# 实验三 简易单周期CPU实验

MIPS架构CPU的传统流程可分为取指、译码、执行、访存、回写(Instruction Fetch, Decode, Execution, Memory Request, Write Back)，五阶段。

实验一完成了ALU设计并掌握了存储器IP的使用；实验二实现了单周期CPU的取指、译码阶段，完成了PC、控制器的设计。在实验一与实验二的基础上，单周期CPU的设计的各模块已经具备，再引入数字逻辑课程中所实现的多路选择器、加法器等门级组件，通过对原理图的理解，分析单条（单类型）指令在数据通路中的执行路径，依次连接对应端口，即可完成单周期CPU。

## 3.1 实验目的

（1）掌握不同类型指令在数据通路中的执行路径。

（2）掌握Vivado仿真方式。

## 3.2实验设备

PC机一台；

计算机系统能力培养实践平台（MINISYS定制开发板）

或：Nexys4 DDR实验开发板；

Xilinx Vivado开发套件(2017.x版本)。

## 3.3实验项目内容

阅读实验原理实现以下模块：

1. Datapath，其中主要包含alu(实验一已完成)，PC(实验二已完成)，adder、mux2、signext、sl2(其中adder、mux2数字逻辑课程已实现，signext、sl2参见实验原理)，
2. Controller(实验二已完成)，其中包含两部分，分别为main\_decoder，alu\_decoder。
3. 指令存储器inst\_mem(Single Port Rom)，数据存储器data\_mem(Single Port Ram)；使用Block Memory Generator IP构造指令，注意考虑PC地址位数统一。(参考实验一)
4. 参照实验原理，将上述模块依指令执行顺序连接。实验给出top文件，需兼容top文件端口设定。
5. 实验给出仿真程序，最终以仿真输出结果判断是否成功实现要求指令。

## 3.4实验原理

**3.4.1 总体框架及通路图**

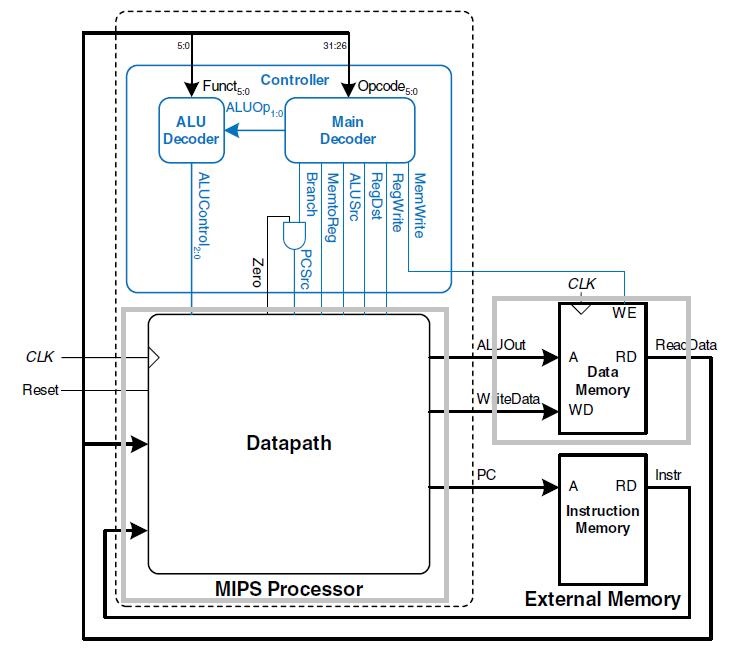


图3.1

如图3.1，完整的单周期CPU实现框架图。其中灰色线框外的部分已在实验二中完成，继续完成datapath，即可将单周期MIPS软核完整实现。

下图为单周期的完整通路图，涵盖add、and、sub、or、slt、beq、j、lw、sw、addi等指令。实验三仅实现上述指令，完整的MIPS指令集将与硬件综合设计中完善，此处以掌握数据通路分析为主要目的。



