出都是 0。下面用 Verilog 描述真值表， Ctr 模块接收输入，根据我们描述的真值表，就可以得出对应的输出信号。

首先展示模块的输入输出信号如下。

```
`timescale 1ns / 1ps

module Ctr(
    input [5:0] opCode, output regDst, output aluSrc, output memToReg,
    output regWrite, output memRead, output memWrite, output branch,
    output [1:0] aluOp, output jump
    );
    reg RegDst; reg ALUSrc; reg MemToReg; reg RegWrite; reg MemRead;
    reg MemWrite; reg Branch; reg [1:0] ALUOp; reg Jump;
    assign regDst = RegDst; assign aluSrc = ALUSrc;
    assign memToReg = MemToReg; assign regWrite = RegWrite;
    assign memRead = MemRead; assign memWrite = MemWrite;
    assign branch = Branch; assign aluOp = ALUOp; assign jump = Jump;
```

根据实验指导书提供的内容，利用case语句完成真值表的枚举。具体代码如下。由于ALU计算种类比较复杂，因此ALU具体执行的运算需要主控制单元（Ctr）产生一个 ALUOp信号，用来控制ALU控制单元（ALUCtr），ALUCtr 再根据 Ctr 产生的控制信号，以及指令的编码格式，最终确定 ALU 执行的运算方式。

此处 ALUOp 的值需要和后面编写 ALUCtr 源代码时保持一致。由于本实验 ALU 所需要进行的运算有限，因此 ALUOp 可以用 2 位二进制数来保存。在接下来的实验中， ALUOp 的值可能需要扩充。

```
always @ (opCode)
 begin
    case(opCode)
    6’b000000:    begin
       RegDst = 1; ALUSrc = 0; MemToReg = 0; RegWrite = 1; MemRead = 0;
       MemWrite = 0; Branch = 0; ALUOp = 2’b10; Jump = 0;        end
    6’b100011:    begin
       RegDst = 0; ALUSrc = 1; MemToReg = 1; RegWrite = 1; MemRead = 1;
       MemWrite = 0; Branch = 0; ALUOp = 2’b00; Jump = 0;        end
    6'b101011:    begin
       RegDst = 0; ALUSrc = 1; MemToReg = 0; RegWrite = 0; MemRead = 0;
       MemWrite = 1; Branch = 0; ALUOp = 2’b00; Jump = 0;        end
    6'b000100:    begin
       RegDst = 0; ALUSrc = 0; MemToReg = 0; RegWrite = 0; MemRead = 0;
       MemWrite = 0; Branch = 1; ALUOp = 2’b01; Jump = 0;        end
    6'b000010:    begin
       RegDst = 0; ALUSrc = 0; MemToReg = 0; RegWrite = 0; MemRead = 0;
       MemWrite = 0; Branch = 0; ALUOp = 2’b00; Jump = 1;        end
    default:          begin
       RegDst = 0;  ALUSrc = 0; MemToReg = 0; RegWrite = 0; MemRead = 0;
       MemWrite = 0; Branch = 0; ALUOp = 2’b00; Jump = 0;        end
    endcase
 end
endmodule
```

根据提供的真值表，依次编码 Ctr 的控制信号即可。

2.2 ALUCtr

算术逻辑单元控制器（ALUCtr）根据主控制器 Ctr 产生的控制信号 ALUOp 以及指令编码的低6位，来产生 ALU 的控制信号。其中本实验中 ALU

的操作共有 and、or、add、sub、set、nor 六种。实验指导书中给出了 ALUCtr 的真值表格式。

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

图 5. ALU 控制单元输入输出真值表

根据真值表，类似上面 Ctr 源代码的编写方式，编写 ALUCtr 源代码。首先是模块相关的输入输出信号的定义。

```
`timescale 1ns / 1ps

module ALUCtr(
    input [1:0] aluOp,
    input [5:0] funct,
    output [3:0] aluCtrOut
    );

    reg [3:0] ALUCtrOut;
    assign aluCtrOut = ALUCtrOut;
```

