

组成原理实验课程第 六 次实报告

实验名称	单周期 CPU 实验			班级	张金老师
学生姓名	杨冰雪	学号	2110508	指导老师	董前琨
实验地点	实验楼 A 楼 306		实验时间	2023.6.5	

1、实验目的

1. 理解 MIPS 指令结构，理解 MIPS 指令集中常用指令的功能和编码，学会对这些指令进行归纳分类。
2. 了解熟悉 MIPS 体系的处理器结构，如延迟槽，哈佛结构的概念。
3. 熟悉并掌握单周期 CPU 的原理和设计。
4. 进一步加强运用 verilog 语言进行电路设计的能力。
5. 为后续设计多周期 cpu 的实验打下基础。

2、实验内容说明

1. 熟知 MIPS 指令类型，深入理解常用指令的功能和编码，归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令。
2. 设计本次实验的方案，画出实验方案的设计框图，确定设计中与 FPGA 板上交互的接口，画出包含外围模块的整体设计框图。
3. 编写 verilog 代码，要求针对目前 CPU 可运行的 R 型和 I 型 MIPS 指令，各补充一条新的指令，完成调用单周期 CPU 的外围模块的设计，并编写代码。
4. 对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证。
5. 实验结束后，按照规定的格式完成实验报告的撰写。

3、实验原理图

单周期 CPU 的实现框图如下：

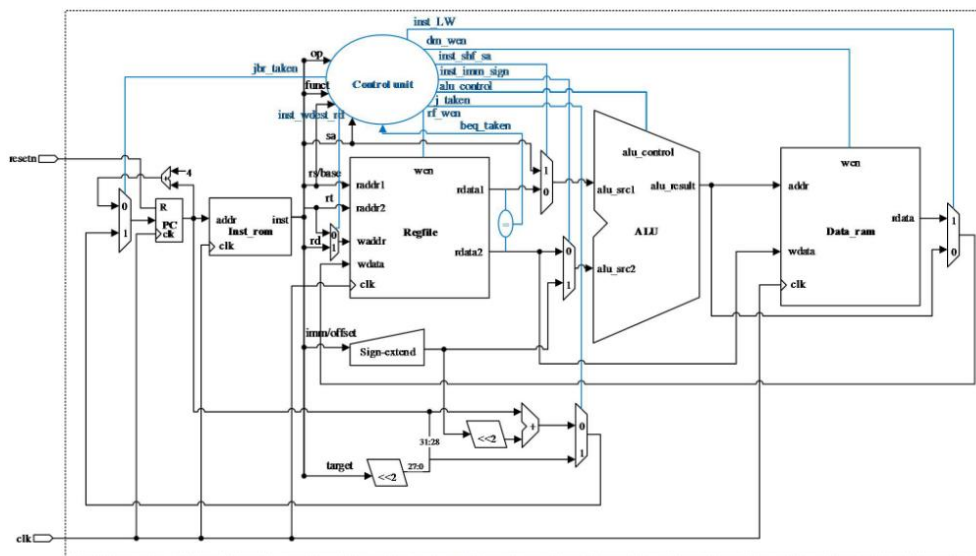


图 7.3 单周期 CPU 的实现框图

4、实验步骤

1. 在 ALU 模块中添加低位加载运算
首先先定义低位加载的信号和其对应的结果：

```
wire alu_hui; //低位加载
```

```
wire [31:0] hui_result;
```

把控制信号传给定义的低位加载信号：

```
assign alu_hui = alu_control[12]; //低位加载
```

添加低位加载具体的运算：

```
assign hui_result = {16'd0, alu_src2[31:16]};
```

在选择相应结果输出中添加低位加载的结果：

```
assign alu_result = (alu_add|alu_sub) ? add_sub_result[31:0] :
    alu_slt      ? slt_result :
    alu_sltu     ? sltu_result :
    alu_and      ? and_result :
    alu_nor      ? nor_result :
    alu_or       ? or_result  :
    alu_xor      ? xor_result :
    alu_sll      ? sll_result :
    alu_srl      ? srl_result :
    alu_sra      ? sra_result :
    alu_lui      ? lui_result :
    alu_hui      ? hui_result :
    32'd0;
```

