# 实验二 单周期CPU取指译码实验

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。实验一完成了执行阶段的ALU部分，并进行了简单的访存实验，本实验将实现取指、译码两个阶段的功能。

取指阶段可分为PC自增和访问指令存储器，PC随着时钟周期变化自增，输出地址及使能信号，传至指令存储器，读出相应地址的指令。

指令取出后，需进行译码，判断具体指令类型，生成相关信号。控制器(Controller)是CPU的重要组成部分，通过控制器生成CPU数据通路中各个控制信号，使得单条指令能够正确完成数据路径的执行。

## 2.1 实验目的

（1）掌握单周期CPU控制器的工作原理及其设计方法。

（2）掌握单周期CPU各个控制信号的作用和生成过程。

（3）掌握单周期CPU执行指令的过程。

1. 掌握取指、译码阶段数据通路执行过程。

## 2.2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)。

## 2.3实验项目内容

阅读实验原理实现以下模块：

1. PC，要求输出指令存储器Inst\_Rom读使能信号ce，地址addr(长度自定，与ROM地址匹配)。若选择按字节使能(Byte Write Enable)，PC+4得到地址；否则使用PC+1。
2. Controller，其中包含两部分，分别为main\_decoder，alu\_decoder。参考MIPSfpga中控制器的实现代码，使用组合逻辑产生下列信号：

表2.1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Memtoreg | memwrite | pcsrc | alusrc | regdst | regwrite | jump | Alucontrol |
| [0:0] | [0:0] | [0:0] | [0:0] | [0:0] | [0:0] | [0:0] | [2:0] |
| Led[0] | Led[1] | Led[2] | Led[3] | Led[4] | Led[5] | Led[6] | Led[7:9] |

1. 使用Block Memory Generator IP核，构造指令存储器，注意考虑PC地址位数统一。(参考实验一)
2. 将指令存储器读取的指令显示到7段数码管中。控制器信号输出至led0-led9(参考实验一)
3. 导入仿真程序，验证取指、译码阶段正确执行。
4. 加入时钟分频器，将板载100Mhz频率降低为1hz，连接PC、Inst\_Rom时钟。(参考数字逻辑实验)
5. 建立top顶层，根据实验原理图，将上述5个模块正确连接，上板执行。

## 2.4实验原理



图2.1

实验完整原理图如图2.1所示，根据原理图可将5个模块连接并完成实验。下面分别介绍各模块原理。

**2.4.1 取指阶段原理**

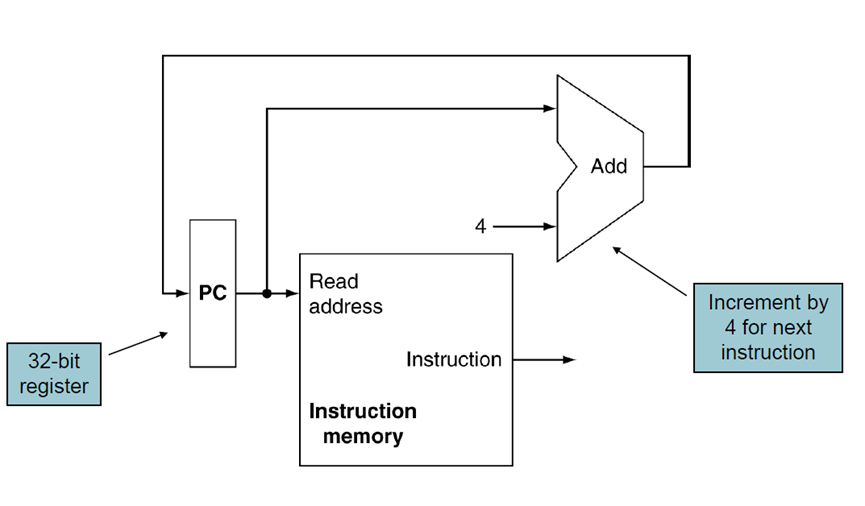


图2.2

如图2.2所示，PC为32bit(1 word)的寄存器，其存放指令地址，每条指令执行完毕后，增加4，即为下一条指令存放地址。指令地址传入指令存储器，即可取出相应地址存放的指令。