这里的 aluCtrOut 就是该算数单元控制器发送给 ALU 的控制信号，一个信号唯一对应 ALU 单元的一种运算。

接下来是真值表的逻辑描述部分。

```
always @ (aluOp or funct)
begin
    casex({aluOp, funct})
        8'b00xxxxxx:
            ALUCtrOut = 4'b0010;
        8'b01xxxxxx:
            ALUCtrOut = 4'b0110;
        8'b10xx0000:
            ALUCtrOut = 4'b0010;
        8'b1xxx0010:
            ALUCtrOut = 4'b0110;
        8'b1xxx0100:
            ALUCtrOut = 4'b0000;
        8'b1xxx0101:
            ALUCtrOut = 4'b0001;
        8'b1xxx1010:
            ALUCtrOut = 4'b0111;
    endcase
end
endmodule
```

可以看出，根据 aluOp 的值，和指令低6位的 funct 值，可以对应出 aluCtrOut 的值。上面的代码完全是实验指导书对应的逻辑翻译。

2.3 ALU

算术逻辑单元 ALU 根据 ALUCtr 产生的控制信号，对相应输入的操作数进行运算，其中 Zero 位用于确定运算结果是否为 0 。ALU 的运算结果通过 ALURes 来输出。

以下展示 ALU 模块的的源代码编写。

```
`timescale 1ns / 1ps
module ALU(
    input [3 : 0] aluCtrOut, input [31 : 0] inputA, input [31 : 0] inputB,
    output [31 : 0] aluRes, output zero );

    reg Zero; reg [31 : 0] ALURes;
    assign zero = Zero; assign aluRes = ALURes;

    always @ (inputA or inputB or aluCtrOut)
    begin
        case(aluCtrOut)
        4'b0000:
            ALURes = inputA & inputB;
        4'b0001:
            ALURes = inputA | inputB;
        4'b0010:
            ALURes = inputA + inputB;
        4'b0110:
            ALURes = inputA - inputB;
        4'b0111:
            ALURes = inputA < inputB;
        4'b1100:
            ALURes = ~(inputA | inputB);
        endcase
        if(ALURes == 0)
            Zero = 1;
        else
            Zero = 0;
    end
```

