



**计 算 机 组 成 原 理 课 程 设 计**

流水线 CPU 团队实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 | 王娟 |
| 组 长 | 杨训迪 |
| 组 员 | 丁颖 肖成文 周雨佳 |
| 组长联系方式 | 18810783722 |

二O二O年九月

目 录

[第一章 设计题目 1](#_Toc52742559)

[第二章 组员分工 1](#_Toc52742560)

[第三章 设计目的 2](#_Toc52742561)

[第四章 设计环境 2](#_Toc52742562)

[第五章 设计原理及内容 2](#_Toc52742563)

[5.1 CPU整体架构 2](#_Toc52742564)

[5.1.1 数据通路 4](#_Toc52742565)

[5.1.2 控制逻辑 4](#_Toc52742566)

[5.2 由单周期CPU到流水线CPU 7](#_Toc52742567)

[5.2.1 结构相关 8](#_Toc52742568)

[5.2.2 数据相关 8](#_Toc52742569)

[5.2.3 Load延迟 8](#_Toc52742570)

[5.2.4 控制相关 8](#_Toc52742571)

[第六章 设计步骤 9](#_Toc52742572)

[6.1 verilog代码编写 9](#_Toc52742573)

[6.1.1 CPU五级设计 9](#_Toc52742574)

[6.1.2 流水效果的实现 12](#_Toc52742575)

[6.1.3 冒险与解决办法 13](#_Toc52742576)

[6.2 下板测试 17](#_Toc52742577)

[第七章 问题及解决方法 19](#_Toc52742578)

[第八章 心得体会及总结 19](#_Toc52742579)

[第九章 参考文献 19](#_Toc52742580)

# 设计题目

实现经典五级流水线CPU，支持41条MIPS指令，包含冒险冲突处理，结合汇编与接口设计实验，设计测试用例，并下板验证成功。

# 组员分工

小组分工见表2.1 小组分工。

|  |  |
| --- | --- |
| **组员** | **分工任务** |
| 杨训迪 | * 41条指令的数据通路和控制逻辑的设计 * 跳转模块，控制模块，存储模块的verilog编写 * 代码整合，top文件，41条指令的单条测试 * 流水线CPU实验报告的部分编写 * soc文件，数据存储器IP核(未完成) |
| 丁颖 | * vga实现和mips实现 * vga与cpu接口对接 * vga下板演示 * 汇编与接口实验报告的部分编写 |
| 肖成文 | * 冒险处理的数据通路和控制逻辑的设计与实现 * 寄存器堆模块，ALU模块的verilog编写 * 冒险处理板块功能的测试，41条指令的综合测试及相关结果的数码管显示 * 流水线CPU实验报告的部分编写 |
| 周雨佳 | * 41条指令汇编测试代码编写 * CP0实现异常和中断处理（未整合） * PPT制作 * 汇编与接口实验报告的部分编写 |

表2.1 小组分工

# 设计目的

深入理解计算机相关理论，熟练掌握计算机软硬件工作原理，学习并掌握计算机接口设计，学习并掌握RISC指令集的处理器设计，熟悉Vivado工具应用。培养计算机系统能力。

# 设计环境

|  |  |
| --- | --- |
| 操作系统 | Windows 10 |
| 编程语言 | Verilog |
| 编辑器 | Vivado 2019 |
| 汇编语言 | MIPS |
| 汇编程序编辑器 | Mars |

# 设计原理及内容

## CPU整体架构

本流水线CPU覆盖41指令(见表5.1-1)，下面将从数据通路和控制逻辑阐述设计思路。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 指令 | opcode | function | 功能 |
| 1 | ADD | 000000 | 100000 | rd=rs+rt |
| 2 | ADDU | 000000 | 100001 | rd=rs+rt (无符号数) |
| 3 | ADDI | 001000 | / | rd=rs+im |
| 4 | ADDIU | 001001 | / | rd=rs+im（无符号数） |
| 5 | SUB | 000000 | 100010 | rd=rs-rt |
| 6 | SUBU | 000000 | 100011 | rd=rs-rt (无符号数) |
| 7 | SLT | 000000 | 101010 | rd=(rs<rt)?1:0 |
| 8 | SLTI | 001010 | / | rd=(rs<im)?1:0 |
| 9 | SLTU | 000000 | 101011 | rd=(rs<rt)?1:0 (无符号数) |
| 10 | SLTIU | 001011 | / | rd=(rs<im)?1:0（无符号数） |
| 11 | AND | 000000 | 100100 | rd=rs&rt |
| 12 | ANDI | 001100 | / | rd=rs&im |
| 13 | NOR | 000000 | 100111 | rd=!(rs|rt) |
| 14 | OR | 000000 | 100101 | rd=rs|rt |
| 15 | ORI | 001101 | / | rd=rs|im |
| 16 | XOR | 000000 | 100110 | rd=rs xor rd |
| 17 | XORI | 001110 | / | rd=rs xor im |
| 18 | SLL | 000000 | 000000 | rd=rt<<shamt |
| 19 | SLLV | 000000 | 000100 | rd=rt<<rs |
| 20 | SRL | 000000 | 000010 | rd=rt>>shamt |
| 21 | SRLV | 000000 | 000110 | rd=rt>>rs |
| 22 | SRA | 000000 | 000011 | rd=rt>>shamt（符号位保留） |
| 23 | SRAV | 000000 | 000111 | rd=rt>>rs（符号位保留） |
| 24 | LUI | 001111 | / | rt=im\*65536 |
| 25 | CLZ | 011100 | 100000 | clz rd,rs |
| 26 | CLO | 011100 | 100001 | clo rd,rs |
| 27 | BEQ | 000100 | / | PC=(rs==rt)?PC+im<<2:PC |
| 28 | BNE | 000101 | / | PC=(rs!=rt)?PC+im<<2:PC |
| 29 | BGTZ | 000111 | / | PC=(rs>0)?PC+im<<2:PC |
| 30 | BLEZ | 000110 | /(rt) | PC=(rs<=0)?PC+im<<2:PC |
| 31 | BGEZ | 000001 | /(rt) | pc = (rs>=0)pc+offset<<2:pc |
| 32 | BLTZ | 000001 | / | pc = (rs<0)pc+offset<<2:pc |
| 33 | J | 000010 | / | PC={(PC+4)[31,28],addr,00} |
| 34 | JAL | 000011 | / | $31=PC;PC={(PC+4)[31,28],addr,00} |
| 35 | JR | 000000 | 001000 | PC=rs |
| 36 | LB | 100000 | / | LB rt, offset(base) |
| 37 | LH | 100001 | / | LH rt, offset(base) |
| 38 | LW | 100011 | / | LW rt, offset(base) |
| 39 | SB | 101000 | / | SB rt, offset(base) |
| 40 | SH | 101001 | / | SH rt, offset(base) |
| 41 | SW | 101011 | / | SW rt, offset(base) |

