**CPU设计报告**

1. **指令格式设计**

该cpu采用的是变长指令，部分指令为单字长（16位），部分指令为双字长（32位）。指令格式如下：

单字长：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 操作码 | | | | |  |  |  |  |  | 源寄存器 | | | 目的寄存器 | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

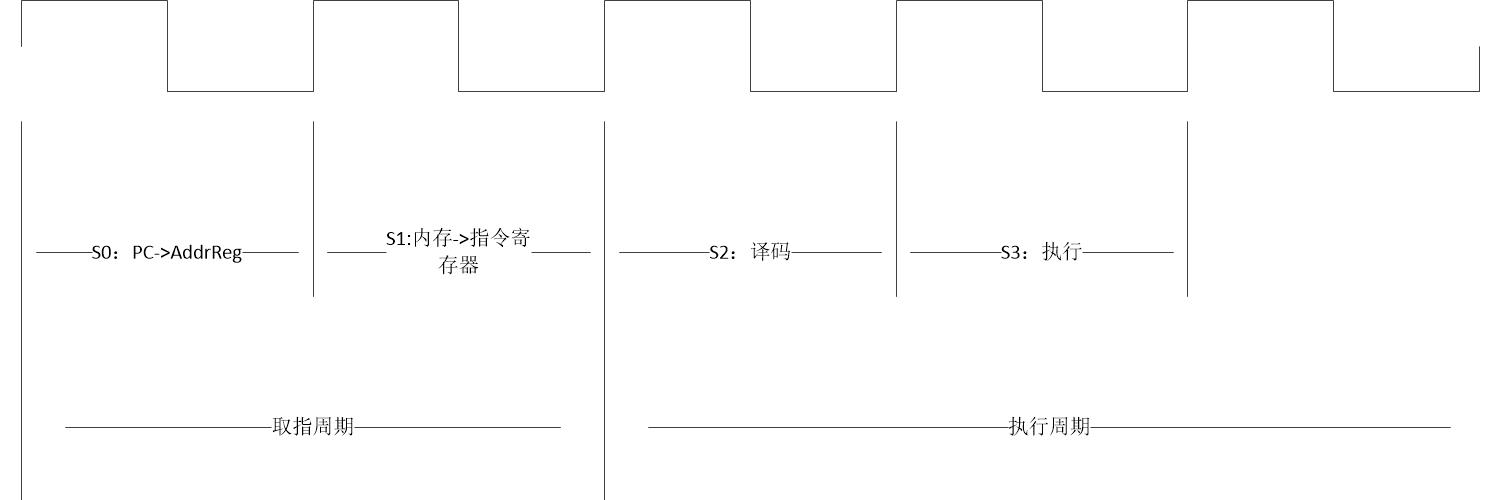
双字长：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 操作码 | | | | |  |  |  |  |  |  |  |  | 目的寄存器 | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 地址或立即数 | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