经过以上过程，成功在 ALU 模块中添加了低位加载这一运算。

2. 在 inst_rom 模块中添加指令

将[19:0]的指令存储器修改为[21:0]:

```
wire [31:0] inst_rom[21:0];
```

为两条新添加的指令添加指令编码：

```
assign inst_rom[19] = 32'h00026883;
assign inst_rom[20] = 32'h2C0E000E;
```

为两条指令添加对应的读指令：

```
5'd19: inst <= inst_rom[19];
5'd20: inst <= inst_rom[20];
```

3. 在 single_cycle_cpu 模块中实现指令

在实现指令列表中添加新添的指令：

```
wire inst_ADDU, inst_SUBU, inst_SLT, inst_AND;
wire inst_NOR, inst_OR, inst_XOR, inst_SLL;
wire inst_SRL, inst_SRA, inst_ADDIU, inst_BEQ, inst_BNE;
wire inst_LW, inst_SW, inst_LUI, inst_HUI, inst_J;

assign inst_sra = inst_SRA; // 算数右移
assign inst_lui = inst_LUI; // 立即数装载高位
assign inst_hui = inst_HUI; // 立即数装载低位
```

定义操作数和赋值：

```

wire inst_add, inst_sub, inst_slt, inst_sltu;
wire inst_and, inst_nor, inst_or, inst_xor;
wire inst_sll, inst_srl, inst_sra, inst_lui, inst_hui;

assign inst_sra = inst_SRA; // 算数右移
assign inst_lui = inst_LUI; // 立即数装载高位
assign inst_hui = inst_HUI; // 立即数装载低位

```

赋值 ALU 操作码:

```

assign alu_control = {inst_hui, // ALU操作码, 独热编码
                      inst_add,
                      inst_sub,
                      inst_slt,
                      inst_sltu,
                      inst_and,
                      inst_nor,
                      inst_or,
                      inst_xor,
                      inst_sll,
                      inst_srl,
                      inst_sra,
                      inst_lui};

```

5、实验结果分析

1.原始代码上箱图片

LOONGSON	
PC:0000004C	INST:00026883
MADDR:00000000	MDATA:00000000
REG00:00000000	REG01:00000001
REG02:00000010	REG03:00000011
REG04:00000004	REG05:0000000D
REG06:FFFFFFE2	REG07:FFFFFFF3
REG08:00000011	REG09:00000001
REG0A:0000000D	REG0B:00000000
REG0C:000C0000	REG0D:00000000
REG0E:00000000	REG0F:00000000
REG10:00000000	REG11:00000000
REG12:00000000	REG13:00000000
REG14:00000000	REG15:00000000
REG16:00000000	REG17:00000000
REG18:00000000	REG19:00000000
REG1A:00000000	REG1B:00000000
REG1C:00000000	REG1D:00000000
REG1E:00000000	REG1F:00000000

当运行完一系列指令后, REG01 应该储存 00000001H, 与图示相符, 运算正确!

REG02 应该储存 00000010H, 与图示相符, 运算正确!

REG03 应该储存 00000011H, 与图示相符, 运算正确!

REG04 应该储存 00000004H, 与图示相符, 运算正确!

REG05 应该储存 0000000DH, 与图示相符, 运算正确!

REG06 应该储存 FFFFFFFE2H, 与图示相符, 运算正确!

REG07 应该储存 FFFFFFFF3H, 与图示相符, 运算正确!

REG08 应该储存 00000011H, 与图示相符, 运算正确!

