一、CPU内核的总体设计

1．CPU内核的主要性能指标和特性

1. 采用从32位MIPS CPU（RISC指令集）中提取常用指令构成的指令集，参考MIPS的体系结构进行设计，总线宽度为32位。指令字长为定长。
2. 存储结构为哈佛结构，包括4KB ROM和4KB RAM，由FPGA内部的资源编程而得。
3. 编址方式为I/O与存储器统一编址，数据线宽度为32位，地址线为32根。数据表示为定点整数。
4. 寄存器组织部分包含显示寄存器、PC寄存器、按键信息寄存器等。
5. 本CPU的字节序为小端次序。
6. 本CPU模块没有采用流水线。
7. 制造工艺，即多少纳米。

2．CPU内核的构成及各部分的简要说明

CPU的内部结构大致分为控制单元CU，运算单元ALU，数据通路datapath与内存控制器和时钟等几部分。

（1）控制器是计算机的控制中心，控制着整个CPU的工作，要完成取指令、分析指令等操作，然后交给ALU执行，同时还要形成下一条指令的地址；控制器决定了计算机运算过程的自动化，不仅保证程序的正确执行，而且能够处理异常事件；控制器一般包括指令控制逻辑、时序控制逻辑、总线控制逻辑、中断控制逻辑等几部分。

（2）运算器是计算机对数据进行加工处理的中心，主要由算术逻辑部件，寄存器组和状态寄存器组成，主要完成对二进制数的定点算术运算、逻辑运算以及移位操作。

（3）数据通路与内存控制器内有多个寄存器以及内存控制器，可以根据控制器的指令进行寄存器操作以及对主存进行读写操作。

1. CPU的指令系统

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | 功能 | 编码 | op | funct |
| add | rd←rs+rt | R | 000000 | 100000 |
| addi | rt←rs+SignExtImme | I | 001000 | n/a |
| beq | if rs=rt then branch | I | 000100 | n/a |
| bqtz | if rs>0 then branch | I | 000111 | n/a |
| j | jump to target | J | 000010 | n/a |
| sll | rd←rt <<shamt | R | 000000 | 000000 |
| slt | rd←(rs<rt) ? 1:0 | R | 000000 | 101010 |
| sub | rd←rs-rt | R | 000000 | 100010 |

1. add（加）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | rs | rt | rd | 0 | 0x20 |