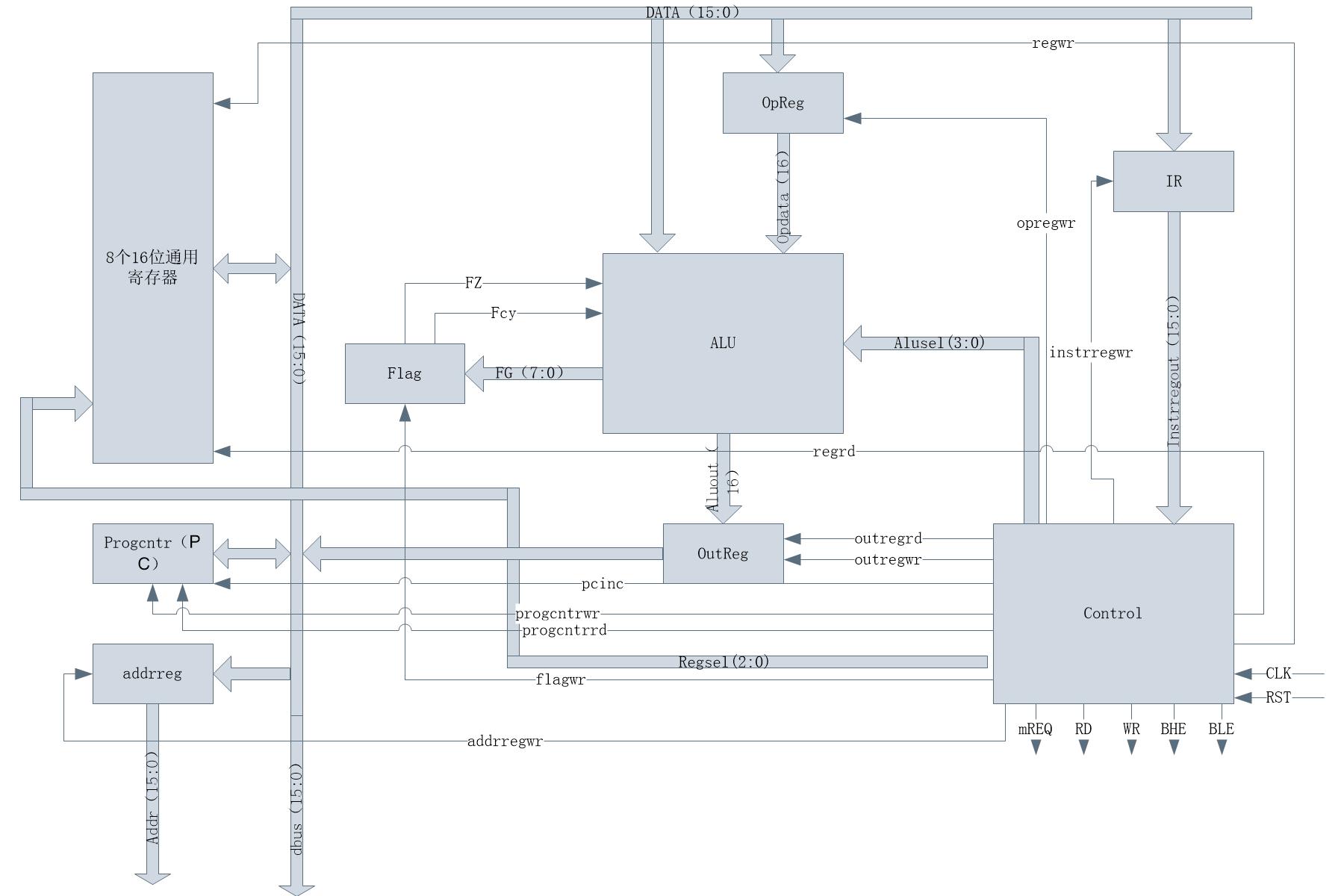
|  |  |  |
| --- | --- | --- |
| 指令 | 操作码 | 说明 |
| MOV | 00000 | 立即数寻址（双字长） |
| 00001 | 直接寻址（内存到寄存器，双字长） |
| 00010 | 直接寻址（寄存器到内存，双字长） |
| 00011 | 寄存器寻址（单字长） |
| 00100 | 间接寻址（单字长） |
| 00101 | 变址寻址（双字长） |
| ADC | 00110 | 立即数寻址（双字长） |
| 00111 | 寄存器寻址（单字长） |
| SBB | 01000 | 立即数寻址（双字长） |
| 01001 | 寄存器寻址（单字长） |
| AND | 01010 | 立即数寻址（双字长） |
| 01011 | 寄存器寻址（单字长） |
| OR | 01100 | 立即数寻址（双字长） |
| 01101 | 寄存器寻址（单字长） |
| CLC | 01110 | 0→Cy |
| STC | 01111 | 0→Z |
| JMP | 10000 | 直接寻址（双字长） |
| JZ | 10001 | Z=1时跳转（单字长） |
| JC | 10010 | Cy=1时跳转（单字长） |

1. **微操作的定义**
2. 取指令微操作：
   1. PC寄存器→内部数据总线→AddrReg地址寄存器
   2. PC+1；内存→外部数据总线→内部数据总线→ instr（指令寄存器）
3. 运算微操作：
   1. 第一个操作数→OpReg
   2. 第二个操作数→ALU
   3. ALU→OutReg
   4. OutReg→Reg
4. 访存微操作：
   1. 地址值→AddrReg（地址寄存器）
   2. 内存→数据总线→Reg 或者 ② Reg→数据总线→内存
5. **节拍的划分**

