

**综合实验报告**

( 2021—2022 年度第 2 学期)

名 称： 计算机组成与结构综合实验

题 目： 16位CPU设计

院 系： 计算机系

班 级： 软件工程2001

学 号： 220201090123

学生姓名： 杨照

实验课序号： 2

指导教师： 王晓霞、徐伟峰、张铭泉

设计周数： 1

成 绩：

日期： 2022年 5 月30 日

**说明：**

1、本组组员：杨照（本人）、陈子齐、龙治贤

2、分工：杨照----负责代码编写工作

陈子齐----协助调试、修改代码

龙治贤----查阅相关资料

3、申请成绩：优

**《计算机组成与结构》综合实验**

**任 务 书**

**一、 目的与要求**

1. 目的
   1. 熟悉VHDL硬件描述语言及开发环境，了解硬件系统开发的基本过程。
   2. 掌握ALU基本设计方法和运算器的数据传送通路。
   3. 掌握内存（RAM）的访问时序和方法。
   4. 理解总线数据传输的基本原理。
   5. 理解控制器的功能，加深对多周期CPU的控制器与指令周期的理解。
2. 要求
   1. 运用所学的知识和方法采用最佳方案完成实验设计。
   2. 编写简洁代码完成实验要求内容。
   3. 认真书写实验报告，包括实验目的、实验内容、实验设计方案、实验仿真结果等。

**二、 主要内容**

1. 16位运算器设计实验
   1. 用VHDL语言实现一个简单的16位ALU。
2. 通过实现一个状态机，根据状态机状态的变化来输入操作数及操作码，并最终实现不同的运算，将结果和标志位呈现出来。
3. 实验中ALU要求实现基本的算术运算、逻辑运算、移位运算等，具体功能如下表所示。合理设置每条指令的标志位，包括进位标志、溢出标志、最高位和全零标志。

|  |  |  |
| --- | --- | --- |
| 操作码 | 功能 | 描述 |
| ADD | A + B | 加法 |
| SUB | A – B | 减法 |
| AND | A and B | 与 |
| OR | A or B | 或 |
| XOR | A xor B | 异或 |
| NOT | not A | 取非 |
| SLL | A sll B | 逻辑左移B位 |
| SLR | A srl B | 逻辑右移B位 |
| SAL | A sal B | 算数左移B位 |
| SAR | A sar B | 算数右移B位 |
| ROL | A rol B | 循环左移B位 |
| ROR | A ror B | 循环右移B位 |

要求：ALU的数据输入A、B的长度为16位，操作码op为4位，算术运算时数据用补码表示。

1. 将实验过程中进行的操作与结果数据记录在下表中。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 输入数据 | | | 实际输出 | | 与预期一致性 |
| 操作码 | 操作数A | 操作数B | 运算结果 | 标志位 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

* 1. 通过仿真波形和实验板验证ALU的功能。
  2. 思考题

1. ALU进行算术逻辑运算所使用的电路是组合逻辑电路还是时序逻辑电路？
2. 如果给定了A和B的初值，且每次运算完后结果都写入到B中，再进行下次运算。这样一个带暂存功能的ALU要增加一些什么电路来实现？
   1. 实验拓展，实现ADC和SBB指令。
3. 存储器实验

使用教学计算机上的FPGA芯片，设计一个状态机和内存读写逻辑，完成对存储器RAM的访问。

* 1. 具体要求：

1. 写RAM1。将手拨开关上的数据，写入到RAM1的相应存储单元中，在LED灯上分别显示地址和数据。
2. 读RAM1。给定地址，能将数据从相应存储单元中读出，送到LED上显示。
   1. 具体步骤：
3. 定义输入信号：拨码开关决定读写地址或数据；绑定Ram1的数据线、地址线；绑定LED灯，用来显示从内存中读出的数据。
4. 定义状态机：读写状态控制的状态机：控制当前控制器是处于读状态还是写状态；内存读周期、写周期的状态机：根据时钟进行跳转，控制读写的过程。
   1. 在实验过程中可以使用七段数码管监测状态机当前状态，发光二极管只有16个，可以只显示地址的低8位和数据的低8位。
   2. 思考题：静态存储器的读、写时序各有什么特点？
5. 16位CPU设计实验

实现一个基于MIPS指令集的CPU，数据总线16位，地址总线16位，具有8个16位的通用寄存器。指令包括R型指令（如ADDU，SUBU，NOT，OR等），I型指令（如LW，SW等），B型指令（如BEQZ，BNEZ等）和J型指令传送指令（如JR，JALR等），具体指令见实验指导书P22-P32。

* 1. 具体要求：

1. 至少完成7条指令，这些指令组合能实现相对独立的逻辑功能。其中必须包括R型指令，I型指令，B型或J型指令。
2. 按照取指、译码、执行、访存和写回五个工作周期，分析每条指令的指令流程。
3. 根据指令流程，设计每条指令的CPU数据通路，定义涉及的所有微操作控制信号。然后逐一合并数据通路，说明增加或减少处理器功能部件的理由。给出控制器的完整设计过程。
4. 编写VHDL程序实现CPU，并通过实验板验证。
   1. 思考题：设计完成后，给出每条指令输入后在数据通路中的执行过程。