表5.1-1 41条指令

### 数据通路

下图展示了五级流水线CPU的数据通路，可划分为取值，译码/分析，执行，访存，回写五级，级与级之间通过锁存器保存和传递数据，通过hazards\_unit实现解决冒险冲突等问题。

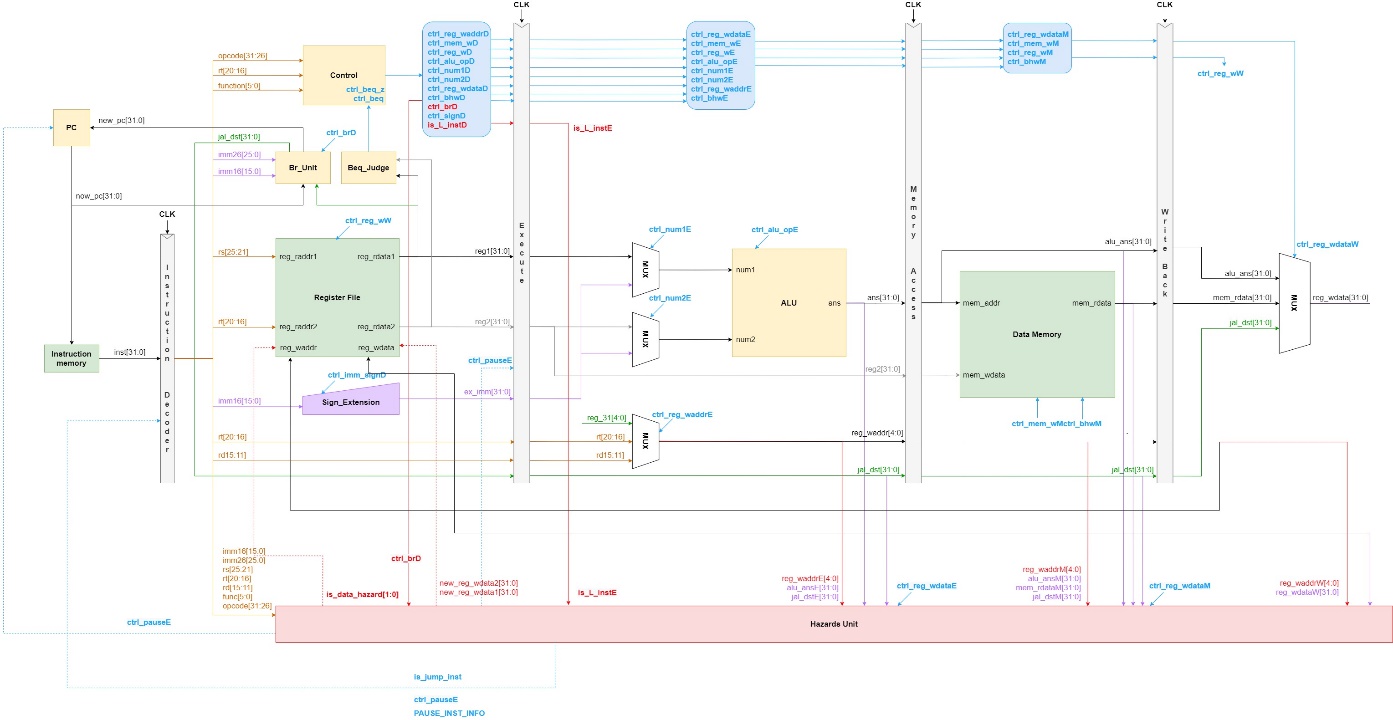


图5.1.1-1 五级流水线CPU数据通路

### 控制逻辑

一共有12个基本控制信号(不包含流水线和冒险处理控制处理)，控制信号对应的位宽和控制对象见表5.1.2-1，41条指令对应的控制信号见表5.1.2-2。

|  |  |  |
| --- | --- | --- |
| 控制信号 | 位宽 | 控制对象 |
| ctrl\_reg\_w | 1 | register file |
| ctrl\_reg\_waddr | 3 | mux\_reg\_waddr |
| ctrl\_reg\_wdata | 3 | mux\_reg\_wdata |
| ctrl\_mem\_w | 1 | data\_mem |
| ctrl\_alu\_op | 13 | ALU |
| ctrl\_num1 | 1 | mux\_num1 |
| ctrl\_num2 | 1 | mux\_num2 |
| ctrl\_br | 4 | Br\_unit |
| ctrl\_beq | 1 | Control |
| ctrl\_beq\_z | 4 | Control |
| ctrl\_imm\_sign | 2 | sign\_extension |
| ctrl\_bhw | 2 | data\_mem |