ALU 的运算逻辑符合实验指导书中提供的表格。

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

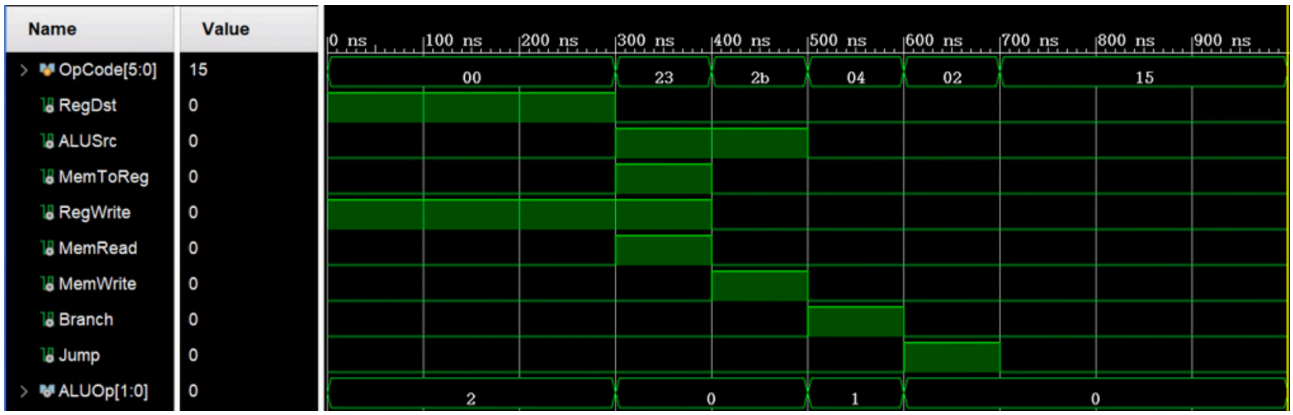
3、仿真实现

3.1 Ctr_tb

创建并编写激励文件，目标是检查输入各种指令时， Ctr 是否可以输出正确的控制信号。激励文件的代码如下所示。

```
`timescale 1ns / 1ps
module Ctr_tb();
    reg [5:0] OpCode; wire RegDst; wire ALUSrc; wire MemToReg;
    wire RegWrite; wire MemRead; wire MemWrite; wire Branch;
    wire Jump; wire [1:0] ALUOp;
    Ctr u0(
        .opCode(OpCode), .regDst(RegDst), .aluSrc(ALUSrc), .memToReg(MemTo
Reg), .regWrite(RegWrite), .memRead(MemRead), .memWrite(MemWrite),
.branch(Branch), .aluOp(ALUOp), .jump(Jump) );
    initial begin OpCode = 0;
        #100;
        #100 OpCode = 6'b000000; #100 OpCode = 6'b100011;
        #100 OpCode = 6'b101011; #100 OpCode = 6'b000100;
        #100 OpCode = 6'b000010; #100 OpCode = 6'b010101;
    end
endmodule
```

上述激励代码表示经过 100 ps的初始化后，每经过 100 ps，输入一条新的指令。观察对应的输出控制信号是否与实验报告的仿真实例相同。



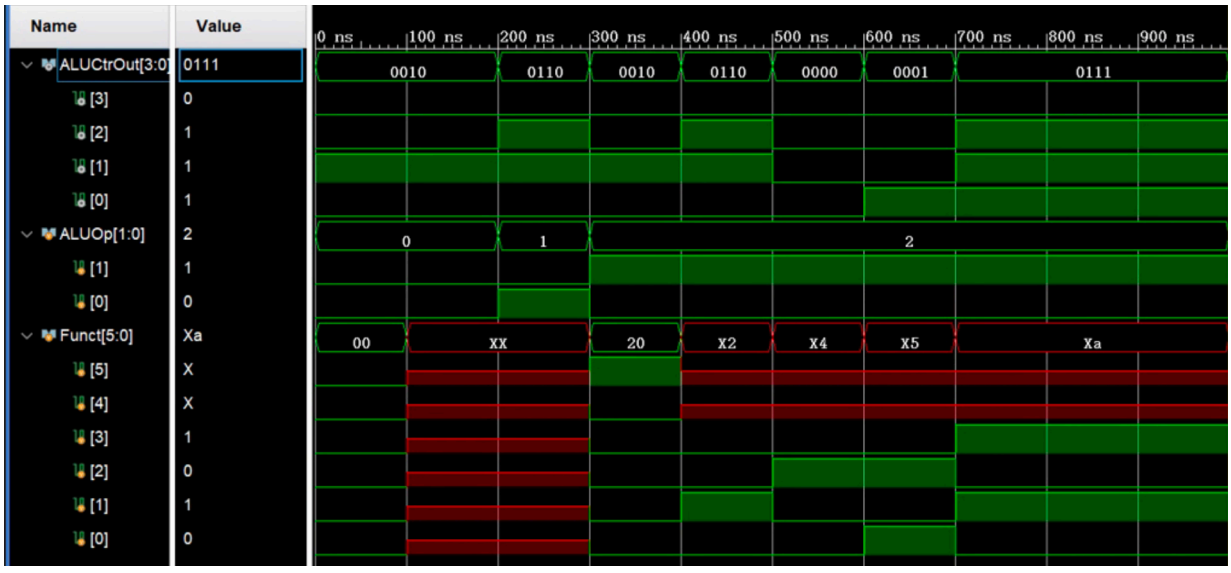
3.2 ALUCtr_tb

创建新的仿真激励文件，并设置为顶层（每次运行仿真都需要将对应的激励代码设为顶层），激励代码如下。

```
`timescale 1ns / 1ps
module ALUCtr_tb();
    wire [3:0] ALUCtrOut; reg [1:0] ALUOp; reg [5:0] Funct;
    ALUCtr u0(
        .aluOp(ALUOp), .funct(Funct), .aluCtrOut(ALUCtrOut) );

    initial begin
        ALUOp = 0; Funct = 0;
        #100;   ALUOp = 2'b00; Funct = 6'bxxxxxx;
        #60;    ALUOp = 2'bx1; Funct = 6'bxxxxxx;
        #60;    ALUOp = 2'b1x; Funct = 6'bxx0000;
        #60;    ALUOp = 2'b1x; Funct = 6'bxx0010;
        #60;    ALUOp = 2'b1x; Funct = 6'bxx0100;
        #60;    ALUOp = 2'b1x; Funct = 6'bxx0101;
        #60;    ALUOp = 2'b1x; Funct = 6'bxx1010;
        #540;   end
endmodule
```