**三、 进度计划**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **设计(实验)内容** | **完成时间** | **备注** |
| 1 | 16位运算器设计实验 |  |  |
| 2 | 存储器实验 |  |  |
| 3 | 16位CPU设计实验 |  |  |

**四、 设计（实验）成果要求**

1. 运算器设计实验
   1. 编程实现基本ALU，完成表中运算功能，给出正确的结果。
   2. 实验板验证应包括运算结果和标志位。
   3. 实验拓展，实现ADC和SBB指令。
2. 存储器实验
   1. 完成SRAM写，给定地址，写入相应数据。
   2. 完成SRAM读，给定地址，读出对应数据。
   3. LED和数码管正确显示数据及状态的变化。
3. CPU设计实验
   1. 设计完成7条指令。
   2. 给出完整的设计报告，包括基本部件设计，如寄存器组、特殊寄存器、多路选择器等；每一条指令的数据通路图，以及CPU总数据通路图；控制器的设计等。
   3. 实验板验证指令的正确性。
4. 总体要求
   1. 完成任务书所列主要内容，详细的写出实验报告，要求规范、认真，并及时提交实验报告。
   2. 杜绝报告抄袭行为，一经发现均判为不及格。

**五、 考核方式**

1. 总成绩＝考勤成绩+视频讲解成绩+实验报告成绩
   1. 分组完成实验，每组2~3人。
   2. 每人单独提交自己的实验报告和5分钟视频讲解。
   3. 报告中说明小组分工，自己完成的主要工作，并在留言中说明申请的成绩，如“优”、“良”、“中”等。
2. 成绩评定要求：
   1. “优”要求：实验1+实验2+实验3，需要集中答辩。
   2. “良”要求：实验1+实验2+实验3（部分指令）。
   3. “中”或“及格”要求：实验1+实验2。

学生姓名：杨照

指导教师： 王晓霞、徐伟峰、张铭泉

2022年4月15 日

**实验一 运算器设计实验**

1. **目的与要求**

1、目的：用VHDL语言实现一个简单的16位ALU。

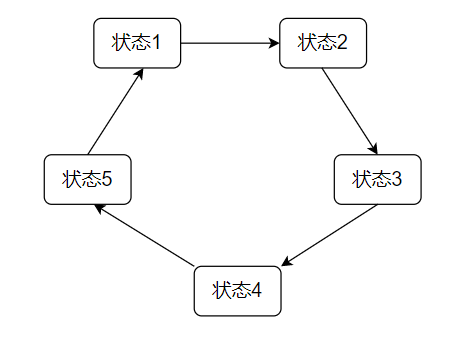
2、要求：

1. 通过实现一个状态机，根据状态机状态的变化来输入操作数及操作码，并最终实现不同的运算，将结果和标志位呈现出来。
2. 合理设置每条指令的标志位，包括进位标志、溢出标志、最高位和全零标志。
3. 将实验过程中进行的操作与结果数据记录在表中。

**二、实验正文**

1、实验内容

1. 通过实现一个状态机，根据状态机状态的变化来输入操作数及操作码，并最终实现不同的运算，将结果和标志位呈现出来。



1. 合理设置每条指令的标志位，包括进位标志、溢出标志、最高位和全零标志。

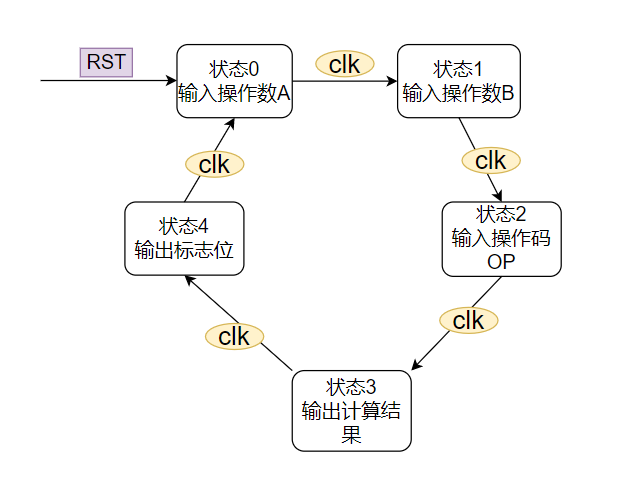
|  |  |  |
| --- | --- | --- |
| 操作码 | 功能 | 描述 |
| 0000 | A + B | 加法 |
| 0001 | A – B | 减法 |
| 0010 | A and B | 与 |
| 0011 | A or B | 或 |
| 0100 | A xor B | 异或 |
| 0101 | not A | 取非 |
| 0110 | A sll B | 逻辑左移B位 |
| 0111 | A srl B | 逻辑右移B位 |
| 1000 | A sal B | 算数左移B位 |
| 1001 | A sar B | 算数右移B位 |
| 1010 | A rol B | 循环左移B位 |
| 1011 | A ror B | 循环右移B位 |
| 1100 | A+B+CF | 带进位加法 |
| 1101 | A-B-CF | 带进位减法 |

1. 将实验过程中进行的操作与结果数据记录在表中。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A（十六进制）** | **B（十六进制）** | **OP（二进制）** | **结果（十六进制）** | **标志位（二进制）** |
| 0001H | 000EH | 0010B | 0000H | 0001H |
| 8000H | 0001H | 0110B | 0000H | 1101H |
| 0001H | 0001H | 1100B | 0003H | 0000H |