由于指令是变长的，所以并没有固定的指令周期，但前三个周期是一样的。



1. **处理器详细结构设计框图及功能描述**

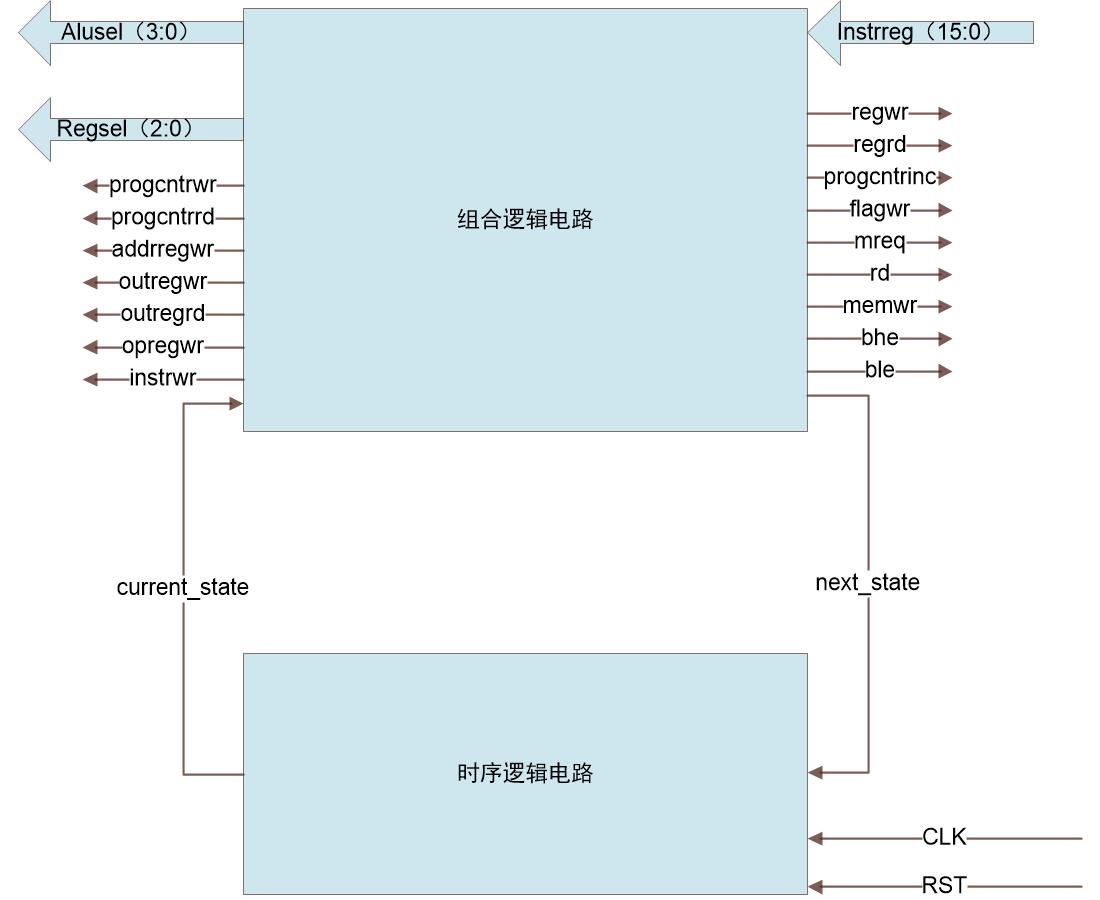


功能描述：

当复位信号无效时，CPU在时钟信号的控制下工作，在CLK的上升沿进行状态的跳转。取值周期的第一个周期里将PC的值通过数据总线给addrreg，在取值周期的第二个周期mREQ、RD、BHE、BLE有效，主存将数据放在数据总线上，instrregwr信号有效数据写入IR中，同时pcinc有效，完成PC+1工作。接下来是执行周期，执行周期里的第一个周期为译码周期，译码周期里对指令进行译码同时决定CPU的下一个状态。然后是多个执行微操作的执行周期，完成对指令的执行，最后回到取值周期的一个周期，开始取下一条指令并执行，然后继续重复这一过程直到执行完所有指令。

