实验5 简单计算机系统-系统设计C-实验报告

电 25 吴晨聪 2022010311

# 5.1 cpuF模块设计

要求增加三条跳转指令：BEQ，BNE，JMP，而实现该目的需要设计一个程序计数器PC并更改控制器controller。

程序计数器PC的功能主要是在有jump信号时将pcout直接改变为imm，在有branch信号时将pcout直接改变为pcout+1+imm，如果没有这两个跳转信号则取pcout=pcout+1。此外为了方便表示程序停止，在pcout增加到255以后便不再增加，即ROM中的代码只会跑一遍。

由于控制器新增加了输出jump和branch，新增加了输入zero，所以需要另外创建一个控制器模块controller\_2.v。控制器内部的逻辑较为简单，BEQ时判断i是否等于j来输出branch；BNE时则判断i是否不等于j来输出branch；JMP则直接输出jump。此外要注意op=15是没有设置指令的，但为了程序的鲁棒性有必要在op=15时做一些设置，以免regfile或者RAM中的内容遭到修改。

rom存入的机器码与汇编语言格式指令之间的关系如下表所示。

|  |  |  |
| --- | --- | --- |
| 行号 | 指令代码 | 机器码 |
| 0 | ANDI R0,R0,0 | 0x8000 |
| 1 | ANDI R1,R1,0 | 0x8500 |
| 2 | ADDI R0,R0,11 | 0xa00b |
| 3 | ADDI R1,R1,22 | 0xa516 |
| 4 | AND R2,R0,R1 | 0x0180 |
| 5 | OR R3,R0,R1 | 0x11c0 |
| 6 | ADD R2,R2,R3 | 0x2b80 |
| 7 | SUB R3,R3,R2 | 0x3ec0 |
| 8 | ADD R3,R3,R2 | 0x2ec0 |
| 9 | ADDC R2,R0,R1 | 0x6180 |
| 10 | SUB R1,R3,R2 | 0x3e40 |
| 11 | SUBC R2,R3,R2 | 0x5e80 |
| 12 | SLT R2,R1,R0 | 0x4480 |
| 13 | SLT R3,R0,R1 | 0x41c0 |
| 14 | SUB R2,R0,R1 | 0x3180 |
| 15 | SUBC R3,R0,R1 | 0x51c0 |
| 16 | ANDI R0,R0,0 | 0x8000 |
| 17 | SW R2, R0, 11 | 0xc20b |
| 18 | SW R3, R0, 12 | 0xc30c |
| 19 | LW R3, R0, 11 | 0xb30b |
| 20 | LW R3, R0, 11 | 0xb30b |
| 19 | LW R2, R0, 12 | 0xb20c |
| 20 | LW R2, R0, 12 | 0xb20c |
| 21 | ADD R2,R2,R0 | 0x2880 |
| 22 | ADD R3,R3,R0 | 0x2cc0 |
| 25 | BEQ R2,R3,2 | 0xdb02 |
| 26 | 0000 | 0x0000 |
| 27 | BNE R2,R3,3 | 0xeb03 |
| 28 | 0000 | 0x0000 |
| 29 | ADDI R0,R0,10 | 0xa00a |
| 30 | JMP 31 | 0x701c |
| 31 | 0000 | 0x0000 |
| 32 | ADDI R0,R0,5 | 0xa005 |

表格中，

25~26行，I型跳转指令，R2等于R3时跳转到27行；

27~28行，I型跳转指令，R2不等于R3时跳转到30行；

29行，I型addi指令，将寄存器R0置10；

30~31行，J型指令，跳过30行；

32行，I型addi指令，将寄存器R0置5。

最终观察R0为10还是5判断R2、R3是否相等。

由于ROM输入有寄存器，所以执行跳转指令时，跳转控制信号要在下个时钟周期才能到达pc，故在跳转指令后加0000指令。

一張含有 螢幕擷取畫面, 電子產品, 多媒體軟體 的圖片

AI 產生的內容可能不正確。

图1 cpuF仿真运行结果

从图中可知，R2=13，R3=14，二者不相等。

图中指令的执行顺序为db02(25)→0000(26)→eb03(27)→0000(28)→a005(32)。且结果为5。与上述相印证。说明跳转指令正确执行。

# 5.2 N为正整数计算 1 ~ N 中所有质数之和

**高级语言实现:**

先利用Matlab高级语言实现功能，代码如图2所示：

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

AI 產生的內容可能不正確。

图2 Matlab实现质数求和代码

整体判断逻辑为，第一层循环(for i)从2遍历到N，依次判断各个数是否为质数；第二层循环(for j)从2遍历到i-1，判断i是否能被j整除，若i能被整除则i不是质数，若不能被整除则判断能否被下一个j整除，只有不能被所有的j整除才是质数；第三层循环(while)用循环减的办法来实现整除，k的初值取为i，每次循环减去j直到k<=0为止，若k<0则说明i不会被j整除；若k=0说明i会被j整除。每个第一层循环(for i)的结尾都会把sum的结果加上i（若i为质数）。