2、实验步骤

（1）确定五个状态分别输入操作数A、B，操作码OP，输出结果和标志位，图示如下：



（2）实现相关VHDL代码，下附部分功能代码：

|  |
| --- |
| if clock'event and clock='0' then  If state =0 then state:=1;a:=input;output<=input;  elsif state=1 then state:=2;b:=input;output<=input;  elsif state=2 then state:=3;op:=input(3 downto 0);output<=input;  elsif state=3 then state:=4;output<=y;  elsif state=4 then state:=0;  OUTPUT<="000000000000" & cflag & oflag & sflag & zflag;  when "0010"=> y:=a and b; --与  cflag:='0';oflag:='0';  sflag:=y(15);  if y="0000000000000000" then zflag:='1';else zflag:='0';end if;  when "0110"=> y:=to\_stdlogicvector(to\_bitvector(a) sll (conv\_integer(b))); --逻辑左移b位  cflag:=a(16-conv\_integer(b));  oflag:=a(15) xor y(15);  sflag:=y(15);  if y="0000000000000000" then zflag:='1';else zflag:='0';end if;  when"1100"=>yc:=('0'&a)+('0'&b)+("0000000000000000"&cflag); --带进位加法  y:=yc(15 downto 0);  cflag:=yc(16);  oflag:=(a(15) and b(15) and not y(15))or(not a(15) and not b(15) and y(15));  sflag:=y(15);  if y="0000000000000000" then zflag:='1';else zflag:='0';end if; |

（3）代码调试无误后，添加管脚文件，将代码下载到教学芯片中

（4）在 THINKPAD教学机上运行时， RST和时钟均用手动开关。操作码和操作数在

开关 SWO~ SW15 上输入；为便于观察和调试，每次 ALU 得到操作数，都会在

LED灯上显示一下，便于确认是否操作成功。最后的结果和标志位都在L0~L15

上显示

（5）记录实验结果

**三、综合实验总结**

1. 实验难点

（1）实验环境的使用以及硬件连接的操作流程。

（2）状态机的设计。

1. 心得体会

通过本次实验我基本掌握了VHDL语言的基本语法，学会了ISE以及Vivado等软件的使用方法，深也更加深层地体会到了ALU的工作流程，感受到了硬件知识的魅力，从刚上手的无头绪，到仔细看实验指导书和查阅相关资料后的恍然大悟，体验了由难到易的过程，获得了攻坚克难后的喜悦感。

**实验二 存储器实验**

**一、目的与要求**

1、目的

使用教学计算机上的FPGA芯片，设计一个状态机和内存读写逻辑，完成对存储器RAM的访问。

2、要求

1. 写RAM1。将手拨开关上的数据，写入到RAM1的相应存储单元中，在LED灯上分别显示地址和数据。
2. 读RAM1。给定地址，能将数据从相应存储单元中读出，送到LED上显示。

**二、实验正文**

1、实验内容

对于存储器芯片的访问，首先要熟悉存储器芯片的访问时序，并了解THINPAD教学机上存储器芯片与FPGA芯片的具体连接方式，也就是说，如何在THINPAD教学机上具体实现对RAM芯片的访问时序。

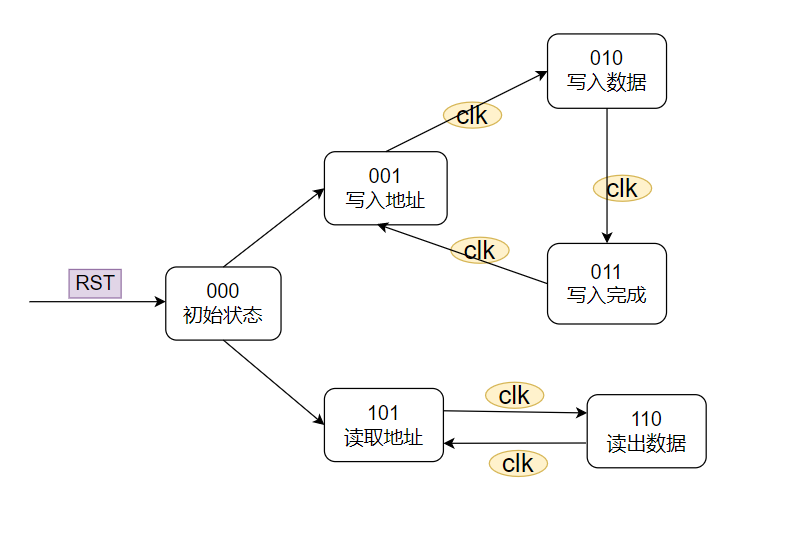
从时序要求上看，读取SRAM时要提前准备好地址，并将数据线设置成高阻，然后就可以读出数据了；写SRAM时要提前准备好地址数据，然后将写信号拉低即可将数据写入相应地址。

要完成实验，就要给内存芯片提供地址、数据和控制信号。初始地址和数据均来自拨码开关，需要有寄存器接受并保存好，然后分别送到地址总线和数据总线。控制信号由3个，以RAM1为例，分别是RAM1\_EN、RAM1\_OE和RAM1\_RW,对应连接到RAM芯片的CE\OE\WE管脚，