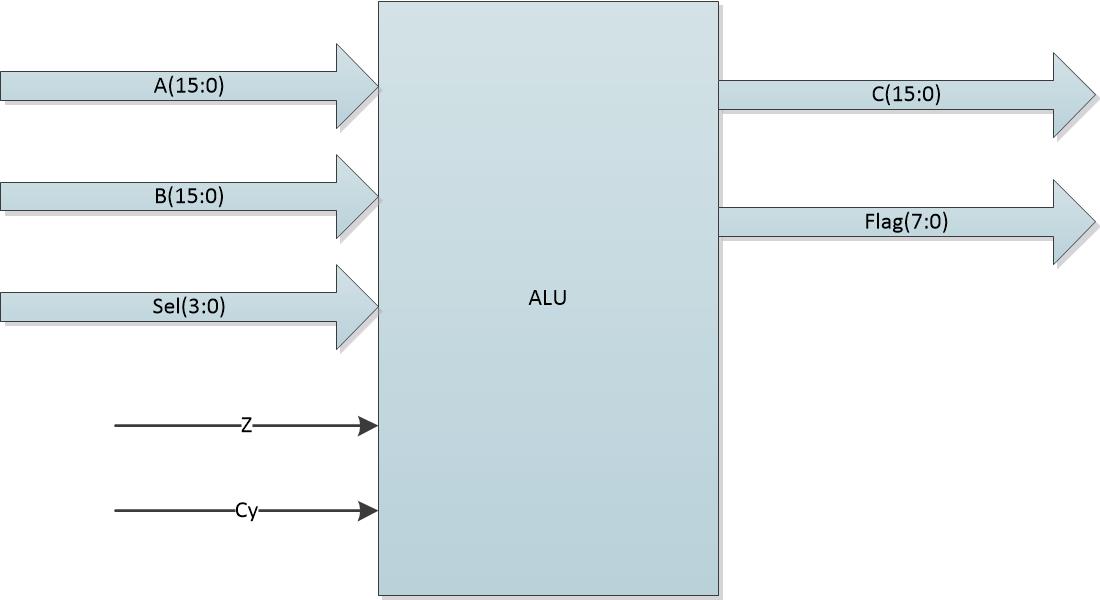
1. **各功能模块结构设计框图及功能描述**

Control模块：



组合逻辑电路根据current\_state的状态值确定各个输出信号的逻辑值，并且确定next\_state的状态值。而时序逻辑电路则在RST信号无效时，在时钟信号的上升沿将next\_state赋给current\_state，从而实现一个基于状态机模式的中央控制模块。

ALU模块：



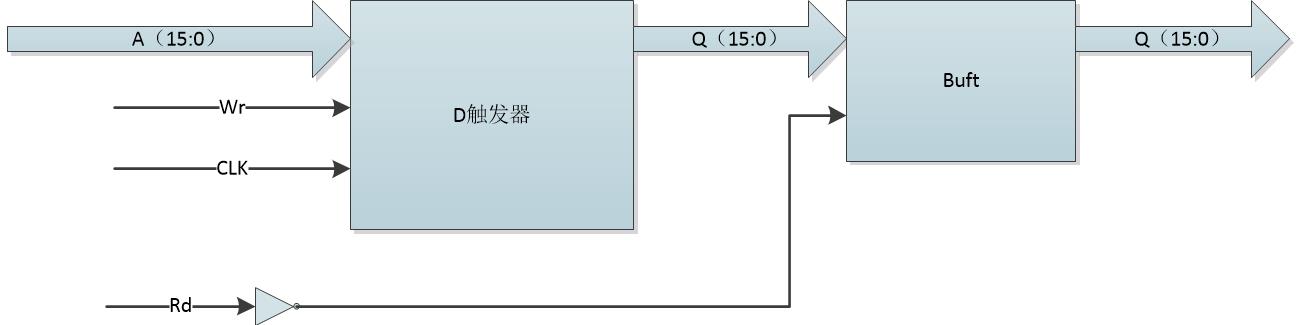
ALU根据Sel信号的内容完成运算操作，A和B是两个操作数，Z是零标志，Cy是进位标志，Flag是7位运算后的各个标志的结果，C是运算结果。