功能：将通用寄存器rs，rt中的两个32位整数相加。

1. beq（相等则跳转）

|  |  |  |  |
| --- | --- | --- | --- |
| 0x04 | rs | rt | offset |

功能：比较通用寄存器rs和rt的内容，如果不想等则程序执行流程不发生转移；如果相等则跳过若干条指令。

1. addi（立即数加）

|  |  |  |  |
| --- | --- | --- | --- |
| 0x08 | rs | rt | imme |

功能：将立即数imme符号扩展至32位，与通用寄存器rs中的32位整数相加

1. bgtz（大于零则跳转）

|  |  |  |  |
| --- | --- | --- | --- |
| 0x07 | rs | 0 | offset |

功能：如果通用寄存器rs的内容如果大于0，则跳过若干条指令。

1. Sll（逻辑左移）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | rt | rd | shamt | 0 |

功能：将通用寄存器rt中的32位数逻辑左移shamt位，结果存至通用寄存器rd。

1. Slt（小于则置1）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | rs | rt | rd | 0 | 0x2a |

功能：将通用寄存器rs和rt的内容视作带符号数进行比较，若rs<rt，则向通用寄存器rd中存入1；若rs>rt,则向rd存入0。

1. Sub（减）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | rs | rt | rd | 0 | 0x22 |

功能：将通用寄存器rs、rt的两个32位整数做减法

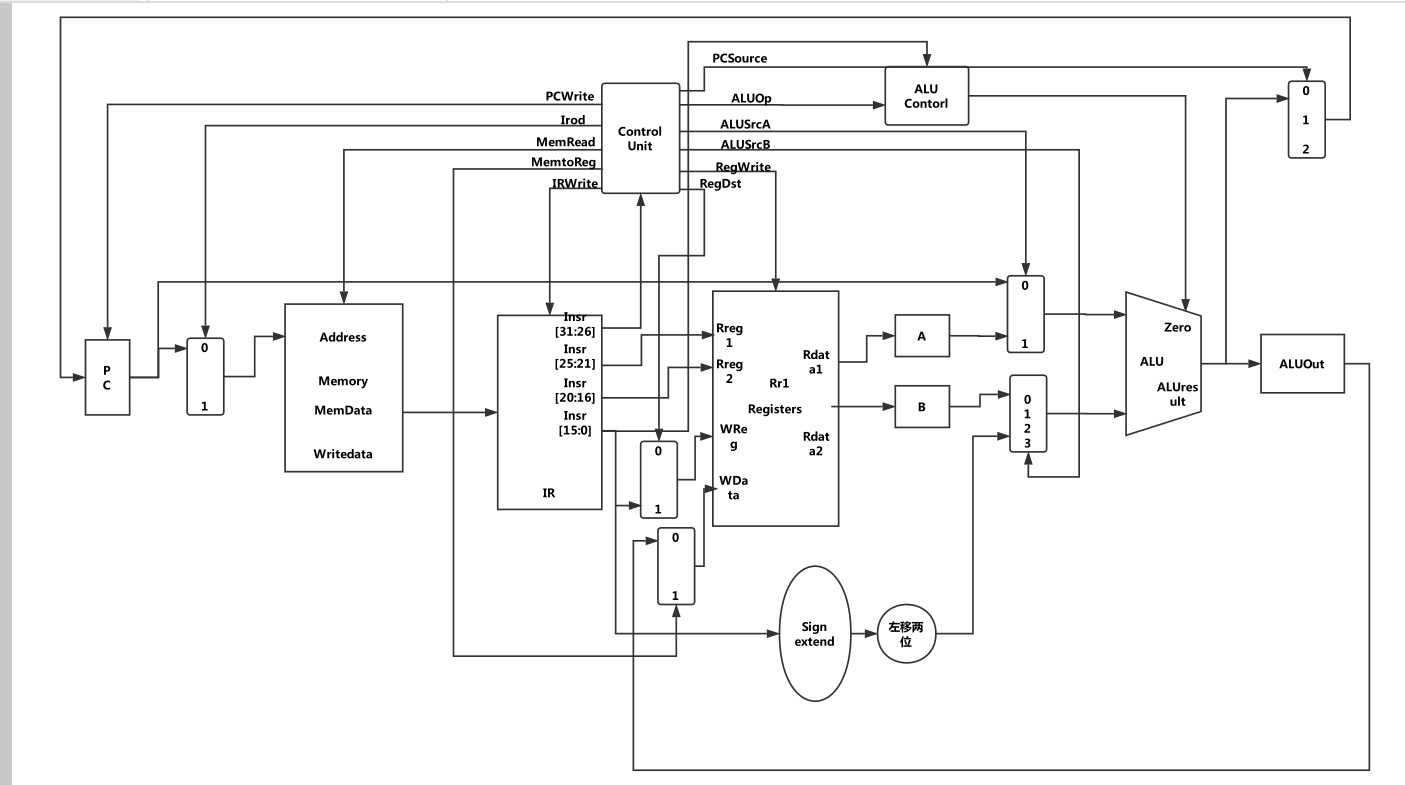
1. j（无条件转移）

|  |  |
| --- | --- |
| 0x02 | target |

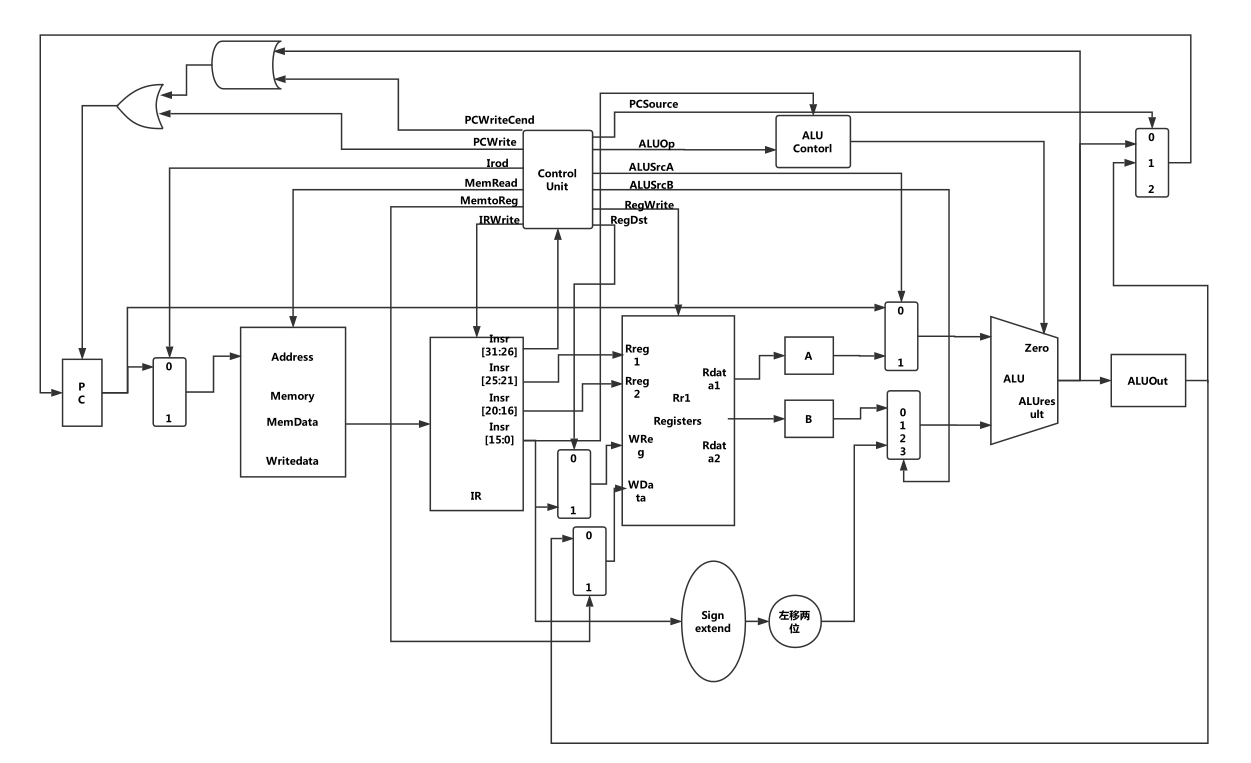
功能：改变指令的执行流程，本条指令执行完后，不再执行紧随其后的指令，而是转移到target指向的位置继续执行。