需要根据对存储芯片的访问要求来正确设置。

2、实验步骤

（1）定义状态机，分布完成各个操作



（2）编写相关VHDL代码，部分代码如下：

|  |
| --- |
| elsif state="001" then state<="010"; -- 写地址  en<='1';  oe<='1';  rw<='1';  add<="00"&input;output<=input;  tadd<=input;  elsif state="010" then state<="011"; -- 写数据  data<=input;output<=input;  elsif state="011" then state<="001"; -- 显示地址、数据  en<='0';  oe<='1';  rw<='0';  output<=tadd(7 downto 0)&data(7 downto 0);  elsif state="101" then state<="110"; -- 读地址  en<='1';  oe<='1';  rw<='1';  add<="00"&input;output<=input;  elsif state="110" then state<="101"; -- 读出数据  en<='0';  oe<='0';  rw<='1';  output<=data; |

（3）代码调试完成，编写相关管脚文件，并烧入实验板

（4）验证实验，验证结果如下：

|  |  |  |
| --- | --- | --- |
| **写入地址** | **写入数据** | **显示输出（地址、数据低八位）** |
| 0001H | 000FH | 010FH |
| 0002H | 00F0H | 02F0H |

|  |  |  |
| --- | --- | --- |
| **读地址** | **读出数据** | **读写是否一致** |
| 0001H | 000FH | 是 |
| 0002H | 00F0H | 是 |

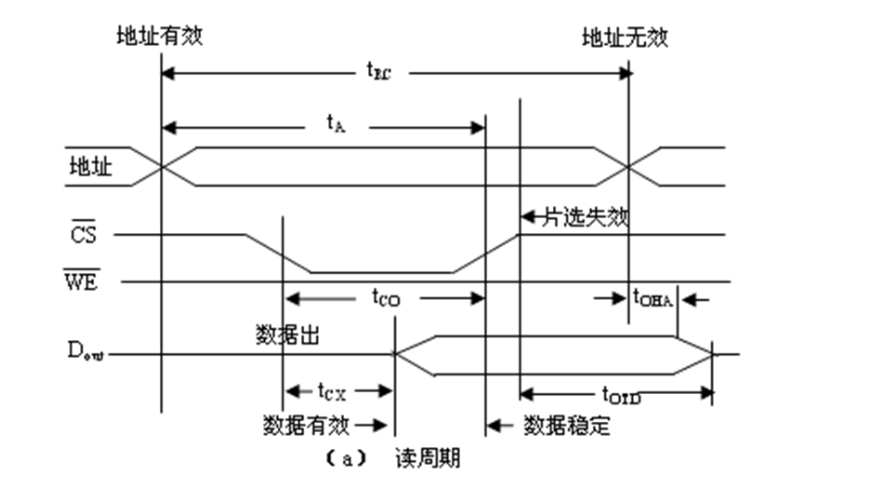
**三、综合实验总结**

1. 实验难点

对于板子上存储器的访问，我们发现在读内存时读出的数据并不会随着地址的改变而改变，根据实验指导书上的从吋序要求上看，在读取 SRAM 吋提前准备好地址，并将数据线设置成高阻，但问题没有得到解决，根据老师的提示，访问存储器时占用地址总线，需要设置总线控制信号DBC=1。

1. 心得体会

通过本次实验我学会了访问存储器的原理，实验刚开始的时候由于看错了实验任务书误以为是要自己实现一个存储器而不知道是访问THINPAD上的ram。可见看问题要严谨，刚开始就理解错了题意，花再多的时间都是白费。在实验过程中我们遇到了问题：读出的数据并不会随着地址的改变而改变，最后根据老师的提示得到了问题的解决方案，才最终完成该实验。