图3.2

**3.4.2 ALU**模块重设计

由于实验一中输出需要，ALU输入端口B(num2)被设定为定值，且ALU中F[2:0]对应操作与实际CPU中略有出入，故需进行修改。



图3.3

ALU实现功能见下表：

|  |  |
| --- | --- |
| **F2:0** | **Function** |
| 000 | A & B |
| 001 | A | B |
| 010 | A + B |
| 011 | not used |
| 100 | A & ~B |
| 101 | A | ~B |
| 110 | A - B |
| 111 | SLT |

满足上述功能需求，控制器中ALU decoder需满足下表，若实验二参照pdf中提供参考代码，则已支持下表中所有译码：

|  |  |
| --- | --- |
| **ALUOp1:0** | **Meaning** |
| 00 | Add |
| 01 | Subtract |
| 10 | Look at Funct |
| 11 | Not Used |

|  |  |  |
| --- | --- | --- |
| **ALUOp1:0** | **Funct** | **ALUControl2:0** |
| 00 | X | 010 (Add) |
| X1 | X | 110 (Subtract) |
| 1X | 100000 (add) | 010 (Add) |
| 1X | 100010 (sub) | 110 (Subtract) |
| 1X | 100100 (and) | 000 (And) |
| 1X | 100101 (or) | 001 (Or) |
| 1X | 101010 (slt) | 111 (SLT) |

同理，控制器中Main decoder需满足下表进行译码，参照pdf则已全部支持。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Op5:0** | **RegWrite** | **RegDst** | **AluSrc** | **Branch** | **MemWrite** | **MemtoReg** | **ALUOp1:0** |
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 0 | 00 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 |

**3.4.3组件连接：以** lw **指令在datapath中的执行路径为例**

图3.4为复杂电路模块，其中register file实验已给出。



图3.4

STEP 1: 取指，连接PC与Inst\_mem；



图3.5

STEP 2: inst[25:21]连接至RF的A1(读地址1)，输出RD1；



图3.6

STEP 3: 立即数扩展，连接inst[15:0]至signext模块，输出signimm(32 bit)；



图3.7

STEP 4: 计算数据存储器地址，其中ALU分别连接RF的RD1、Signimm，结果输出至data\_mem地址端口；



图3.8

STEP 5: 从data\_mem读出数据，并写回寄存器堆RF，写回地址为inst[20:16]；



图3.9

STEP 6: 取下一条指令，PC+4;



图3.10

**3.4.4** sw**指令**

将 rt (inst[20:16])对应寄存器数据写入内存；



图3.11

**3.4.5** R-Type**指令**

从rs(inst[25:21])与rt(inst[20:16])读取地址，加入mux2二选一选择器，AluSrc信号控制;

ALUResult 写回到RF，加入mux2，Memtoreg信号控制；

写入rd(inst[15:11]) (与lw指令写回rt(inst[20:16])区分)，加入mux2，RegDst信号控制；



图3.12

**3.4.6** beq**指令**

判断 rs 和 rt 对应寄存器值是否相等，输出zero；

计算跳转地址，写入PC:

BTA = (sign-extended immediate << 2) + (PC+4)



图3.13

**3.4.7** addi**指令**



图3.14

数据通路无变化，控制器需修改，见下表；

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Op5:0** | **RegWrite** | **RegDst** | **AluSrc** | **Branch** | **MemWrite** | **MemtoReg** | **ALUOp1:0** |
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 |
| **addi** | **001000** | **1** | **0** | **1** | **0** | **0** | **0** | **00** |

**3.4.8** j**指令**



图3.15

Inst[25:0]左移2bit，与PC+4[31:28]构成新地址写入PC；

控制器增加jump信号，控制PC写入的地址；

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Op5:0** | **RegWrite** | **RegDst** | **AluSrc** | **Branch** | **MemWrite** | **MemtoReg** | **ALUOp1:0** | **Jump** |
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 | **0** |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 | **0** |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 | **0** |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 | **0** |
| **j** | **000100** | **0** | **X** | **X** | **X** | **0** | **X** | **XX** | **1** |