表5.1.2-1控制信号

控制信号所控制的信息及取值见下:

1. ctrl\_reg\_w: 控制写入数据至寄存器堆。值为1: 写; 值为0: 不写
2. ctrl\_reg\_waddr: 控制将写入寄存器堆的地址来源。值为001: 地址来源于rt; 值为010: 地址来源于rd; 值为100: 地址为31号寄存器。
3. ctrl\_reg\_wdata: 控制将写入寄存器堆的数据来源。值为001: 数据来源于ALU的计算结果; 值为010: 数据来源于data\_mem读出的数据; 值为100: 数据来源于跳转指令后面的第二条指令的地址(jal)。
4. ctrl\_mem\_w: 控制写入数据至数据存储器。值为1: 写入; 值为0: 不写
5. ctrl\_alu\_op: 控制ALU的运算符，可将指令归为13种运算。
6. ctrl\_num1: 控制第一个操作数的来源。值为0: 操作数来源于立即数; 值为1: 操作数来源于rs所指向的内容
7. ctrl\_num2: 控制第二个操作数的来源。值为0: 操作数来源于立即数; 值为1: 操作数来源于rt所指向的内容
8. ctrl\_br: 控制跳转方式。当指令为BEQ/BNE/BGTZ/BLEZ/BGEZ/BLTZ, 值为0001; 当指令为J, 值为0010; 当指令为JAL, 值为0100; 当指令为JR, 值为1000
9. ctrl\_beq: 两数比较结果。当rs == rt，值为1; 当rs != rt，值为0
10. ctrl\_beq\_z: 第一个操作数与0比较结果。当比较结果为0001; 当比较结果为 , 值为010; 当比较结果为 , 值为100; 当比较结果为 , 值为1000
11. ctrl\_imm\_sign: 控制符号扩展的方式。值为00: 有符号扩展; 值为01: 无符号扩展; 值为10: 取出移位量并进行相应的扩展
12. ctrl\_bhw: 控制存取的数据大小。值为00: 字; 值为01: 半字; 值为10: 字节

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 指令 | ctrl\_reg\_w | ctrl\_reg\_waddr | ctrl\_reg\_wdata | ctrl\_mem\_w | ctrl\_alu\_op | ctrl\_num1 | ctrl\_num2 | ctrl\_br | ctrl\_beq | ctrl\_beq\_z | ctrl\_imm\_sign | ctrl\_bhw |
| 1 | ADD | 1 | 010 | 001 | 0 | 0000000000001 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 2 | ADDU | 1 | 010 | 001 | 0 | 0000000000001 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 3 | ADDI | 1 | 001 | 001 | 0 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 4 | ADDIU | 1 | 001 | 001 | 0 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 01 | 00 |
| 5 | SUB | 1 | 010 | 001 | 0 | 0000000000010 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 6 | SUBU | 1 | 010 | 001 | 0 | 0000000000010 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 7 | SLT | 1 | 010 | 001 | 0 | 0000000000100 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 8 | SLTI | 1 | 001 | 001 | 0 | 0000000000100 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 9 | SLTU | 1 | 010 | 001 | 0 | 0000000000100 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 10 | SLTIU | 1 | 010 | 001 | 0 | 0000000000100 | 1 | 0 | 0000 | 0 | 0000 | 01 | 00 |
| 11 | AND | 1 | 010 | 001 | 0 | 0000000001000 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 12 | ANDI | 1 | 001 | 001 | 0 | 0000000001000 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 13 | NOR | 1 | 010 | 001 | 0 | 0000000010000 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 14 | OR | 1 | 010 | 001 | 0 | 0000000100000 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 15 | ORI | 1 | 001 | 001 | 0 | 0000000100000 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 16 | XOR | 1 | 010 | 001 | 0 | 0000001000000 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 17 | XORI | 1 | 001 | 001 | 0 | 0000001000000 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 18 | SLL | 1 | 010 | 001 | 0 | 0000010000000 | 0 | 1 | 0000 | 0 | 0000 | 10 | 00 |
| 19 | SLLV | 1 | 010 | 001 | 0 | 0000010000000 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 20 | SRL | 1 | 010 | 001 | 0 | 0000100000000 | 0 | 1 | 0000 | 0 | 0000 | 10 | 00 |
| 21 | SRLV | 1 | 010 | 001 | 0 | 0000100000000 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 22 | SRA | 1 | 010 | 001 | 0 | 0001000000000 | 0 | 1 | 0000 | 0 | 0000 | 10 | 00 |
| 23 | SRAV | 1 | 010 | 001 | 0 | 0001000000000 | 1 | 1 | 0000 | 0 | 0000 | 00 | 00 |
| 24 | LUI | 1 | 001 | 001 | 0 | 0010000000000 | 0 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 25 | CLZ | 1 | 010 | 001 | 0 | 0100000000000 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 26 | CLO | 1 | 010 | 001 | 0 | 1000000000000 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 27 | BEQ | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 0001 | 1 | 0000 | 00 | 00 |
| 28 | BNE | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 0001 | 0 | 0000 | 00 | 00 |
| 29 | BGTZ | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 0001 | 0 | 0001 | 00 | 00 |
| 30 | BLEZ | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 0001 | 0 | 0010 | 00 | 00 |
| 31 | BGEZ | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 0001 | 0 | 0100 | 00 | 00 |
| 32 | BLTZ | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 0001 | 0 | 1000 | 00 | 00 |
| 33 | J | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 0010 | 0 | 0000 | 00 | 00 |
| 34 | JAL | 0 | 100 | 100 | 0 | 0000000000000 | 0 | 0 | 0100 | 0 | 0000 | 00 | 00 |
| 35 | JR | 0 | 000 | 000 | 0 | 0000000000000 | 0 | 0 | 1000 | 0 | 0000 | 00 | 00 |
| 36 | LB | 1 | 001 | 010 | 0 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 00 | 10 |
| 37 | LH | 1 | 001 | 010 | 0 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 00 | 01 |
| 38 | LW | 1 | 001 | 010 | 0 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |
| 39 | SB | 0 | 000 | 000 | 1 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 00 | 10 |
| 40 | SH | 0 | 000 | 000 | 1 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 00 | 01 |
| 41 | SW | 0 | 000 | 000 | 1 | 0000000000001 | 1 | 0 | 0000 | 0 | 0000 | 00 | 00 |