**四、思考题**

1.读操作时，必须保证片选信号为低电平，读写信号为高电平。

tRC （读周期时间）:指对芯片连续两次读操作之间的最小间隔时间。

tA (读出时间):从给出有效地址后，经过译码电路、驱动电路的延迟，到读出所选单

元内容，并经I/O电路延迟，直到数据在外部数据总线上稳定出现所需的时间。显

然，读出时间小于读周期时间。

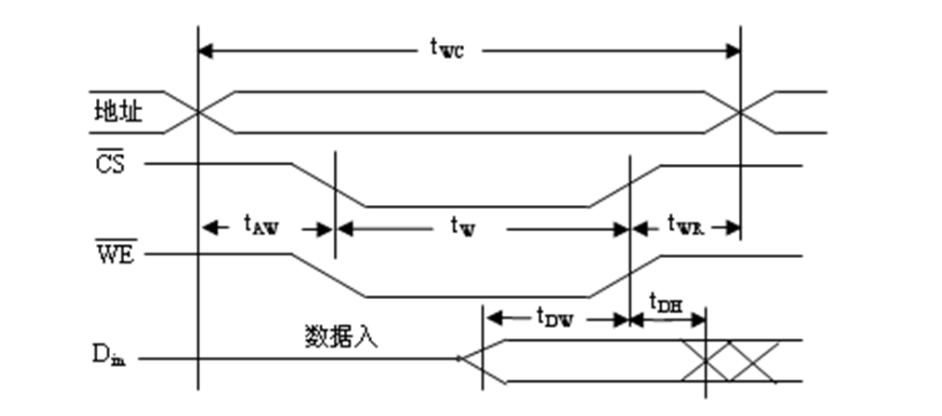
tCO (片选到数据输出稳定的时间):数据能否送到外部数据总线上，不仅取决于地

址，还取决于片选信号。因此，tCO是从有效到数据稳定出现在外部数据总线上的时

间。

tCX（片选到数据输出有效时间）从片选有效到数据开始出现在数据总线上的间隔时间。

tOTD：片选无效后数据还需在数据总线上保持的时间。

tOHA：地址失效后，数据线上的有效数据维持时间，以保证所读数据可靠。

2.写操作时，为低电平，读写信号为低电平。

tW（写入时间）：为保证数据可靠地写入，与同时有效的时间必须大于或等于tW。

tAW（滞后时间）：地址有效后，必须经过tAW时间，WE/信号才能有效（低），否则可能产生写出错。

tWR（写恢复时间）：WE/无效后，经tWR时间后地址才能改变，否则也可能错误地写入。

tDW：写入数据必须在写无效之前tDW时间就送到数据总线上。

tDH：WE/无效后，数据还要保持的时间。此刻地址线仍有效，tWR>tDH，以保证数据可靠写入。

tWC（写周期时间）:表示连续两次写操作之间的最小时间间隔。tWC = tAW + tW +tWR。

**实验三 16位CPU设计实验**

**一、目的与要求**

1、目的

实现一个基于MIPS指令集的CPU，数据总线16位，地址总线16位，具有8个16位的通用寄存器。指令包括R型指令（如ADDU，SUBU，NOT，OR等），I型指令（如LW，SW等），B型指令（如BEQZ，BNEZ等）和J型指令传送指令（如JR，JALR等）

2、要求

（1）至少完成7条指令，这些指令组合能实现相对独立的逻辑功能。其中必须包括R 型指令，I型指令，B型或J型指令。

（2）按照取指、译码、执行、访存和写回五个工作周期，分析每条指令的指令流程。

（3）根据指令流程，设计每条指令的CPU数据通路，定义涉及的所有微操作控制信号。然

后逐一合并数据通路，说明增加或减少处理器功能部件的理由。给出控制器的完整设

计过程。

（4）编写VHDL程序实现CPU，并通过实验板验证。

**二、实验正文**

1、实验内容

（1）设计完成7条指令：ADDIU、LW、LI、ADDU、BNEZ、SW、JR

（2）给出完整的设计报告，包括基本部件设计，如寄存器组、特殊寄存器、多路选择 器等；每一条指令的数据通路图，以及CPU总数据通路图；控制器的设计等。

（3）实验板验证指令的正确性

2、实验步骤

（1）指令设计

ADDIU指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令编码 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | rx | | | immediate | | | | | | | |
| 指令格式 | ADDIU rx immediate | | | | | | | | | | | | | | | |
| 指令功能 | R[x] ← R[x]+Sign\_extend (immediate) | | | | | | | | | | | | | | | |
| 功能说明 | 对立即数 immediate 进行符号扩展后与寄存器 rx 的值求和，结果保存到寄存器 rx 中 | | | | | | | | | | | | | | | |

LW指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令编码 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | rx | | | ry | | | immediate | | | | | |
| 指令格式 | LW rx ry immediate | | | | | | | | | | | | | | | | |
| 指令功能 | R[y]←MEM[R[x]+Sign\_extend (immediate)] | | | | | | | | | | | | | | | | |
| 功能说明 | 从内存中读取数据到寄存器 ry 中，内存地址为寄存器 rx 的内容与立即数（进行符号扩展后)immediate 之和 | | | | | | | | | | | | | | | | |

LI指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令编码 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | rx | | | immediate | | | | | | | |
| 指令格式 | LI rx immediate | | | | | | | | | | | | | | | |
| 指令功能 | R[x]←Zero\_Extend (immediate) | | | | | | | | | | | | | | | |
| 功能说明 | 将立即数 immediate 的值(进行零扩展后)存储到寄存器 rx 中 | | | | | | | | | | | | | | | |

ADDU指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令编码 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | rx | | | ry | | | rz | | | 0 | 1 |
| 指令格式 | ADDU rx ry rz | | | | | | | | | | | | | | | |
| 指令功能 | R[z]←R[x]+R[y] | | | | | | | | | | | | | | | |
| 功能说明 | 将寄存器 rx 与寄存器 ry 的值求和， 结果保存到寄存器 rz 中 | | | | | | | | | | | | | | | |

BNEZ指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令编码 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | rx | | | immediate | | | | | | | |
| 指令格式 | BNEZ rx immediate | | | | | | | | | | | | | | | |
| 指令功能 | if (R[x]！=0) then PC←PC+Sign\_extend(immediate) | | | | | | | | | | | | | | | |
| 功能说明 | 若寄存器 rx 的值不为 0,则跳转到目的地址执行；否则顺序执行下一条指令 | | | | | | | | | | | | | | | |