4. 按指令类型分类的CPU数据通路图

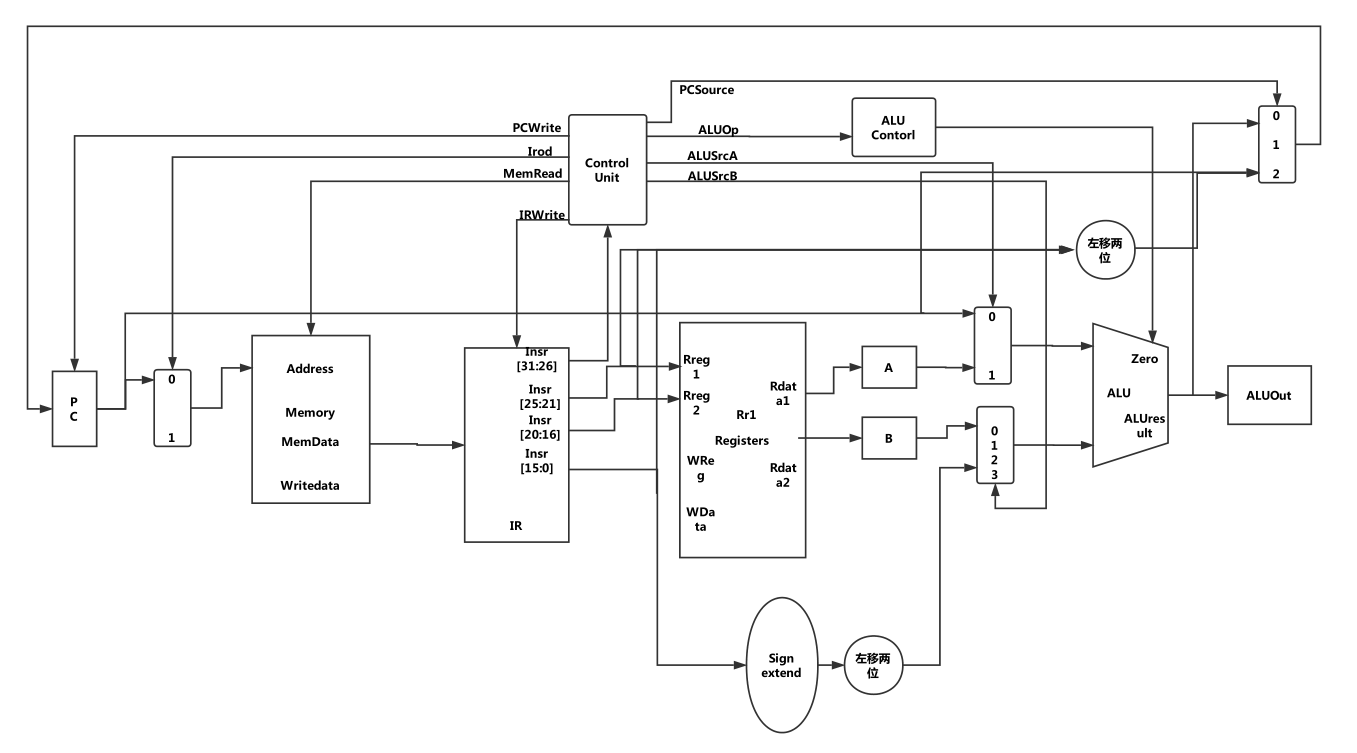
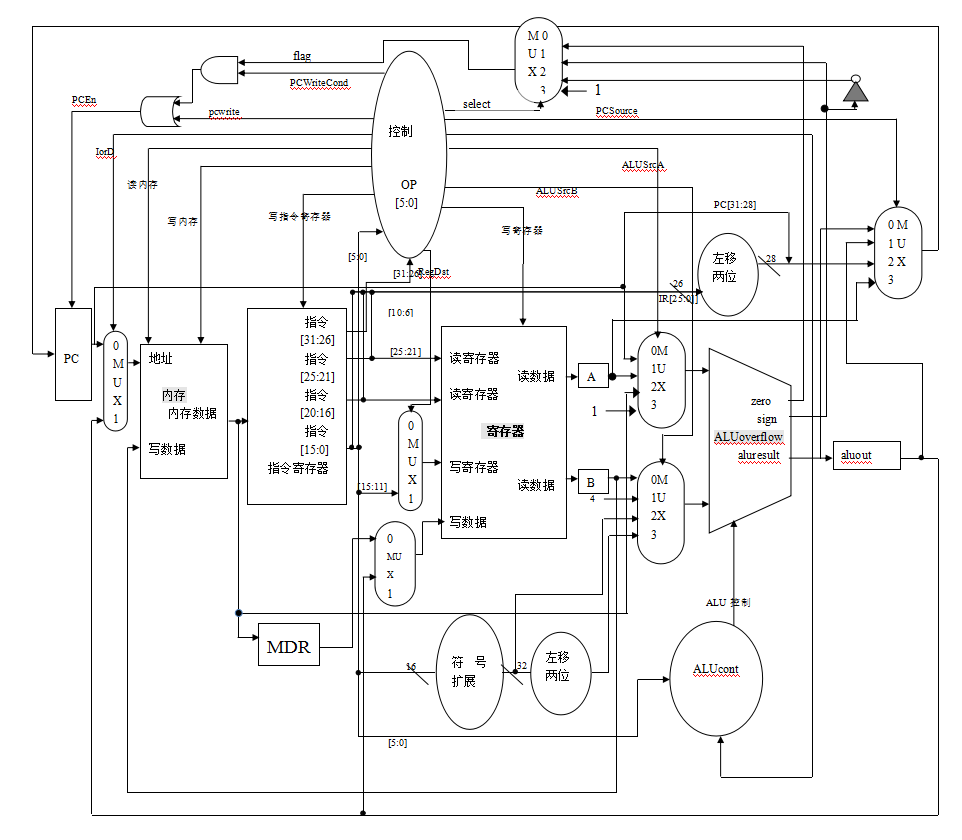
R型指令