此处可以注意到实验指导书中有两种仿真结果，一种结果的信号有 x 态，另一种没有。从上文的叙述中，我们可以发现，由于此实验中 ALU 的运算种类有限，因此一些指令对应的位置不会影响 ALU 的运算选择，即不会影响 ALUCtr 的取值，此时信号对应的比特电平可以任选高或者低，也即 x 态。



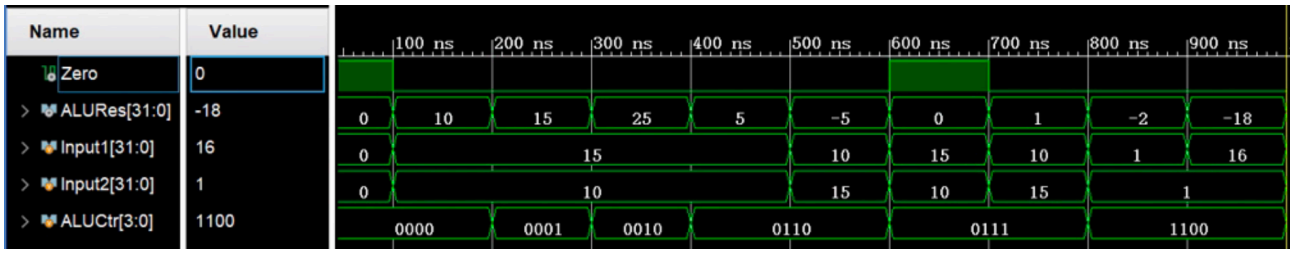
3.3 ALU_tb

新建激励文件，测试 ALU 的功能，激励代码如下。

```
`timescale 1ns / 1ps
module ALU_tb();
    wire Zero; wire [31:0] ALURes;
    reg [31:0] InputA; reg [31:0] InputB; reg [3:0] ALUCtrOut;
    ALU u0(
        .aluRes(ALURes), .inputA(InputA), .inputB(InputB),
        .aluCtrOut(ALUCtrOut), .zero(Zero) );
    initial begin       ALUCtrOut = 0; InputA = 0; InputB = 0;
        #100;           ALUCtrOut = 0'b0000; InputA = 15; InputB = 10;
        #100;           ALUCtrOut = 0'b0001; InputA = 15; InputB = 10;
        #100;           ALUCtrOut = 0'b0010; InputA = 15; InputB = 10;
        #100;           ALUCtrOut = 0'b0110; InputA = 15; InputB = 10;
        #100;           ALUCtrOut = 0'b0110; InputA = 10; InputB = 15;
        #100;           ALUCtrOut = 0'b0111; InputA = 15; InputB = 10;
        #100;           ALUCtrOut = 0'b0111; InputA = 10; InputB = 15;
        #100;           ALUCtrOut = 0'b1100; InputA = 1; InputB = 1;
        #100;           ALUCtrOut = 0'b1100; InputA = 16; InputB = 1;
        #100;           end
    endmodule
```

这段激励代码根据实验指导书的仿真结果编写，用于检测输入各种 ALU 控制信号时，ALU 是否能得出正确运算结果输出。

可以看到仿真结果与指导书一致，证明 ALU 单元运算结果正确。



4、致谢

感谢刘老师以及三位实验助教学长学姐的悉心指导和帮助。

由于本次实验指导书上没有完整代码，因此我在完成实验的过程中出现了许多问题，比如没有正确完成 MIPS 的编码规则设计、在源代码和仿真代码中出现了命名规则不统一等等疏忽，感谢为我提供帮助的助教学长学姐。同时由于上次完成 lab2 上板验证出现的困难，导致我没能提前开始 lab3 的编写，感谢助教学长学姐的帮助，让我能按时在课上完成该实验所要求的内容。

感谢上海交通大学的计算机系统结构实验室，为我们准备了相关的实验环境以及 FPGA 开发板，学校和学院提供的优秀设备让我们受益匪浅。