表5.1.2-2 41条指令控制逻辑信号

## 由单周期CPU到流水线CPU

由前面单周期CPU的实验，我们可以知道，CPU一条指令的执行大致可以分为5个阶段——即取指（IF），译码（ID），执行（EX），访存（MEM），写回（WB）。而在流水线CPU中，我们想要实现多条指令同时在CPU中并行流水的效果（如图5.2-1所示），则需要通过一组并行的触发器来缓存当前的指令状态信息，并在时钟信号的控制下，进行有节奏的逐级流水（如图5.2-2所示）。

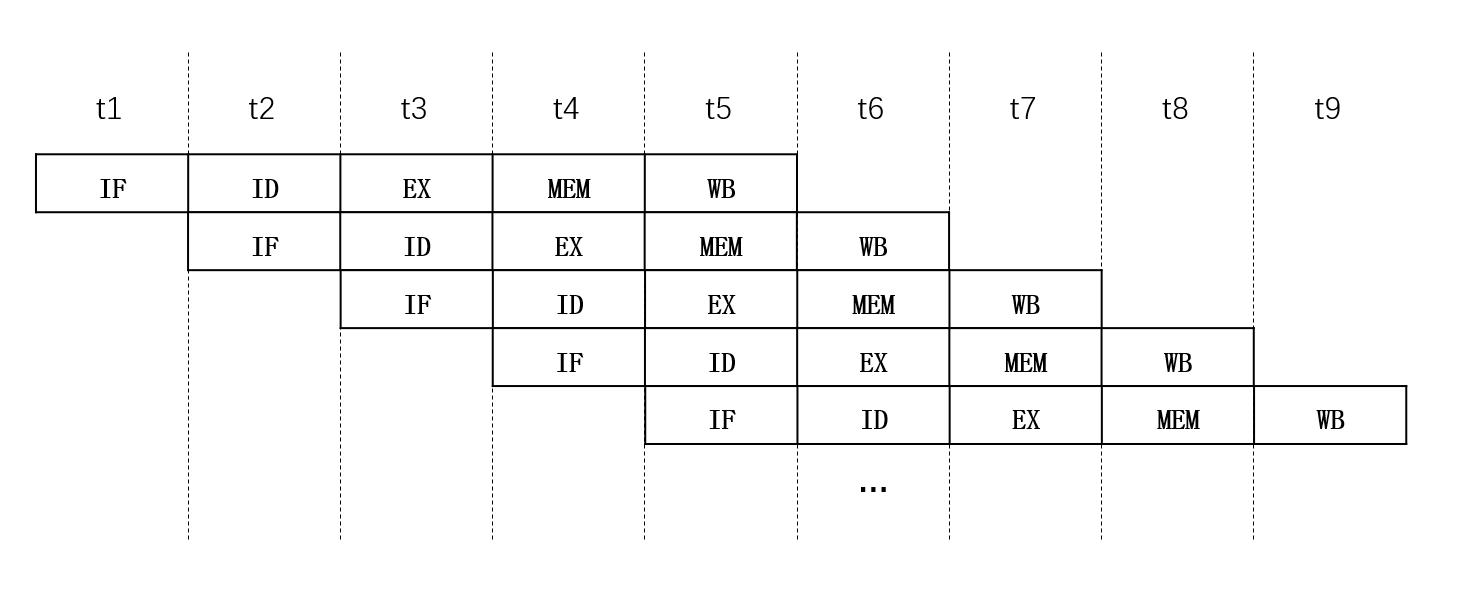


图5.2-1 流水线CPU指令执行过程（五级流水）

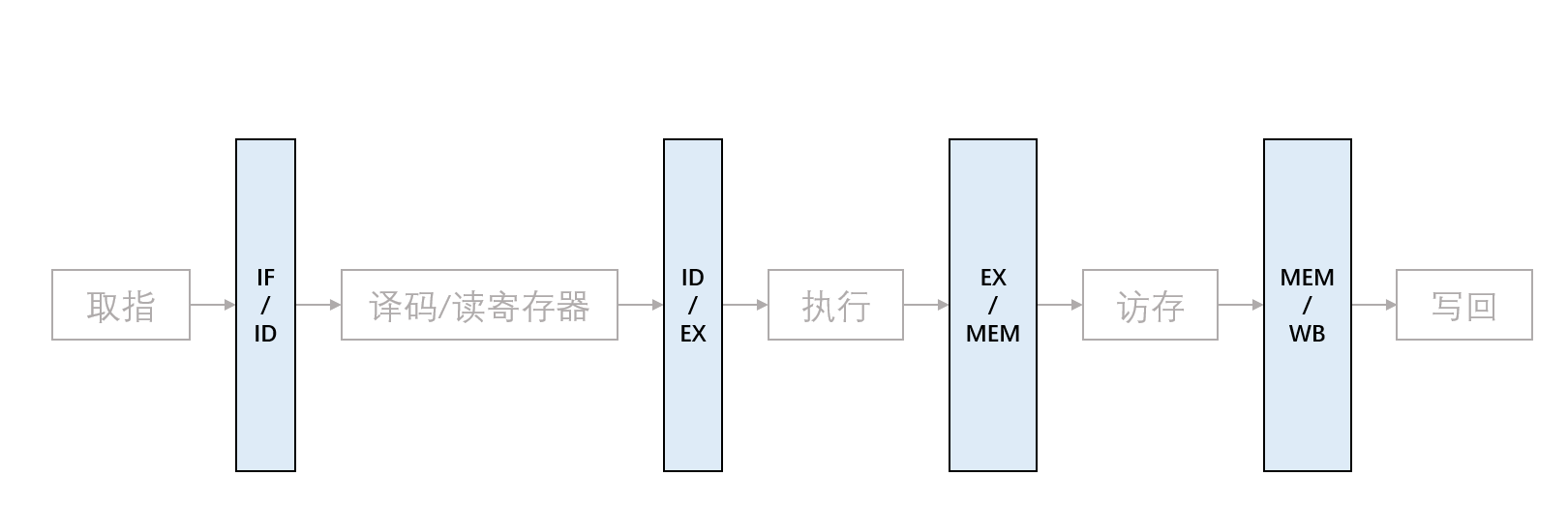


图5.2-2 流水效果的实现（触发器缓冲区）冒险与解决办法

### 结构相关

结构相关是指多条指令进入流水线后再同一机器周期内争用同一功能部件所发生的冲突，例如，冯·诺伊曼结构让取指阶段和访存阶段对内存有一个竞争的冲突。但由于在本实验中，我们采用了哈佛结构，因此不考虑结构相关的内容。

### 数据相关

由于流水线CPU的机制可以让多条指令同时处于CPU的各个阶段，这就导致在后面的指令进入流水线后前面仍有未执行完成的指令（还未写回寄存器），所以后面的指令所能够看到的寄存器堆并非最新的，从而会导致读取错误的出现；这里就可能会涉及到三种数据相关的类型，但因为我们本次实现的流水线CPU是顺序执行的，因而只会产生RAW类型的数据相关，也就是在一条指令的写操作后面跟着另一条指令的读操作，并且读写的是同一个寄存器。

### Load延迟

由于Load类指令（比如lw,lb等）要从数据存储器中读出数据才能得到写回的数据，而其在ALU执行阶段是无法得到写回数据的，所以要等到指令到达了访存阶段才能够进行数据前推。

### 控制相关

由于流水线的阻隔，导致跳转指令的跳转控制信号不能够及时到达PC指导跳转，所以存在一个时钟周期的延时。