I型指令

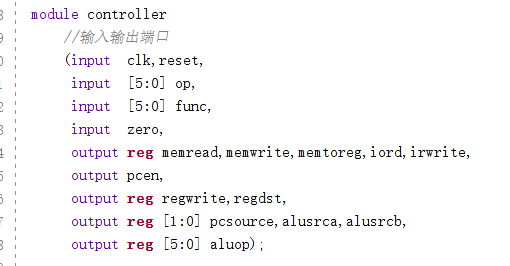


J型指令

5. 总体的CPU数据通路

6. CPU主要模块的控制信号分析

CPU控制器

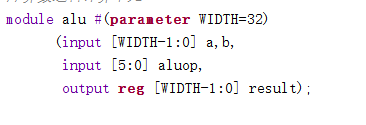


|  |  |  |
| --- | --- | --- |
| 信号 | 说明 | 控制信号有效电平 |
| Pcsource | 输出信号，控制PC信号来源 | 00/01/10 |
| Result-sec | 输出，选择运算结果 | 0/1 |
| PCWrite | 输出 ，PC写信号 | 1 |
| PCWriteCond | 输出，转移条件判断 | 1 |
| Iord | 选择存储器地址、输出 | 0/1 |
| MemRead | 输出，存储器读信号 | 1 |
| IRWrite | 输出，指令寄存器写使能 | 1 |
| RegDst | 输出，目的寄存器选择 | 0/1 |
| MemtoReg | 输出，写入寄存器数据选择 | 0/1 |
| RegWr | 输出，寄存器写使能 | 1 |
| ALUSrcA | 输出，选择ALUAD | 00/01/10/11 |
| ALUSrcB | 输出，选择ALUBD | 00/01/10/11 |
| ALUOP | 输出，ALU操作码，用于指定ALU进行哪种操作 |  |
| clk | 时钟信号 | 上升沿 |
| MemWrite | 输入，存储器写信号 | 1 |

PC

|  |  |  |
| --- | --- | --- |
| 信号 | 说明 | 控制信号有效电平 |
| PC+4 | 输入信号，输入PC+4 |  |
| PCout | 输出信号，输出PC |  |
| CLK | 时钟信号 | 上升沿 |

ALU



|  |  |  |
| --- | --- | --- |
| 信号 | 说明 | 控制信号有效电平 |
| zero | ALU输出为0信号 | 1 |
| result | ALU输出信号 | 1 |
| ALUctr | ALU控制信号 | 00/01/10 |
| a | ALU输入信号1 |  |
| b | ALU输入信号2 |  |

寄存器

|  |  |  |
| --- | --- | --- |
| 信号 | 说明 | 控制信号有效电平 |
| Ra1 | 输入信号，输入指令A |  |
| Ra2 | 输入信号，输入指令B |  |
| Regwrite | 是否写数值进入寄存器 | 1 |
| A | 输出操作数A |  |
| B | 输出操作数B |  |
| CLK | 时钟信号 |  |

7．多周期操作设计（每个周期完成功能及控制信号电平分析）

1. 取值周期

该周期从存储器中取出一条指令并计算下条指令的地址

IR<=Memory[PC]; PC<=PC+4:

执行操作：为了实现该周期的操作，需要添加几个控制信号

MemRead IRwrite并置 Iord(取指令成数据标示，0表示取指令，1表示取数据)为0，选择PC输出，实现PC+4，需要通过运算器进行PC+4，结果送回PC。

置AlusrcA=0，AlusrcB=01， Alusop=00（加法），并将PCsource=00，置PCwrite有效就可以完成该操作。PC的累加和指令的读可以在同一个周期并行完成。PC的值在下一个周期更新。(PC+4的值先暂存在到ALUout里面)。

1. 指令译码

先前的周期以及当前这个周期都不能明确这条指令具体进行什么操作，在此周期可以进行适合的所有指令的操作或不会对指令产生损害的操作。因此，本周期可以进行指令字段里面制定的两个寄存器rs和rt的读取，即使后面指令的执行过程中可能没有使用过这两个寄存器的值，但对寄存器的读取对后面指令的执行没有坏处。本周期将它们从寄存器中读出并暂存到寄存器A和寄存器B中。

本周期还完成跳转指令的计算，尽管这条指令可能并不是跳转指令，这个地址的计算是无害的，潜在的跳转地址保存在ALUout里面。

A <= Reg[ IR[25:21] ];

B <= Reg[ IR[20:16] ];

ALUout <= PC+(sign-extend)[ IR[15:0] ];

执行操作：从寄存器中读取rs和rt的值存放到寄存器A和寄存器B中，同时设置AlusrcA=0，AlusrcB=11， Alusop=00，计算跳转地址。指令的内容决定了机器执行怎样的行为。

1. 执行指令

由指令码确定执行操作的第一个周期，ALU依据之前周期准备好的两个操作数按照指令码所指的计算类型进行计算，具体的指令类型有： R:add, sll, sub,slt

I:beq,bgtz,addi

J:j  
 ALUOUT<=A op B  
 执行操作：ALU对之前周期读出的两个寄存器内容进行运算，设置ALUsrcA=1,ALUscrB=00,ALUop=10(操作由funct字段确定)  
R:add, sll, sub,slt   
 add: alusrca<=01 alusrcb<=00; ALUout<=A ALUop B  
 sub: alusrca<=01 alusrcb<=00; ALUout<=A ALUop B  
 slt: alusrca <=10 alusrcb <=00; if(A<B)  ALUout<=1

else  ALUout<=0  
 sll: alusrca <=10 alusrcb <=00; ALU操作先前一个寄存器内容对另一个字段的移位置ALUsrcA=01,ALUsrcB=10,ALUop根据funct字段而确定。  
I: addi,beq,bgtz  
 ALUout<=A op (sign-extend) immediate  
 执行操作：alusrca <=01 alusrcb <=11 aluop <=100000  
If (A op B) PC<—PC+4+(sign-extend) immediate<<2  
执行操作：PC的值被指令中的跳转地址替换，Pcsource选择跳转地址，Pcwrite信号置为有效  
J:j  
 PC<—{(PC+4)[31:28],IR[25:0].2’b00}  
执行操作：PC的值被指令中的跳转地址替换，PCSource选择跳转地址，Pcwrite信号置为有效。

1. 存储器访问/R型指令执行周期

功能：

该周期是load和store指令的访存操作，以及算数逻辑指令写回结果的周期。当从存储器中取得一个数据，先得把他暂存到存储器数据寄存器中（MDR），为下一个访存周期做准备。

控制信号电平分析：

①存储器load指令

一个数据字从存储器中获取并存入到MDR中

MDR<=Memory[ALUOUT]

控制信号：

MemRead=1 IorD=1

②存储器store指令：

数据写入到存储器中

Memory[ALUOUT]<=B

控制信号：

MemWrite=1 IorD=1

③R型指令：

将ALUOUT的值写入到结果寄存器中

Reg[instr[15:11]]<=ALUOUT

控制信号:

RegDst=1 MemtoReg=0 RegWrite=1

(5)存储器读结束周期

功能：

从存储器中读取的值写回完成load指令

控制信号电平分析：

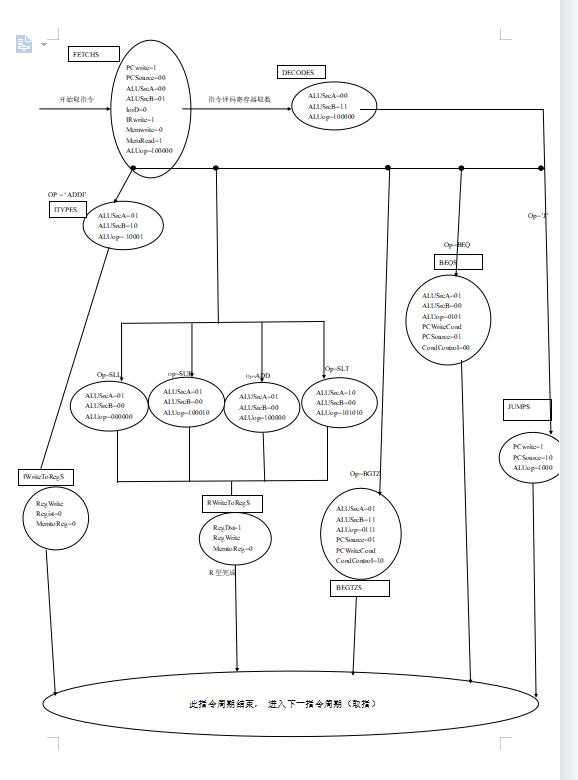
指令操作：

Reg[ra2]<=MDR

控制信号：

MemtoReg=1 RegWrite=1 RegDst=0

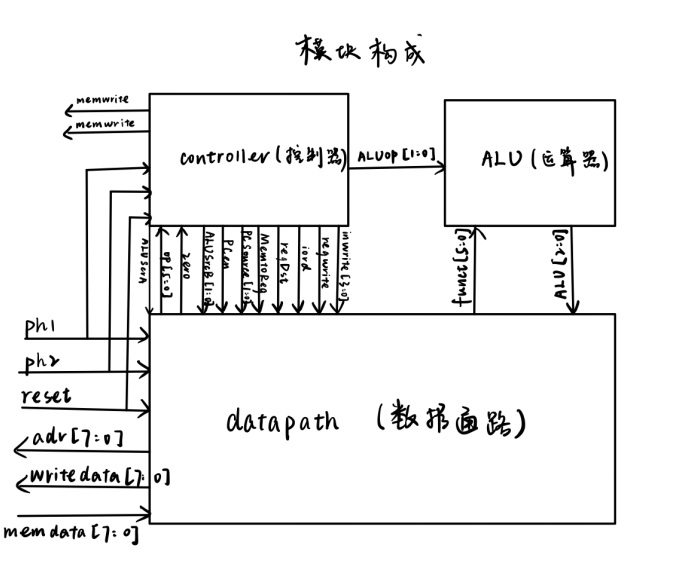
8．控制单元的状态转换图

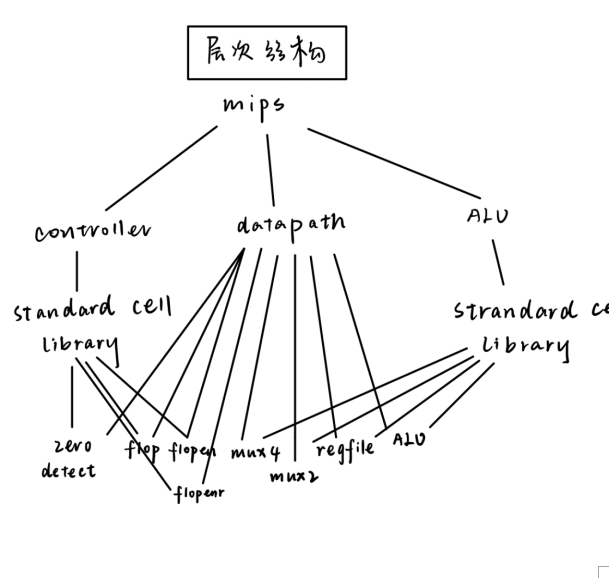


9．CPU的Verilog描述分析（模块构成及层次图）

本CPU有三个模块：

1. 控制器模块 module controller
2. 数据通路模块 module datapath
3. ALU控制模块 module alucontrol





二、SoC（System on Chip）的设计

1．SoC的内部结构框图

**exmem**

adr[15:0]

btnc

btnd

btnl memdata

btnr

btnu

clk

memwrite

reset

writedata[31:0]

**dut**

clk

adr[]31:0

memdata[31:0]

memwrite

rd3[31:0]

reset

writedata

**display1**

clk

seg[7:0]

rd3[31:0]

2．SoC的外部封装图及信号分析（输入输出信号）

scan[3:0]

seg7[7:0]