需要注意的是，MIPS架构中，采用字节读写，1 32bit word = 4 byte，故需要地址+4来获取下一条指令。

**2.4.2 指令译码原理**

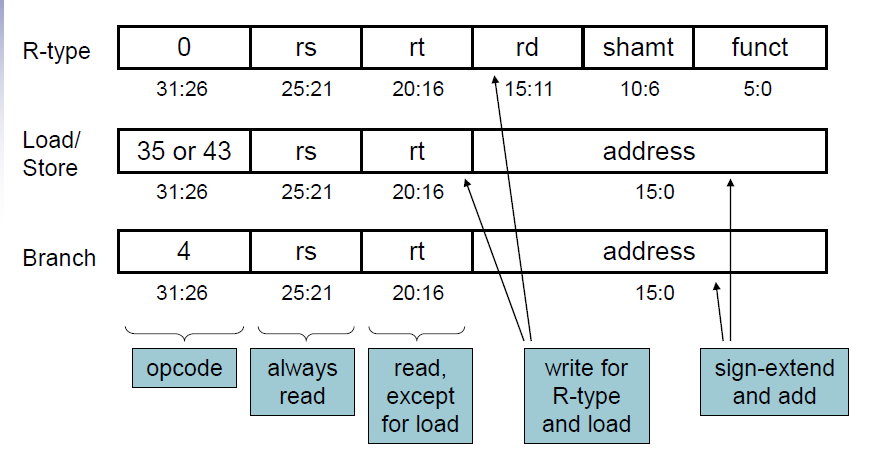


图2.3

如图2.3所示，32位MIPS指令在不同类型指令中分别有不同结构。但[31:16]表示的opcode，以及[5:0]表示的funct，为译码阶段明确指令控制信号的主要字段。下表为Opcode及funct识别得到的部分信号，详细信号表参照课本及课堂Slides。

表2.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Opcode | AluOp | Operation | Funct | Alu function | Alu control |
| Lw | 00 | Load word | XXXXXX | Add | 010 |
| Sw | 00 | Store word | XXXXXX | Add | 010 |
| Beq | 01 | Branch equal | XXXXXX | Subtact | 110 |
| R-type | 10 | Add | 100000 | Add | 010 |
| Subtract | 100010 | Subtract | 110 |
| And | 100100 | And | 000 |
| Or | 100101 | Or | 001 |
| Set-on-less-than | 101010 | SLT | 111 |

**2.4.3 控制器实现原理**

表2.3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 信号 | Memtoreg | memwrite | pcsrc | alusrc | regdst | regwrite | jump | alucontrol |
| 显示位置 | Led[0] | Led[1] | Led[2] | Led[3] | Led[4] | Led[5] | Led[6] | Led[7:9] |
| 含义 | 回写到寄存器堆 | 需要写数据存储器 | PC正常+4还是要跳转，0为正常+4，1为跳转 | 需要进行立即数的32位扩展作为第二操作数 | 指令读取时判断是rt还是rd进入寄存器组的写数据端,0为rt,1为rd | 需要写寄存器的指令 | J指令 | R-类型运算指令对应具体的运算，+、-、|、&等命令 |

分析数据通路图，判断指令是否需要写寄存器、访存等等操作，以产生相应的控制信号。

**2.4.4 总体框架及通路图**

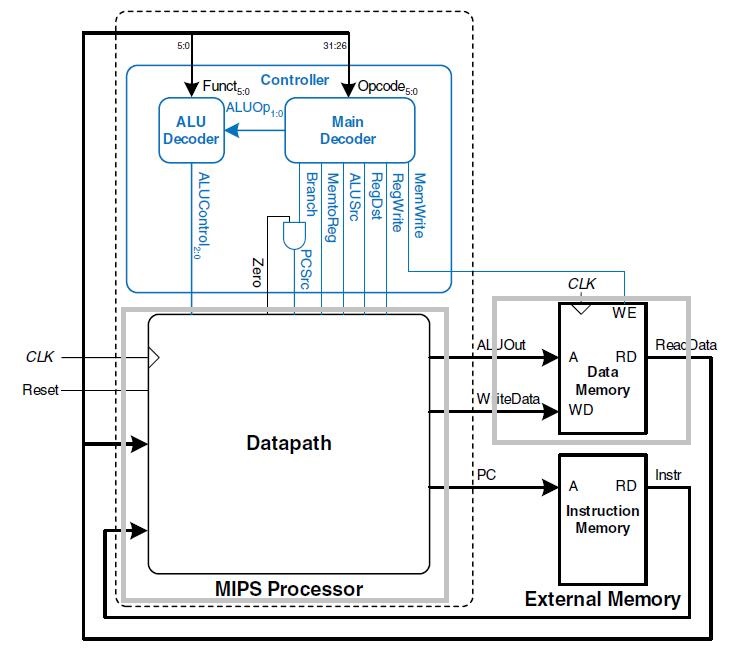


图2.4

如图2.4，完整的单周期CPU实现框架图，将controller、存储器、datapath分别进行模块化处理，将相关信号关联后，即可完成完整的MIPS处理器。本次实验将灰色线框内的部分精简，剩余部分即为图2.1所示的实验原理图。图2.5为详细数据通路中本次实验的标记。