SW指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令编码 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | | 6 | 5 | 4 | | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | rx | | | | ry | | | | immediate | | | | |
| 指令格式 | SW rx ry immediate | | | | | | | | | | | | | | | | | |
| 指令功能 | MEM[R[x]+Sign\_extend (immediate)]R[y]， | | | | | | | | | | | | | | | | | |
| 功能说明 | 将寄存器 ry 的值写入到内存中,内存地址为寄存器 rx 的内容与立即数（进行符号扩展后)immediate 之和 | | | | | | | | | | | | | | | | | |

JR指令

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令编码 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | rx | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 指令格式 | JR rx | | | | | | | | | | | | | | | | |
| 指令功能 | PC←R[X] | | | | | | | | | | | | | | | | |
| 功能说明 | 程序无条件跳转到rx地址单元执行，用于长地址跳转 | | | | | | | | | | | | | | | | |

（2）总体结构

* 寄存器

1)通用寄存器：

r0~r7 (000~111)

2)基本寄存器：

指令寄存器IR: 存放现行指令

程序计数器 PC: 存放 CPU 要执行的下一条指令的地址

数据寄存器 MBR: 实现CPU与主存的接口

地址寄存器 MAR: 实现CPU与主存的接口

暂存器C: 暂存来自主存的源地址或源数据

暂存器D：暂存来自主存的目的地址或目的数据

堆栈指针寄存器：SP

程序状态字寄存器PSW

* 运算部件

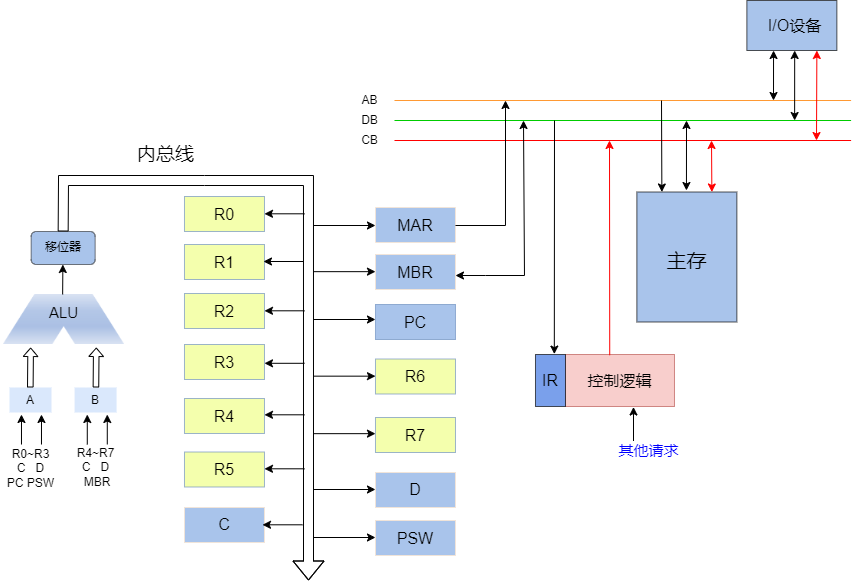
1)ALU：SN74181 四片，SN74182 一片

2)移位器：实现左移，右移，直送，字节交换

3)选择器A，选择器B：选择数据来源

* 控制器CU

控制器模块由状态机实现，控制器依据指令、数据通路通过控制线向各功能模块发出控制命令。

（3）数据通路

（4）部分指令的数据通路分析

**ADDU:** ADDIU、LW、LI、ADDU、BNEZ、SW、JR

1. 取指令阶段:需要从程序寄存器PC中取出当前指令地址送给指令存储器ir输入端,然后从指令存储器数据输出端口得到指令內容以便下一周期译码,同时要将PG当前内容加1送给PC寄存器,使其指向下一条指令
2. 译码阶段:需要从指令中得到Rx寄存器和Ry寄存器的编号,从输出端A、B取出其中内容即源操作数,送给运算器输入端以便于下一步进行加法运算,同时需要从指令中取出Rz寄存器编号作为目的地址,以便之后将运算器运算结果送给寄存器组的输入端,写入寄存器Rz
3. 执行阶段:A、B通过运算部件ALU进行加法运算
4. 写回阶段:将输出端得到的运算结果送给寄存器组的输入端,写入寄存器Rz

**LW:**

1. 取指令阶段：需要从程序寄存器PC中取出当前指令地址送给指令存储器ir,然后从指令存储器数据输出端口得到指令内容以便下一周期译码，同时将PC加1送给PC，指向下一条指令。
2. 译码阶段：需要从指令中的第10位到第8位取出Rx寄存器编号，从输出端A取出Rx寄存器内容送给运算器A输入端，第4位到第0位取出5位立即数经过扩展送给运算器B输入端，同时还要将指令第7位到第5位送给Ry输入端作为目的寄存器编号。
3. 执行阶段：A、B输入的内容送给运算器进行加法运算，ALU输出内容送给数据存储器地址输入端addr。
4. 访存阶段：将 addr里的地址对应的数据从data里读出。
5. 写回阶段：数据存储器数据输出端D送给寄存器组数据写入端W，根据Ry寄存器编号，将数据写入Ry寄存器。