# 设计步骤

## verilog代码编写

### CPU五级设计

#### 取指阶段

该模块包含两个文件：pc，inst\_mem(指令寄存器)。pc.v接收并锁存新的指令地址(来源于br\_unit)，并传递给指令寄存器下一条读取的指令地址；inst\_mem(指令寄存器)根据接收得来的地址，输出指令。

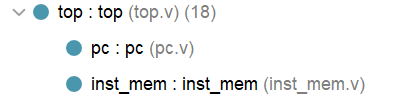


图6.1.1.1-1取指阶段有关代码文件

#### 译码/分析阶段

##### 译码处理

if\_id.v文件负责译码处理。根据三种指令类型: R型，I型，J型，截取32位指令中某些连续位传递给不同的模块。由于包含BGEZ和BLTZ命令(通过opcode和rt区分)，所以需要提取20-16位传递给control单元。

##### 跳转分支

这部分由beq\_judge，br\_unit负责。beq\_judge.v计算比较结果，见图6.1.1.2-2，并将比较结果输出传给control模块，control模块根据比较结果和对应的MIPS指令，输出控制信号ctrl\_br给模块br\_unit。br\_unit根据控制信号ctrl\_br，生成下一条指令地址，传给pc模块。

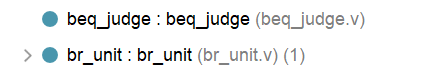


图6.1.1.2-1 跳转分支相关文件



图6.1.1.2-2 比较结果

##### 其他处理

这部分还包括ctrl模块，reg\_file模块，sign\_extension模块，sign\_extension模块，id\_ex模块。ctrl模块根据表5.1.2-2生成相应的控制信号；模块reg\_file为两读一写的寄存器堆；模块sign\_extension负责符号扩展，分为有符号扩展， 无符号扩展和取出移位量并进行相应的扩展，具体处理方式见图6.1.1.2-3；Id\_ex为取值/分析阶段和执行阶段的缓冲，保存下一个数据流，传递当前数据流。