Clk

reset

reset

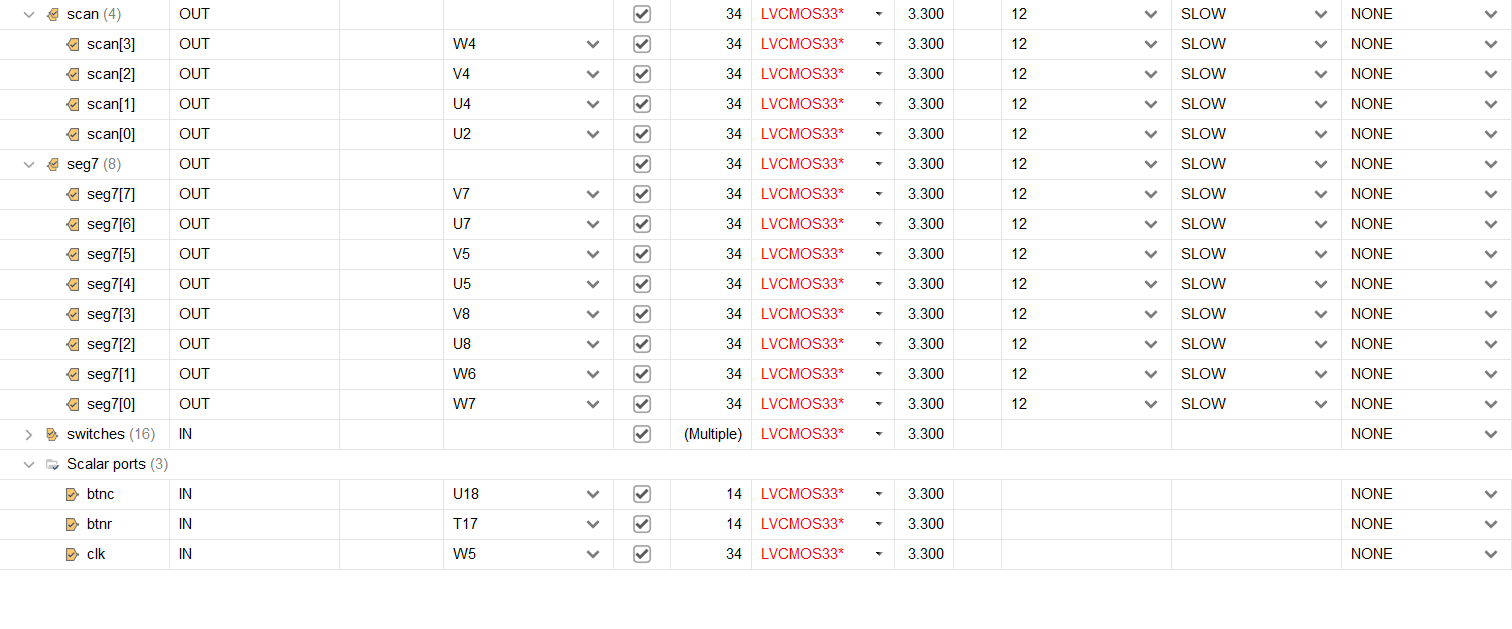
Clk：时钟信号输入

Reset：复位信号输入

Scan[3:0]：四个数码管的位选信号

Seg7[7:0]：数码管的输出信号

3．SoC的管脚约束表（信号与管脚对应表）



4．时钟模块的设计（封装图及信号分析）

在这次课设中我们用到的时钟信号是采用MMCM来生成的，MMCM即混合模式时钟管理器，用于基于某种与输入时钟之间的频率关系，生成不同的时钟信号。输入时钟频率为100MHz，即板载晶振的固有频率，本次CPU设计并没有用到分频等功能，所以没有时钟模块的设计。

5．存储器模块的设计（封装图及信号分析）

**exmem**

adr[31:0]

memdata

btnr

btnc

clk

memwrite

reset

writedata[31:0]