**SW:**

1. 取指令阶段：需要从程序寄存器PC中取出当前指令地址送给指令存储器ir，然后从指令存储器数据输出端口得到指令内容以便下一周期译码，同时将PC加1送给PC，指向下一条指令。
2. 译码阶段：需要从指令中的第10位到第8位取出Rx寄存器编号，从输出端A取出Rx寄存器内容送给运算器A输入端，第4位到第0位取出5位立即数经过扩展送给运算器B输入端，同时还要将指令第7位到第5位送给Ry输入端作为输入数据寄存器编号。
3. 执行阶段：运算器对A、B输入的内容进行加法运算，运算后结果为目的地址需要送给数据存储器地址端addr。
4. 访存阶段：输入数据，将其送给数据存储器数据输入端W。
5. 写回阶段：将数据存储器输入端W的数据写入目的地址addr。

**LI：**

1. 取指令阶段:需要从程序寄存器PC中取出当前指令地址送给指令存储器ir输入端,然后从指令存储器数据输出端口得到指令內容以便下一周期译码,同时要将PG当前内容加1送给PC寄存器,使其指向下一条指令.
2. 译码阶段：需要从指令中的第10位到第8位取出Rx寄存器编号，还要将第7位到第0位取出8位立即数再经过扩展。
3. 执行阶段：将立即数送给临时变量t3

写回阶段：将临时变量t3的值送到寄存器组输入端，写回寄存器Rx。

**三、综合实验总结**

1. 实验难点

CPU的设计相较于前两个实验更加具有综合性，难度也更大，我们也遇到了在做存储器时一样的问题（访问存储器时占用地址总线，需要设置总线控制信号DBC=1）为了简化实验，我们采取自己定义存储器的方式，而没有用到板子上的ram。

1. 心得体会

通过本次16位cpu的设计，我对计算机组成与结构课程的相关知识有了更全面、更深层的掌握，也对硬件的相关知识有了更深刻的理解，同时我的分析、解决问题的能力得到了极大的锻炼，我也认识到知识总是交叉的，比如想要完成本次CPU实验，如果只了解CPU的知识那是远远不够的，必须融会贯通，与其他相关知识联系起来，才能实现理想的结果。作为计算机专业的学生，我感觉通过本学期计算机组成与结构的学习、通过本次实验的锻炼，我才真正有底气跟别人说自己是计算机专业的学生了。感谢老师的细心教导与，才让我得以克服困难完成实验！