图6.1.1.2-3 符号扩展处理

#### 执行阶段

执行阶段由alu，ex\_mem，mux\_reg\_addr，mux\_num1，mux\_num2等组成。alu实现算术运算和逻辑运算，32条指令(除去跳转指令)可归为13类运算，见图6.1.1.3-1，运用verilog自带的运算符进行运算，其中操作数来源于mux\_num1和mux\_num2；模块ex\_mem与模块id\_ex类似，皆为缓冲；模块mux\_reg\_addr，选择存入数据的寄存器地址来源，可来源于rt，来源于rd和来源于31号寄存器。



图6.1.1.3-1 13种运算对应的ctrl\_alu\_op值

#### 访存阶段

这部分主要由data\_mem(数据存储器)负责。控制信号ctrl\_mem\_w控制写入或读取数据，控制信号控制存取数据大小，按照小端方式进行存取。图6.1.1.4-1显示如何写入数据；图6.1.1.4-2显示如何读取数据，注意半字和字节大小的数据需扩展为32位数据。



图6.1.1.4-1 字/半字/字节写入



图6.1.1.4-2字/半字/字节读取

#### 回写阶段

回写阶段由mem\_web和mux\_reg\_wdata组成。模块mem\_web，实现访存与回写阶段之间的寄存器，将执行阶段的结果在下一个时钟周期传递到回写阶段；模块mux\_reg\_wdata，选择最终写回数据的数据来源，可来源于alu的计算结果，从数据存储器取出的数据，来源于跳转指令后面的第二条指令的地址(jal指令)。

### 流水效果的实现

为了实现流水效果，在原有代码基础上添加缓冲区部分，作为示例，这里给出访存和写回阶段间的缓冲区定义，如图6.1.2-1所示。



图6.1.2-1 MEM-WB缓冲区定义

### 冒险与解决办法

#### 数据相关

在本实验中，我们采用了“设计旁路，进行数据前推”的方法来解决这种类型的数据相关；简单来说，就是通过创建一个hazards\_unit模块，分别将处于执行阶段，访存阶段和写回阶段的前三条指令的写回数据，写回地址，以及相关的控制信号数据前推，作为输入传送到hazards\_unit模块；而在hazards\_unit模块，通过将各级的写回地址同当前需要访问寄存器的地址进行比对，并通过控制信号is\_data\_hazard的值，判断是否发生数据相关，如果发生数据相关，便将更新后的寄存器的值作为输出，传送给寄存器模块（如图6.1.3.1-1，图6.1.3.1-2，图6.1.3.1-3所示），此时访问寄存器的指令的操作数将选择为后面各级对应前推的写回数据，而放弃从寄存器中读出的数据。