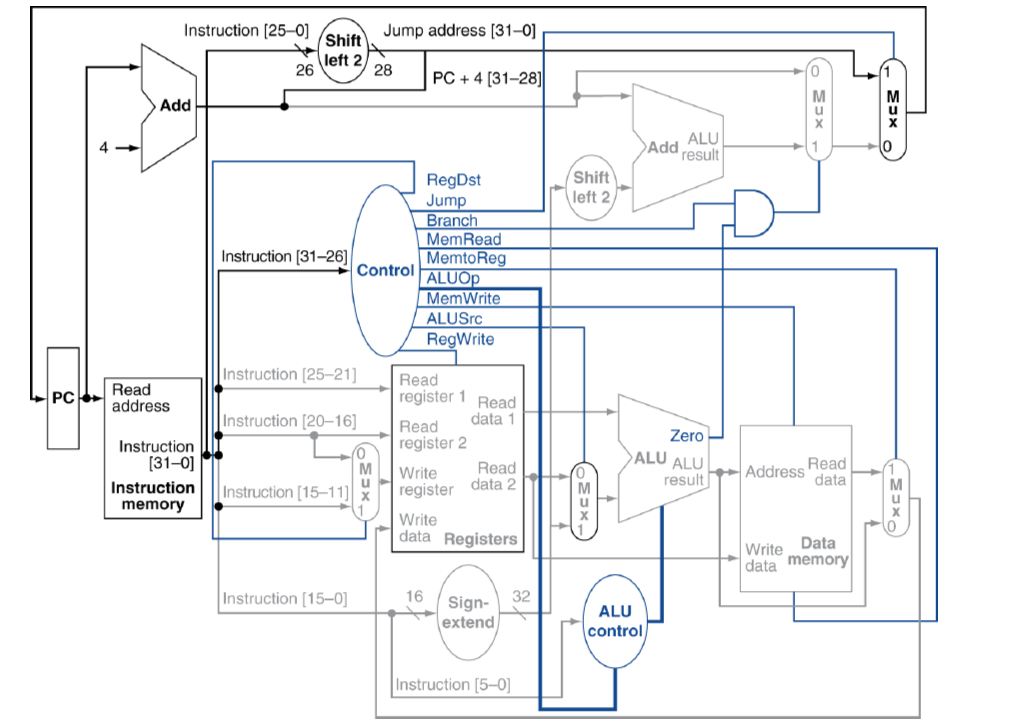


图2.5

本次实验完成图2.5中除灰色部分外的大部分功能，能够完成简单的取指译码通路。本图供单条指令所需控制信号的分析使用。

## 3.5实验步骤

1、从实验一、数字逻辑课程实验中，导入Display、clk\_div模块

2、创建PC模块

3、创建main\_decde, alu\_decode模块

4、创建Controller，调用main\_decode, alu\_decode

5、使用Block Memory，导入coe文件

6、自定义顶层文件，连接相关模块

文件目录关系如下图：

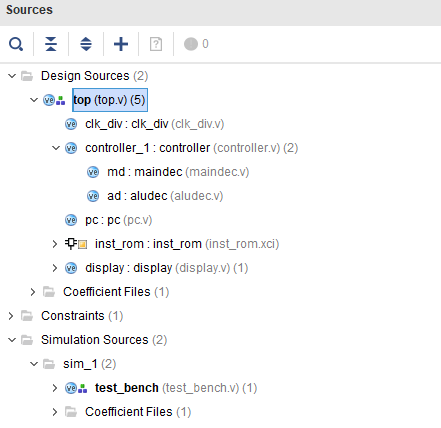


图2.6

**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 |
| addu | 000000 | rs | rt | rd | 00000 | 100001 | addu $1,$2,$3 | $1=$2+S3 | (rd)←(rs)+(rt); rs=$2,rt=$3,rd=$1,无符号数 |
| sub | 000000 | rs | rt | rd | 00000 | 100010 | sub $1,$2,$3 | $1=$2-S3 | (rd)←(rs)-(rt); rs=$2,rt=$3,rd=$1 |
| subu | 000000 | rs | rt | rd | 00000 | 100011 | subu $1,$2,$3 | $1=$2-S3 | (rd)←(rs)-(rt); rs=$2,rt=$3,rd=$1,无符号数 |
| 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 |
| xor | 000000 | rs | rt | rd | 00000 | 100110 | xor $1,$2,$3 | $1=$2^S3 | (rd)←(rs)^(rt); rs=$2,rt=$3,rd=$1 |
| nor | 000000 | rs | rt | rd | 00000 | 100111 | nor $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 |
| sltu | 000000 | rs | rt | rd | 00000 | 101011 | sltu $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, 无符号数 |
| sll | 000000 | 00000 | rt | rd | shamt | 000000 | sll $1,$2,10 | $1=$2<<10 | (rd)←(rt)<<shamt,rt=$2,rd=$1,shamt=10 |
| srl | 000000 | 00000 | rt | rd | shamt | 000010 | srl $1,$2,10 | $1=$2>>10 | (rd)←(rt)>>shamt, rt=$2, rd=$1, shamt=10, (逻辑右移) |
| sra | 000000 | 00000 | rt | rd | shamt | 000011 | sra $1,$2,10 | $1=$2>>10 | (rd)←(rt)>>shamt, rt=$2, rd=$1, shamt=10, (算术右移，注意符号位保留) |
| sllv | 000000 | rs | rt | rd | 00000 | 000100 | sllv $1,$2,$3 | $1=$2<<$3 | (rd)←(rt)<<(rs), rs=$3,rt=$2,rd=$1 |
| srlv | 000000 | rs | rt | rd | 00000 | 000110 | srlv $1,$2,$3 | $1=$2>>$3 | (rd)←(rt)>>(rs), rs=$3,rt=$2,rd=$1, (逻辑右移) |