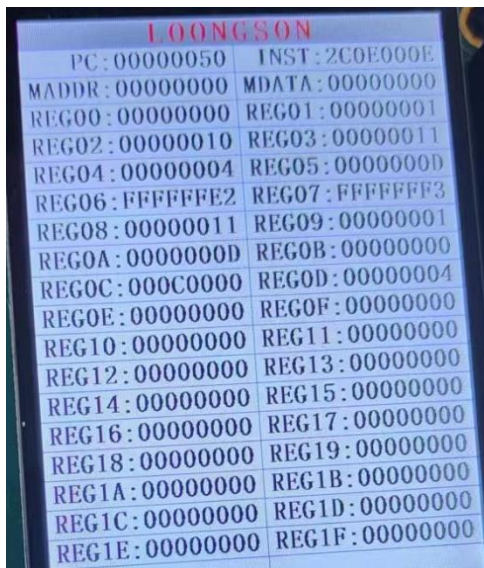
REG09 应该储存 00000001H, 与图示相符, 运算正确!

REG0A 应该储存 0000000DH, 与图示相符, 运算正确!

REG0B 应该储存 00000000H, 与图示相符, 运算正确!

REG0C 应该储存 000C0000H，与图示相符，运算正确！

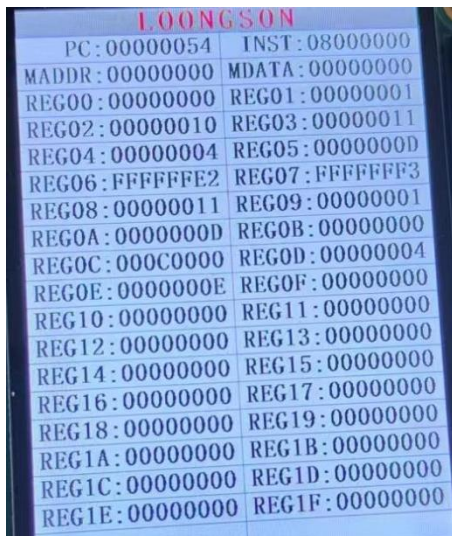
2. 逻辑右移执行完后图片



LOONGSON	
PC: 00000050	INST: 2C0E000E
MADDR: 00000000	MDATA: 00000000
REG00: 00000000	REG01: 00000001
REG02: 00000010	REG03: 00000011
REG04: 00000004	REG05: 0000000D
REG06: FFFFFFFE2	REG07: FFFFFFFF3
REG08: 00000011	REG09: 00000001
REG0A: 0000000D	REG0B: 00000000
REG0C: 000C0000	REG0D: 00000004
REG0E: 00000000	REG0F: 00000000
REG10: 00000000	REG11: 00000000
REG12: 00000000	REG13: 00000000
REG14: 00000000	REG15: 00000000
REG16: 00000000	REG17: 00000000
REG18: 00000000	REG19: 00000000
REG1A: 00000000	REG1B: 00000000
REG1C: 00000000	REG1D: 00000000
REG1E: 00000000	REG1F: 00000000

执行逻辑右移的指令 00026883H 后，执行结果储存在 REG0D 中，结果为 00000004H，运算正确！

3. 执行低位加载后图片



LOONGSON	
PC: 00000054	INST: 08000000
MADDR: 00000000	MDATA: 00000000
REG00: 00000000	REG01: 00000001
REG02: 00000010	REG03: 00000011
REG04: 00000004	REG05: 0000000D
REG06: FFFFFFFE2	REG07: FFFFFFFF3
REG08: 00000011	REG09: 00000001
REG0A: 0000000D	REG0B: 00000000
REG0C: 000C0000	REG0D: 00000004
REG0E: 0000000E	REG0F: 00000000
REG10: 00000000	REG11: 00000000
REG12: 00000000	REG13: 00000000
REG14: 00000000	REG15: 00000000
REG16: 00000000	REG17: 00000000
REG18: 00000000	REG19: 00000000
REG1A: 00000000	REG1B: 00000000
REG1C: 00000000	REG1D: 00000000
REG1E: 00000000	REG1F: 00000000

执行完低位加载的指令 2C0E000EH 后，执行结果储存在 REG0E，结果为 0000000EH，运算正确！

6、总结感想

经过实验，对 MIPS 指令的结果有了更深的理解，了解了 MIPS 指令集常用的功能和编码，了解了 CPU 设计的基本原理，学会了设计单周期 CPU，也对 Verilog 语言的使用更加熟练。