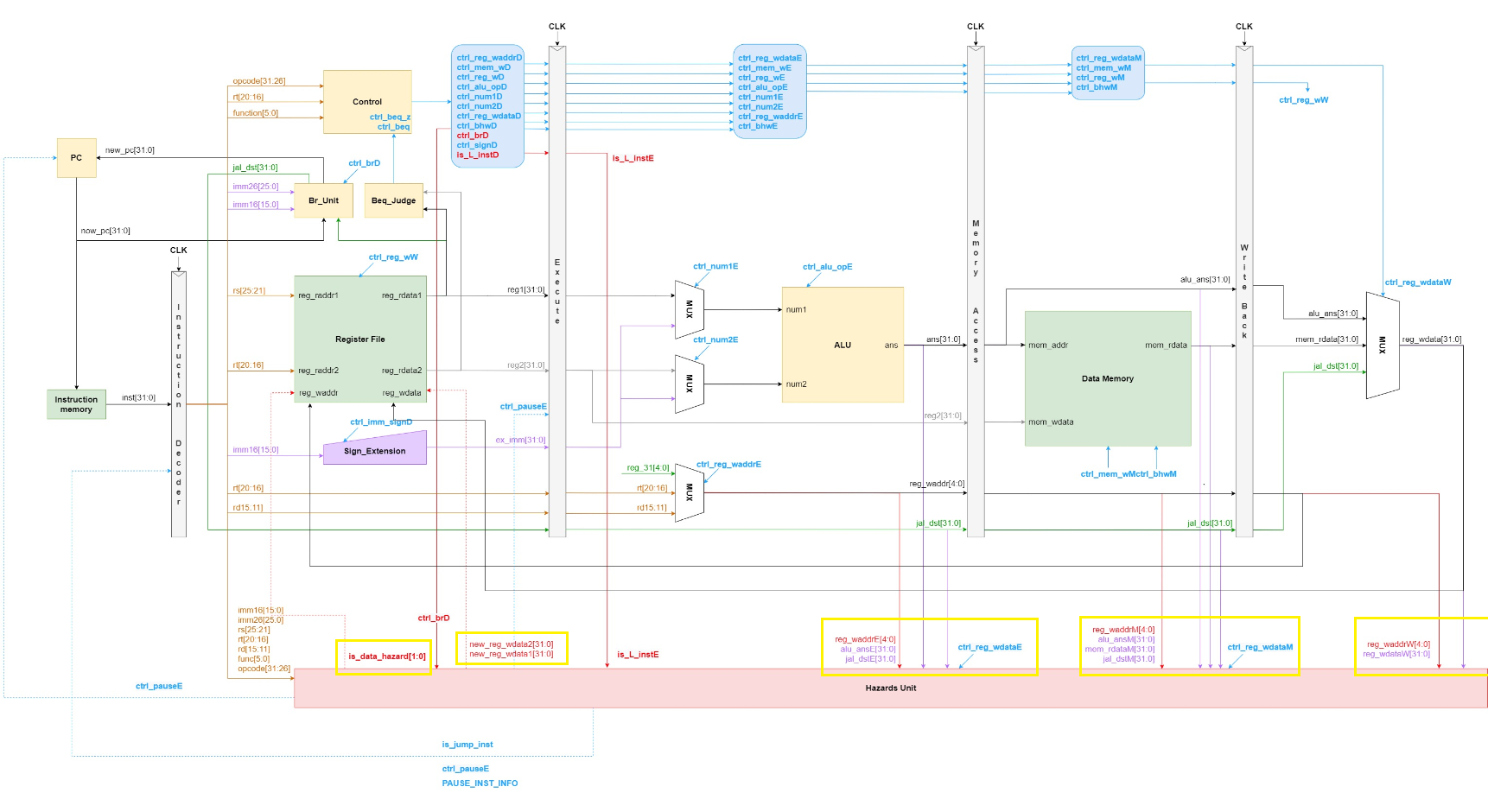


图6.1.3.1-1 hazards\_unit模块相关数据通路设计



图6.1.3.1-2 hazards\_unit模块中相关代码

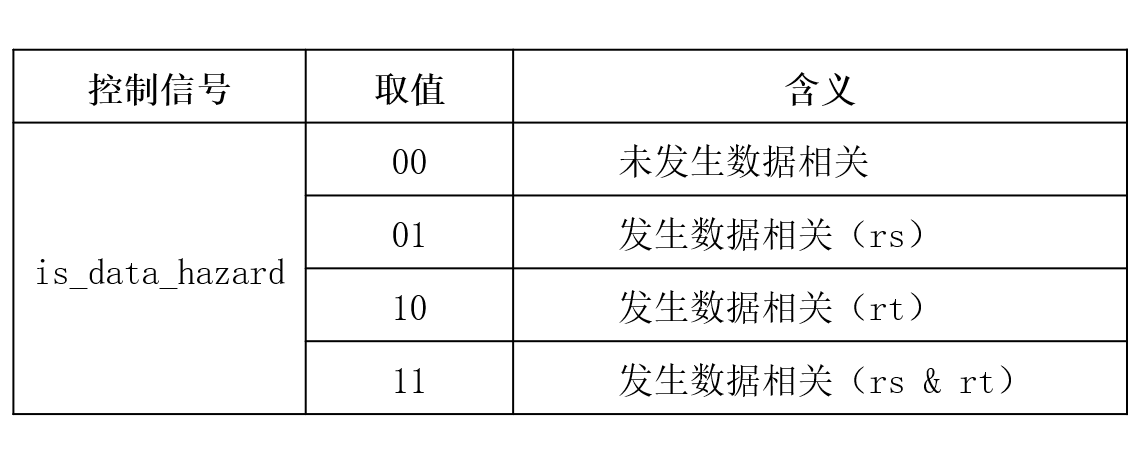


图6.1.3.1-3 is\_data\_hazard取值含义解释

#### Load延迟

在本实验中，我们采用了“阻塞暂停+数据前推”的方式来解决该问题；即当hazardz\_unit模块检测到load类指令的数据相关时（如图6.1.3.2-1所示），流水线从此处暂停，并向后传送一个空拍（nop），等待load类指令到达访存阶段再进行数据前推。



图6.1.3.2-1 hazards\_unit模块中相关代码

#### 控制相关

在本实验中，我们这里采用了插入空周期来解决控制相关，也就是当hazardz\_unit模块检测到取指阶段取出的指令检测为跳转类指令时（如图6.1.3.3-1所示），维持住当前PC的值不变，并且向下传送一个空周期（nop）；

****

图6.1.3.3-1 hazards\_unit模块中相关代码

## 下板测试

在前面我们进行了41条指令的逐条测试，为了更加直观的展示测试结果，我们选取了一个较为复杂的例子（涉及逻辑运算，算术运算，分支跳转等多种类型的指令，并且涉及到了数据相关，控制相关等冒险问题）进行下板验证。由于在本实验中，我们并没有实现除法指令，因此我们的测试程序的功能即——不使用除法指令，实现除法运算，并将相应的运算结果在数码管上进行显示，其中数码管前四位显示商，后四位显示余数（以十六进制的形式显示）。