| srav | 000000 | rs | rt | rd | 00000 | 000111 | srav $1,$2,$3 | $1=$2>>$3 | (rd)←(rt)>>(rs), rs=$3,rt=$2,rd=$1, (算术右移，注意符号位保留) |
| jr | 000000 | rs | 00000 | 00000 | 00000 | 001000 | jr $31 | goto $31 | (PC)←(rs) |
| **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 |
| addiu | 001001 | rs | rt | immediate | | | addiu $1,$2,10 | $1=$2+10 | (rt)←(rs)+(sign-extend)immediate,rt=$1,rs=$2 |
| andi | 001100 | rs | rt | immediate | | | andi $1,$2,10 | $1=$2&10 | (rt)←(rs)&(zero-extend)immediate,rt=$1,rs=$2 |
| ori | 001101 | rs | rt | immediate | | | ori $1,$2,10 | $1=$2|10 | (rt)←(rs)|(zero-extend)immediate,rt=$1,rs=$2 |
| xori | 001110 | rs | rt | immediate | | | xori $1,$2,10 | $1=$2^10 | (rt)←(rs)^(zero-extend)immediate,rt=$1,rs=$2 |
| lui | 001111 | 00000 | rt | immediate | | | lui $1,10 | $1=10\*65536 | (rt)←immediate<<16 & 0FFFF0000H，将16位立即数放到目的寄存器高16位，目的寄存器的低16位填0 |
| 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 |
| bne | 000101 | rs | rt | offset | | | bne $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 |
| slti | 001010 | rs | rt | immediate | | | slti $1,$2,10 | if($2<10)    $1=1 else    $1=0 | if ((rs)<(Sign-Extend)immediate) then (rt)←1; else (rt)←0, rs=$2, rt=$1 |
| sltiu | 001011 | rs | rt | immediate | | | sltiu $1,$2,10 | if($2<10)    $1=1 else    $1=0 | if ((rs)<(Zero-Extend)immediate) then (rt)←1; else (rt)←0, rs=$2, rt=$1 |
| **J-类型** | **op** | **address** | | | | |  | | |
| j | 000010 | address | | | | | j 10000 | goto 10000 | (PC)←( (Zero-Extend) address<<2),  address=10000/4 |
| jal | 000011 | address | | | | | jal 10000 | $31=PC+4  goto 10000 | ($31)←(PC)+4;  (PC)←( (Zero-Extend) address<<2),  address=10000/4 |