opreg、addrreg、instr模块（寄存器模块）：



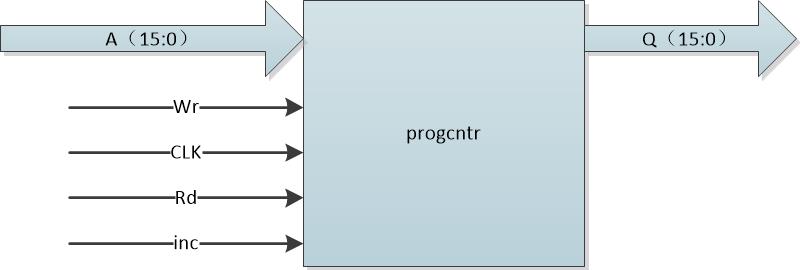
opreg、addrreg、instr这三个模块采用的是同一个元件，其实质就是一个D触发器。当Wr信号即使能信号有效时，在时钟下降沿将A的内容赋给Q。

outreg模块：



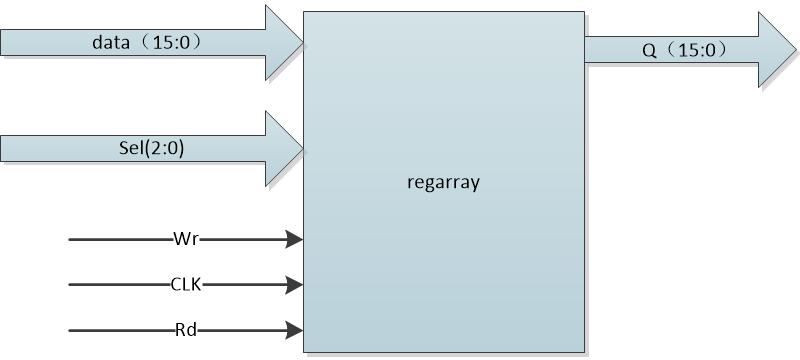
输出寄存器的功能和一般的寄存器差不多，主要由一个D触发器构成，只是多了一项功能为只有Rd信号有效的时候触发器的值才会被输出，而Rd无效时Q保持高阻态。

progcntr模块：



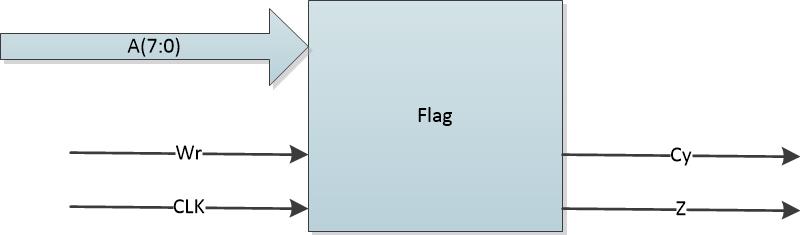
progcntr模块即PC寄存器当Wr信号有效时，寄存器的值会在CLK的下降沿变成信号A的值，当Wr无效而inc信号有效时寄存器的值会在时钟的下降沿自动加1。当Rd信号有效时，Q会输出寄存器的值，Rd无效时Q保持高阻态。

regarray模块：



这CPU中通用寄存器组模块，3位的Sel信号指示使用哪一个第几号寄存器，当Wr信号有效时数据data将在CLK的下降沿写入当前使用的寄存器中。而当Rd信号有效时当前寄存器的值输出到Q，Rd信号无效时Q保持高阻态。