****

图6.2-1 reg\_file模块中相关代码

****

图6.2-2 top模块中相关代码

****

图6.2-3 soc模块中相关代码

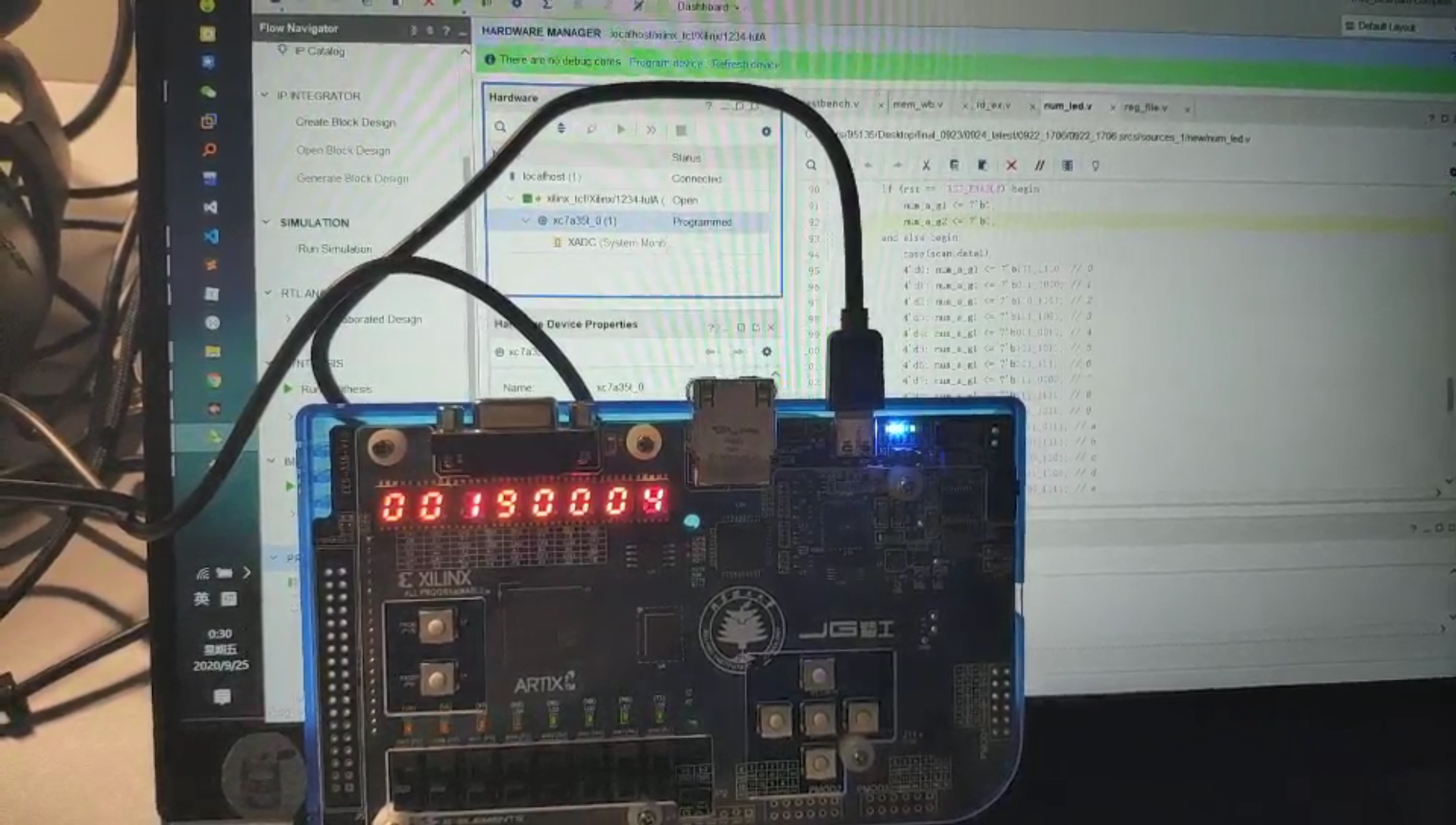


图6.2-4 下板测试结果（129/5 = 商25余4即16进制19和4）

# 问题及解决方法

1. 在最开始处理数据相关问题时，仅考虑到了某一个寄存器内容已经被更新，并未考虑到如 add $t3, $t2, $t1,寄存器 t1，t2 均被修改的情况，所以将控制信号is\_data\_dazard扩展至2位，用00，01，10，11来代表不同的数据相关情况。
2. 在进行下板测试时，由于数码管位数有限，而我们需要显示的结果是以32位数的形式保存的，因此，我们采用了十六进制来显示最后的结果。
3. jal指令中要存入31号寄存器的指令是当前指令的下一条还是下2条。由于网络和书本资料相差较大，经过测试表明应该为下2条指令。
4. 加入数据存储器的IP核时，发现多出2个延迟，由于时间关系，未解决，转为选择初始时的手写数据存储器。

# 心得体会及总结

通过本次实验，一方面，我在实验的过程中，深切地体会到了合作的重要性，无论是小组分工，进度把控，还是板块对接，汇报演示等都是需要每个小组成员参与其中并且及时进行协商调整的，而只有配合的过程好了，最终出来的成果才能是令人满意的。另一方面，我对流水线CPU的实现及其相关冒险冲突问题的解决都有了更加全面和深刻的理解，并且在小组实验的过程中发现了很多在单周期CPU实验中可以进一步优化和完善的地方。

通过设计和编写五级流水线CPU，我对CPU如何运行，如何与外设进行数据和控制信号流通有了更深的理解。短短两周里，虽然完成了一个基本的可运行41条指令的五级流水线CPU，但遗憾是最后未合上中断异常处理以及数据存储器IP核，希望之后的小组任务能加强组员之间的沟通和代码编写能力。

# 参考文献

1. David A.Patterson, John L. Hennessy.郑纬民等译.计算机组成与设计 硬件/软件接口[M].机械工业出版社.
2. 雷思磊.自己动手写CPU[M]. 北京:电子工业出版社.