**四、思考题**

上面已经列出视频中展示的指令数据通路的执行过程。

**附录：CPU完整代码**

|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  entity cpu is  Port ( clk : in STD\_LOGIC;  rst : in STD\_LOGIC;  input : in STD\_LOGIC\_VECTOR (15 downto 0);  output : out STD\_LOGIC\_VECTOR (15 downto 0);  seg : out STD\_LOGIC\_VECTOR (13 downto 0))  end cpu;  architecture Behavioral of cpu is  signal state:STD\_LOGIC\_VECTOR (2 downto 0):="000";--八个周期  signal r0,r1,r2,r3,r4,r5,r6,r7:STD\_LOGIC\_VECTOR (15 downto 0):="0000000000000000";--通用寄存器  signal ir:STD\_LOGIC\_VECTOR (15 downto 0):="0000000000000000";--指令寄存器  signal pc:STD\_LOGIC\_VECTOR (15 downto 0):="0000000000010000";  --程序计数器（默认指令从0010H处开始存储）    subtype word is std\_logic\_vector(15 downto 0);  type memory is array(15 downto 0) of word;  signal ram:memory;-- 存储器（实为数组，长度为64K，存储元素为字）  procedure rr(a:in STD\_LOGIC\_VECTOR (2 downto 0);b:out STD\_LOGIC\_VECTOR (15 downto 0)) is--寄存器取值  begin  case a is  when "000"=>b:=r0;  when "001"=>b:=r1;  when "010"=>b:=r2;  when "011"=>b:=r3;  when "100"=>b:=r4;  when "101"=>b:=r5;  when "110"=>b:=r6;  when "111"=>b:=r7;  when others=>b:="0000000000000000";  end case;  end rr;    begin  process (clk,rst)  variable ram\_addr : STD\_LOGIC\_VECTOR (15 downto 0);--存储器地址段  variable ram\_data : STD\_LOGIC\_VECTOR (15 downto 0);--存储器数据段  variable mar:STD\_LOGIC\_VECTOR (15 downto 0):="0000000000010000";--地址寄存器  variable mbr:STD\_LOGIC\_VECTOR (15 downto 0):="0000000000000000";--数据寄存器  variable op:STD\_LOGIC\_VECTOR (4 downto 0):="00000";--操作码  variable rx,ry,rz:STD\_LOGIC\_VECTOR (2 downto 0):="000";--访问的寄存器  variable imme:STD\_LOGIC\_VECTOR (15 downto 0):="0000000000000000";--立即数  variable t1,t2,t3:STD\_LOGIC\_VECTOR (15 downto 0):="0000000000000000";--临时量  begin  if (rst='0') then --复位  state<="000";  output<="0000000000000000";  mar:="0000000000010000";  pc<="0000000000010000";  elsif (CLK'event and CLK = '1') then  case state is  when "000"=>--初始状态  case input(2 downto 0) is  when"001"=> state<="001";  when"011"=> state<="011";  when others =>  end case;  output<=input;  when "001"=> --向存储器写入数据和指令  output<=input;  ram\_addr:=input;  state<="010";  when"010"=>  ram\_data:=input;  output<=input;  ram(conv\_integer(ram\_addr))<=ram\_data;  state<="001";  when "011"=> --取指阶段  state<="100";  ram\_addr:=mar;  output<=mar; -- 输出指令地址  ram\_data:= ram(conv\_integer(mar));  ir<=ram\_data;  pc<=pc+1;    when "100"=> --译码阶段  state<="101";  output<=ir; --输出指令  op:=ir(15 downto 11);  case op is  when "01001"=> --ADDIU  rx:=ir(10 downto 8);  imme:="00000000"&ir(7 downto 0);  when"11100"=> --ADDU  rx:=ir(10 downto 8);  ry:=ir(7 downto 5);  rz:=ir(4 downto 2);  when "00101"=> --BNEZ  rx:=ir(10 downto 8);  imme:="00000000"&ir(7 downto 0);  when "10011"=> --LW  rx:=ir(10 downto 8);  ry:=ir(7 downto 5);  imme:="00000000000"&ir(4 downto 0);  when "01101"=> --LI  rx:=ir(10 downto 8);  imme:="00000000"&ir(7 downto 0);  when "11011"=> --SW  rx:=ir(10 downto 8);  ry:=ir(7 downto 5);  imme:="00000000000"&ir(4 downto 0);  when “11101”=> --JR  rx:=ir(10 downto 8);  when others=>  end case;  when "101"=> --执行阶段  state<="110";  case op is  when "01001"=> --ADDIU  rr(rx,t1);  t3:=t1+imme;  output<=t3;  when "11100"=> --ADDU  rr(rx,t1);  rr(ry,t2):  t3:=t1+t2;  when "00101"=> --BNEZ  rr(rx,t1);  if (not (t1="0000000000000000")) then  pc<=pc+imme;  end if;  output<=pc;  when "10011"=> --LW  rr(rx,t1);  mar:=t1+imme;  ram\_addr:=mar;  output<=ram\_addr;  when "01101"=> --LI  rr(rx,t1);  t3:=imme;  output<=t3;  when "11011"=> --SW  rr(rx,t1);  rr(ry,t2);  mar:=t1+imme;  ram\_addr:=mar;  mbr:=t2;  output<=ram\_addr;  when”11101”=> --JR  rr(rx,t1);  pc<=t1;  when others=>  end case;  when "110"=> --访存阶段  state<="111";  case op is  when "01001"=>  when "11100"=>  when "00101"=>  when "10011"=>--lw  ram\_data:=ram(conv\_integer(ram\_addr));  mbr:=ram\_data;  output<=mar(7 downto 0)&mbr(7 downto 0);  when "01101"=>  when "11011"=>--sw  ram\_data:=mbr;  ram(conv\_integer(ram\_addr))<=ram\_data;  output<=mar(7 downto 0)&mbr(7 downto 0);  when others=>  end case;  when "111"=>--写回阶段  state<="011";  mar:=pc;  case op is  when "01001"=> --ADDIU  case rx is  when "000"=>r0<=t3;  when "001"=>r1<=t3;  when "010"=>r2<=t3;  when "011"=>r3<=t3;  when "100"=>r4<=t3;  when "101"=>r5<=t3;  when "110"=>r6<=t3;  when "111"=>r7<=t3;  when others=>  end case;  output<="00000"& rx & t3(7 downto 0);  when”11100”=> --ADDU  case rz is  when "000"=>r0<=t3;  when "001"=>r1<=t3;  when "010"=>r2<=t3;  when "011"=>r3<=t3;  when "100"=>r4<=t3;  when "101"=>r5<=t3;  when "110"=>r6<=t3;  when "111"=>r7<=t3;  when others=>  end case;  output<="00000"& rz & t3(7 downto 0);  when "00101"=>  when "10011"=>--LW  case ry is  when "000"=>r0<=mbr;  when "001"=>r1<=mbr;  when "010"=>r2<=mbr;  when "011"=>r3<=mbr;  when "100"=>r4<=mbr;  when "101"=>r5<=mbr;  when "110"=>r6<=mbr;  when "111"=>r7<=mbr;  when others=>  end case;  output<="00000"& ry & mbr(7 downto 0);  when "01101"=>--LI  case rx is  when "000"=>r0<=t3;  when "001"=>r1<=t3;  when "010"=>r2<=t3;  when "011"=>r3<=t3;  when "100"=>r4<=t3;  when "101"=>r5<=t3;  when "110"=>r6<=t3;  when "111"=>r7<=t3;  when others=>  end case;  output<="00000"& rx & t3(7 downto 0);  when "11011"=>  when "11101"=>  when others=>  end case;  when others=>  state<="000";  end case;  end if;  end process;    process (state)  begin  case state is  when "000"=>seg<="01111110111111"; --00  when "001"=>seg<="01111110100001"; --01  when "010"=>seg<="01111111011011"; --02  when "011"=>seg<="01000010100001"; --11  when "100"=>seg<="01000011011011"; --12  when "101"=>seg<="01000011110011"; --13  when "110"=>seg<="01000011100101"; --14  when "111"=>seg<="01000011110110"; --15  when others=>  end case;  end process;    end Behavioral; |