adr[31:0]为32位地址信号

btnc：中键，按下即计算

btnr：右键，按下复位

memdata：数据输出信号

clk：时钟信号

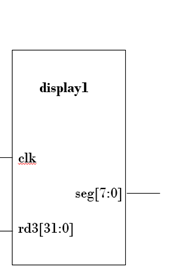
reset：预留复位信号

memwrite：控制存储器的写信号

writedata[31:0]：数据的32位输入信号

6．其它外设接口模块的设计（封装图及信号分析）

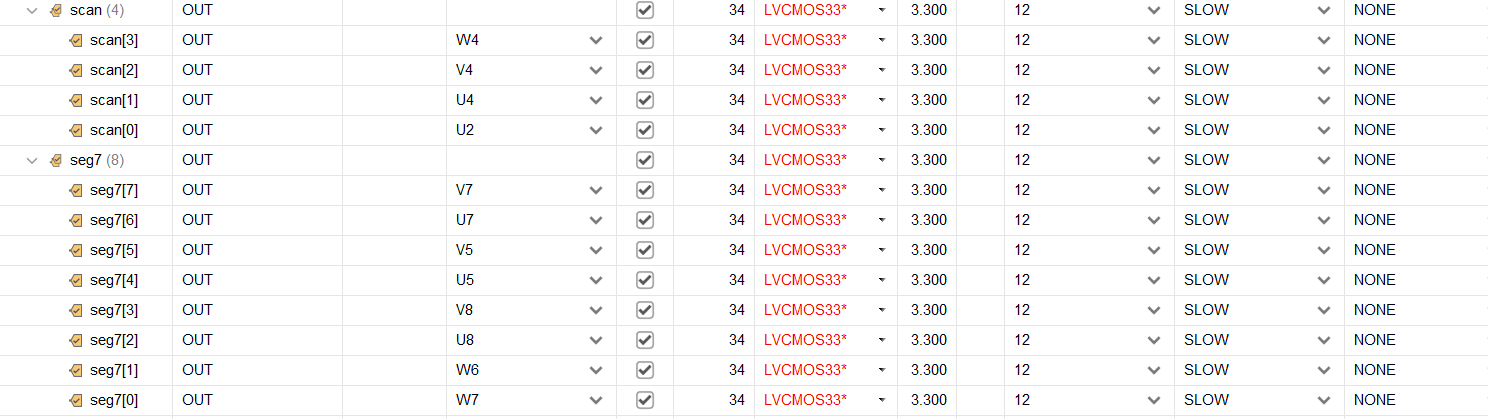
数码管显示模块的设计



clk：时钟信号

rd3[31:0]：数据读入信号

seg[7:0]：数码管输出信号



scan [3:0]数码管位选

seg7 [7；0]数码管段选

三、测试用例的编写及运行

1．测试用例的功能描述

输入：用拨码开关给出年号

处理：判断这一年是否闰年

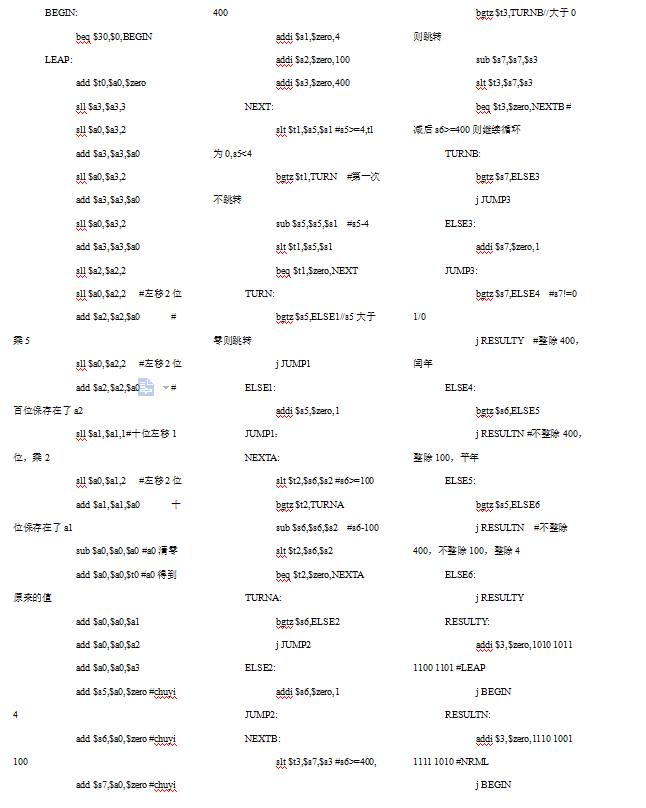
输出：

若是，数码管显示“LEAP”