Flag模块：



当Wr信号有效时，A的值将在CLK的下降沿写入寄存器中，而Cy和Z信号输出的是标志寄存器里进位标志的值和零标志的值。

1. **指令测试序列**

|  |  |  |
| --- | --- | --- |
| 地址 | 储存器内容 | 指令 |
| 0000 | 0000 | MOV R0,00F6 |
| 0001 | 00F6 |  |
| 0002 | 0801 | MOV R1,[00A0] |
| 0003 | 00A0 |  |
| 0004 | 1000 | MOV [00B0],R0 |
| 0005 | 00B0 |  |
| 0006 | 180A | MOV R2,R1 |
| 0007 | 2013 | MOV R3,[R2] |
| 0008 | 281C | MOV R4,[R3+0006] |
| 0009 | 0006 |  |
| 000A | 1818 | MOV R0,R3 |
| 000B | 3004 | ADC R4,FA00 |
| 000C | FA00 |  |
| 000D | 3004 | ADC R4,AF54 |
| 000E | AF54 |  |
| 000F | 3804 | ADC R4,R0 |
| 0010 | 4004 | SBB R4,0005 |
| 0011 | 0005 |  |
| 0012 | 4000 | SBB R0, 00FF |
| 0013 | 00FF |  |
| 0014 | 4819 | SBB R1,R3 |
| 0015 | 5001 | AND R1,00F0 |
| 0016 | 00F0 |  |
| 0017 | 5819 | AND R1,R3 |
| 0018 | 6003 | OR R3,0F06 |
| 0019 | 0F06 |  |
| 001A | 6819 | OR R1,R3 |
| 001B | 7000 | CLC |
| 001C | 3001 | ADC R1,0002 |
| 001E | 0002 |  |
| 001F | 7800 | STC |
| 0020 | 3001 | ADC R1,0001 |
| 0021 | 0001 |  |
| 0022 | 8000 | JMP 0030 |
| 0023 | 0030 |  |
| 0030 | 8811 | JZ 0011 |
| 0031 | 5000 | AND R0,0000 |
| 0032 | 0000 |  |
| 0033 | 881F | JZ IF |
| 0053 | 889A | JZ 9A |
| 003A | 7000 | CLC |
| 003B | 9011 | JC 11 |
| 003C | 7800 | STC |
| 003E | 901F | JC 1F |
| 005D | 9083 | JC 83 |
| 005B | 0000 |  |
| 00A0 | 0070 |  |
| 0070 | 0040 |  |
| 0046 | 56AC |  |

1. **实验总结**

通过自己的不懈努力最终完成了本实验，成功的做出了一个简单CPU，虽然过程有些艰辛，但非常有成就感。在做本实验的过程中，深入的学习VHDL语言，了解了写硬件语言的思想，真正学到了很多以前不会的东西，同时也更加理解CPU的内部结构和工作原理。在写本实验的VHDL代码中遇到了很多问题，大部分问题折磨了我很久，但最终都通过咨询老师和查阅资料将问题解决了。设计部分遇到的第一个大问题是如何实现JZ和JC，最初的想法是Control模块通过内部数据总线获取到PC的值，在Control模块内完成加减然后写回PC里，但是这样做不是特别合理因为Control模块作为中央控制逻辑是不应该接入数据总线，那应该只是单一的发出控制信号而不传输数据。所以经过思考后决定用ALU来实现PC值的加减，因此将这部分代码写进了ALU模块里，最终通过调试实现了8位有符号数的加减运算，成功地解决了第一个困难。之后遇到了一个整个实验中最大的困难，当我将所有代码都写完后也通过了综合，以为大功告成，但是波形里有太多的红线（数据冲突）。我几乎花费了整整一天在实验室里调试自己的代码，但仍旧没有解决问题。我的CPU模块间采用内部数据总线传输数据，最初的设计是输出数据模块的读信号和写入模块的写信号（写入是边沿触发）同时有效，但由于电路的延时，数据基本上写入的是数据总线上旧的内容。后来又改成写入为电平触发，但是有几个数据传输中数据总线上的内容会在写入信号无效前一点发生变化从而又使得写入的内容不对。之后想过让数据在数据总线上保持两个周期，在第二个周期写入数据，但这样太浪费时钟周期了，所以否决了这种想法。经过两三天的思考，发现我需要解决的问题无非就是让数据在总线上保持一段时间后再写入，那么完全可以在时钟上升沿放数据并保持一个周期而在时钟下降沿写入数据，半个周期用来让数据稳定是绰绰有余的。我按照这种想法修改了源代码，波形没有了红线而且输出结果也是正确的。之后下载到实验台上也没有出现任何问题，CPU设计成功，完成了本实验的内容。