以此为基础着手构建基于设计的CPU的汇编语言代码。

**高级语言转化为汇编语言:**

将高级语言转化为汇编语言主要有以下几个问题：1、循环的实现；2、大数字的存储、计算；3、regfile空间的运用。以下将依次讲述如何解决以上三个问题。

首先，循环的实现需要通过BEQ、BNE和JMP指令来实现。由于BEQ和BNE只能向后跳转（因为存储的imm是非负数），而JMP可以跳转到任何位置，所以BEQ和BNE主要用来if语句的判断，而JMP用来跳转回循坏的开始部分。

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

AI 產生的內容可能不正確。

图3 汇编语言代码片段1

如图3所示，第一层循环的开始部分的代码（a1为循环开始处），这里要通过判断i和N+2是否相同来判断是否停止第一层循环（N+2事先存入了RAM的7、8位）。首先通过BNE来判断i和N+2的低八位是否相同，如果不相同则跳过后续的判断继续该循环，如果相同则继续用BEQ判断i和N+2的高八位是否相同，如果相同则直接跳出该循环（x1为第一层循环的结尾）。

一張含有 文字, 字型, 螢幕擷取畫面, 數字 的圖片

AI 產生的內容可能不正確。

图4 汇编语言代码片段2

如图4所示，在循环的结尾直接用JMP跳转回循环的开始，把跳出循环的判断部分都交给开头的BNE和BEQ来判断。如此即实现了循环。

关于大位数字的存储和计算部分，i、j、k、N+2都是一千多的数字，可以用2个RAM的空间进行存储，sum的实际结果超过了2^16，需要3个RAM的空间来处理。分配RAM中的地址编号，i:0,1 j:2,3 k:4,5 N+2:7,8 sum:49,50,51。每次需要时从RAM中取用，用完再存回即可。在涉及两个数的计算时如果是两个占2个RAM空间的数可以直接放入regfile计算，而如果是i+sum这样的共占5位的需要分部相加。

一張含有 文字, 螢幕擷取畫面, 字型, 筆跡 的圖片

AI 產生的內容可能不正確。

图5 汇编语言代码片段3

如图5所示是将i+sum实现的代码部分，实现的方法为先把i的高八位和低八位存放到regfile的R0、R1中，再将sum的最低位放入R2中，R3中存放的是0。先将R1和R2，相加结果存放在R2中。此时可能会产生进位，将R3和R0相加，结果存放在R0中，由于R3存放的是0，所以相当于把进位直接加到了i的高八位中，此时注意到i的高八位最大为4，再加上进位的1也不可能再产生进位，所以可以不再考虑进位的处理。此后将R2存回到RAM中，并将sum的中位和高位取到R2和R1中，把R2和R0相加，结果存放到R2中，再将R1和R0相加结果存放到R1中，此时连续的加法可以直接考虑到进位无需额外存储。最后将R2和R1存回到RAM中，即完成了sum=sum+i的操作。

最后是regfile空间如何利用。regfile只有4位空间，所以每次计算完都会把需要存储的数放回到RAM划定好的空间里，有需要时再取用。此外regfile中的R3一直被赋值为0，因为LW和SW指令都需要立即数和一个寄存器的相加作为地址，R3可以作为地址的基值也可以作为运算时的0值。在j-k的计算中为了方便曾经将j和k全都放进regfile中进行计算，这样就占用了所有的四个空间，但是运算结果只占用两个空间，所以在运算完毕后再将R3置为0即可。

一張含有 螢幕擷取畫面, 文字 的圖片

AI 產生的內容可能不正確。

图6 汇编代码运行结果

利用Matlab进行编程求解，其结果是94056。将写好的代码放入ROM中，其运行结果是1，128，230，将其相加有1\*256\*256+111\*256+104=94056，运算结果正确。

rom存入的机器码与汇编语言格式指令之间的关系见Ex5.xlsx及rom3.hex。

**遇到的问题及解决方法:**

①时序问题

由于ROM和RAM的输入有寄存器，在执行LW指令和跳转指令时，输出会在下一个周期到达regfile和pc，所以在单时钟周期时，需要两个周期确保指令正常执行。

②设计复杂

由于指令有限，在执行循环时需要反复判断再跳转以实现功能，只能是慢慢理清思路，做好跳转标记。而且每改一次代码，就需要更改跳转的具体位置，十分繁琐。

③debug困难

仿真结果无法直接得出逻辑错误，大部分时候都会陷入死循环，甚至不知道何时停止。只能缩小循环范围，再一个周期一个周期地去对比汇编语言和仿真结果，如果逻辑本身不清晰，任务量巨大。