若否，数码管显示“NORM”。

2．测试用例的汇编语言代码

# 指令集共8条 { beq, add, sll, sub, slt, bgtz, j, addi }



3．测试用例的机器语言代码

1FC00001

08100000

00804020

000738C0

00072080

00E43820

00072080

00E43820

00072080

00E43820

00063080

00062080

00C43020

00062080

00C43020

00052840

00052080

00A42820

00842022

00882020

00852020

00862020

00872020

0080A820

0080B020

0080B820

20110004

20120064

20130190

02B1482A

1D200003

02B1A822

02B1482A

1120FFFB

1EA00001

08100025

20150001

02D2502A

1D400003

02D2B022

02D2502A

1140FFFB

1EC00001

0810002D

20160001

02F3582A

1D600003

02F3B822

02F3582A

1160FFFB

1EE00001

08100035

20170001

1EE00001

0810003C

12A00001

0810003E

1EC00001

0810003E

0810003C

2003ABCD

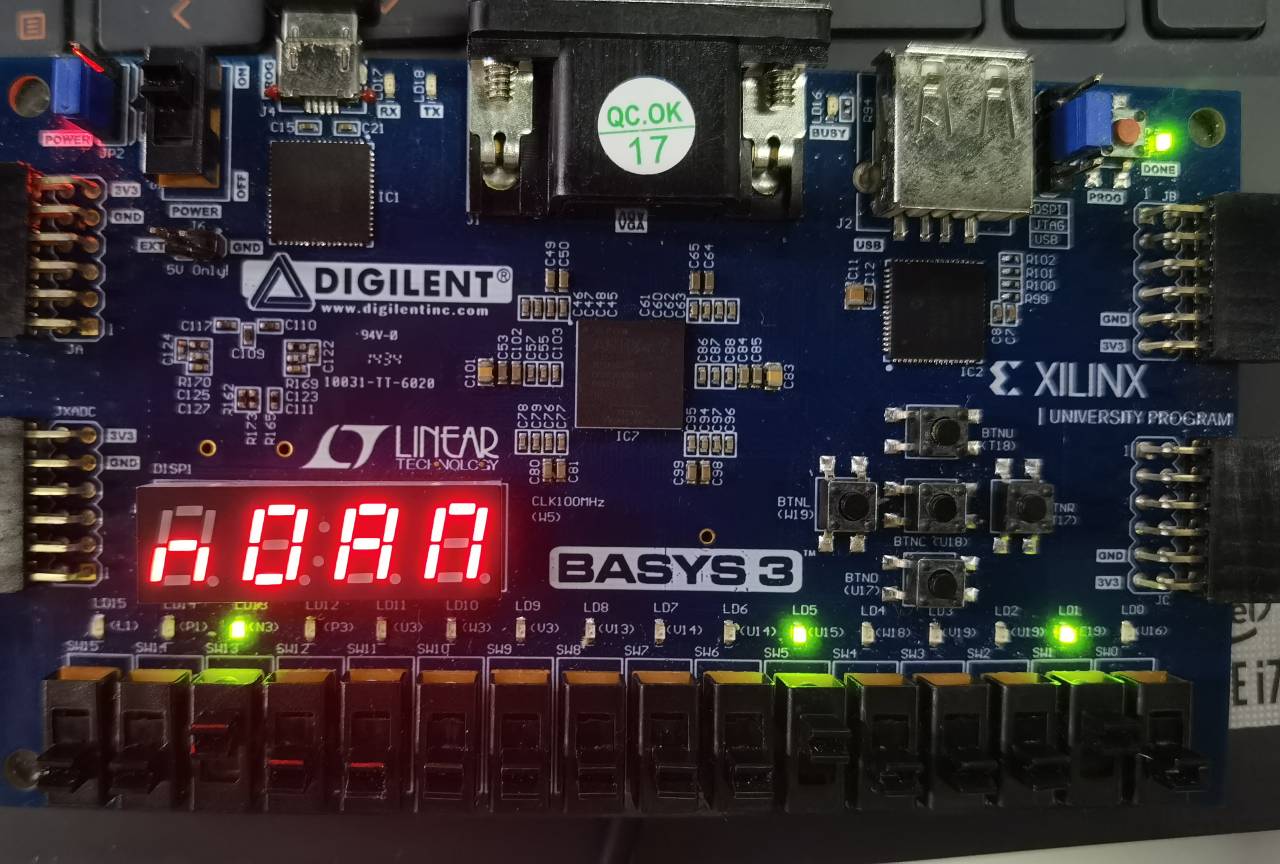
08100000

2003E0CF

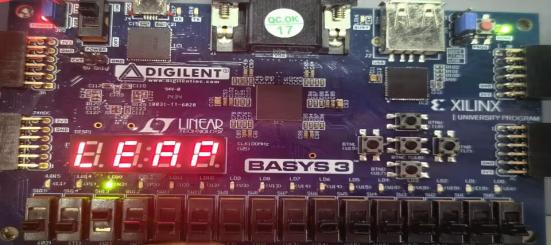
08100000

4．测试用例的运行结果

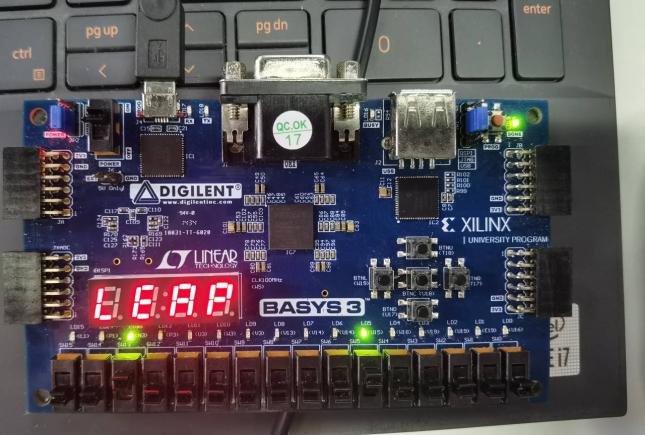
拨码输入2022，显示平年



拨码输入2000 显示闰年



拨码输入2020，显示闰年



四、设计遇到的问题及解决办法

1. 选题结束后不知道如何下手，与之前EDA课设难度不可同日而语，对设计MIPS微处理器毫无头绪。通过翻阅课程设计指导书、询问老师、同学渐渐的明晰了设计步骤；
2. 在判断闰年时需要将四个寄存器里的值放入一个寄存器里，如2020在寄存器中存放的实质上是2、0、2、0，我们需要运算的是2020，那么就需要运算2\*103+0\*102+2\*10+0，可是在书上给出的指令集里是没有乘法的。

解决办法：通过加法和移位来代替乘法运算，移位是为了凑2和22，加法是为了凑5，这样进行不停的拼凑就可以得到10，同理可以得到102，103

1. 在仿真时指令寄存器instr[31:0]一直呈现高阻态，但memdata[31:0]的值正常，而在CPU的代码设计里用到了flopen的函数，所以理论上两个的值应当相等

未完全解决，但可以在开发板上运行，说明instr[31:0]一定是读入了指令的。

1. 汇编文件逻辑待优化，面对一些特殊的数字运算时会耗费大量时间。

未完全解决，重新编写了汇编语言修改了不合理的逻辑，但由于时间关系，新的汇编程序的机器码未能成功编写好。

（5）第一遍仿真的时候一直卡在current time ：0fs……

解决办法：问题出在always上，因为没有给出频率信号就会一直陷入死循环导致运行不出结果，改成always #50之后就能正常运行了。

五、设计总结及心得体会

在本次计算机组成原理课设中，我收获了很多；首先让我对计算机组成原理的理解不仅仅浮于表面，有了更深的理解，对冯诺依曼体系结构的计算机有了一定的认识；并且还了解了数据通路、时序电路这些课本上学不到的知识，还理解了控制器的功能和组成，理解了各类典型指令的执行流程，对指令格式、寻址方式、指令系统、指令分类有了系统的概念，还让我们学会了指令的设计以及相关技术除此之外，我的动手能力和学习能力、以及对计算机系统的分析、设计能力都有了重大的提升。下面是我的一些心得体会。

1. 在设计多周期CPU的数据通路的时候，应该弄清楚CPU的数据通路以及各种控制信号的含义，而不应该盲目下手，以防止出现很多低级错误。我们在一开始就把很多控制信号和数据通路的含义不了解，以至于我们在设计时走了不少偏路。
2. 本次课程设计最核心的便是对于CPU指令的设计，在确定功能的基础上设计CPU的电路和指令的使用，因此，课设开始的地方和课设最重要的地方都是对于CPU指令的选择和设计。根据功能确定CPU的指令的选择，要注意指令的多少和使用才是整个课程设计的核心，整个功能的实现应该在指令的使用中实现，需要设计CPU的verilog代码，但是也要注意把重点放在调用指令实现功能上面。不同类型的指令有不同的数据通路，有不同的执行方式，尽管对于大多数指令来说，指令周期大体相同，但是一定要将所有指令的执行周期情况了熟于心，对于不同指令，做好区分，做好理解，在编写CPU代码之前要将所有选择的指令的执行过程做好细分和设计。
3. 在单周期CPU中，在每个周期内，各个部件的控制信号不会改变，所以，部件之间的处理结果也不会改变。这样，各部件之间的数据通路图就可以采用硬连接的方式；但是，在多周期CPU中，即使在处理一条指令的过程中，随着周期的不同，控制部件的控制信号也不同，这样，在不同周期中发挥作用的部件之间传递数据时，就需要在中间加一级缓存，以防止在传输过程中丢掉一些数据；
4. 本次课设中还很重要的一点就是验证指令的正确性，我们需要在仿真中一条一条的来验证我们所写的指令，发现问题后立马改正；
5. 本次课设总体来说是我面临过最难的一次挑战，在小组同学的通力合作下基本成功完成了任务，让我认识到合作的魅力。