## 3.5实验步骤

1、从实验一中，导入alu模块；

2、从实验二中导入PC、Controller模块；

3、从数字逻辑实验中导入多路选择器、加法器模块；

4、使用Block Memory，其中inst\_mem导入coe文件;

5、参考实验原理，连接各模块；

6、导入顶层文件及仿真文件，运行仿真；

文件目录关系如下图：

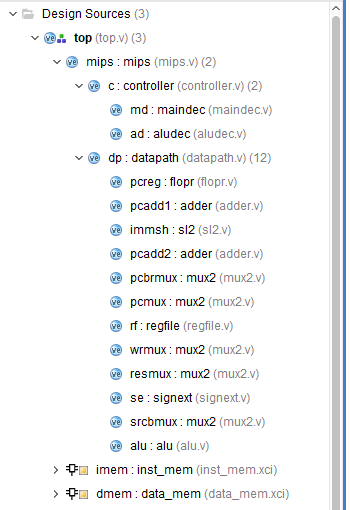


图3.17

**附录A**

实验所附的coe文件中所有指令均包含于下表中，可供查询opcode及funct所代表的具体指令。

表3.1 MIPS的31种指令

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **助记符** | **指 令 格 式** | | | | | | **示 例** | **示例含义** | **操作及解释** |
| BIT # | 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |  |  |  |
| **R-类型** | **op** | **rs** | **rt** | **rd** | **shamt** | **func** |  |  |  |
| add | 000000 | rs | rt | rd | 00000 | 100000 | add $1,$2,$3 | $1=$2+S3 | (rd)←(rs)+(rt); rs=$2,rt=$3,rd=$1 |
| sub | 000000 | rs | rt | rd | 00000 | 100010 | sub $1,$2,$3 | $1=$2-S3 | (rd)←(rs)-(rt); rs=$2,rt=$3,rd=$1 |
| and | 000000 | rs | rt | rd | 00000 | 100100 | and $1,$2,$3 | $1=$2&S3 | (rd)←(rs)&(rt); rs=$2,rt=$3,rd=$1 |
| or | 000000 | rs | rt | rd | 00000 | 100101 | or $1,$2,$3 | $1=$2|S3 | (rd)←(rs) | (rt); rs=$2,rt=$3,rd=$1 |
| slt | 000000 | rs | rt | rd | 00000 | 101010 | slt $1,$2,$3 | if($2<$3)  $1=1 else  $1=0 | if (rs< rt) rd=1 else rd=0;rs＝$2，rt=$3, rd=$1 |
| **I-类型** | **op** | **rs** | **rt** | **immediate** | | |  | | |
| addi | 001000 | rs | rt | immediate | | | addi $1,$2,10 | $1=$2+10 | (rt)←(rs)+(sign-extend)immediate,rt=$1,rs=$2 |
| lw | 100011 | rs | rt | offset | | | lw $1,10($2) | $1=Memory[  $2+10] | (rt)←Memory[(rs)+(sign\_extend)offset],  rt=$1,rs=$2 |
| sw | 101011 | rs | rt | offset | | | sw $1,10($2) | Memory[  $2+10] =$1 | Memory[(rs)+(sign\_extend)offset]←(rt),  rt=$1,rs=$2 |
| beq | 000100 | rs | rt | offset | | | beq $1,$2,40 | if($1=$2)  goto PC+4+40 | if ((rt)=(rs)) then (PC)←(PC)+4+( (Sign-Extend) offset<<2), rs=$1, rt=$2 |
| **J-类型** | **op** | **address** | | | | |  | | |
| j | 000010 | address | | | | | j 10000 | goto 10000 | (PC)←( (Zero-Extend) address<<2),  address=10000/4 |

**附录B**

**参加PDF文件**