计算机组成原理课程设计答辩记录表

|  |  |  |  |
| --- | --- | --- | --- |
| 学 院 | 专业 / 学号 | 学 生 | 指导教师 |
| 计算机科学与技术学院（人工智能学院） | 23计算机科学与技术/ |  | 付峰 杨东鹤 |
| 答辩时间 (年月日/时段) | 2025年6月28日  13:49 | 答 辩 地 点 | 10-413 |
| 答辩记录(可另附页)：  问题1：传输指令的微程序流程如何设计？  回答1：在本次微程序设计中，我先后实现了减法指令、传送指令与条件转移指令的微程序，通过亲自设计并调试这些指令的执行过程，我逐步掌握了 TEC-2 控制系统中微指令各字段的含义与配置技巧。传送指令看似只是简单的数据赋值操作，但其背后仍依赖寄存器读出、ALU 数据通路和写回机制，这让我认识到即使是最基础的操作也需要严谨的微控制设计。值得注意的是，TEC-2 机允许用户自定义指令操作码，且新旧指令可以在同一个程序中并存运行。然而，每次教学机断电或重启后，新指令所依赖的微程序必须重新通过 LDMC 指令加载进控存，才能正常执行。由于新指令是在汇编语言体系之外设计的，它们一般只能通过机器码直接调用，除非对汇编器进行扩展支持。  幸运的是，加载与调用新指令的过程并不复杂，这使得我可以将更多精力集中在微指令逻辑本身的设计上。需要特别理解的是，尽管每条新指令的功能可以独立设计，但它们在取指（通常对应控存地址 19H）和后续中断判断（对应控存地址 A4H）等阶段，仍然需要借助原有微程序段的支持。也正因如此，新旧指令的衔接方式尤为关键。在我的设计中，我确保至少有一条新指令调用原指令的取指过程，另一条新指令结束于 A4H，以调用系统已有的中断检测机制，从而保障整个微程序序列在指令间切换时的正确性和一致性。通过这样的实践，我不仅掌握了如何编码一个完整的微程序段，更加深了对微程序控制原理及其系统衔接逻辑的理解。  问题2：讲解一下TEC-2有关汇编方面的知识？  回答2：比如写汇编指令首先输入A0800,A0820，A 命令表示地址设定，用于进入主存地址输入模式，是你写机器指令或汇编指令时的第一步，0800和0820是段首地址，没必要一直写800和820作为段首地址，段首地址可以随意设置，比如500,600，前提是这个地址空间没有被占用；地址空间不能直接用立即数赋值，必须用寄存器间接赋值；E是写指令，G是运行段的首地址知道结束；NOP用于预存空间，为微指令服务的，直接在汇编中写微指令汇报错，一个NOP对应单地址指令，两个NOP就是双地址指令，三个NOP就是多地址指令；LDMC是用于加载微指令的；我们写汇编指令比如会写D510,D5就是8-15位的操作码，1是DR，0是SR，这里的数字就是寄存器R的下标；D510后面的0005是ADDR也可以是偏移地址DISP；E0828的E后面是写段地址，不一定是段首地址，段中间的地址也是可以的；56位微指令是要写成4个4位16进制，左边高位，右边低位，看别人写的完整微指令时需要把16进制拆成4位2进制去看，对着课程设计指导书后面微指令有关的表去推即可，至于56位微指令为啥是4个4位16进制，47-55位需要补0变成4位16进制，0-47则是对应表格判断变成3个4位16进制；R=D时，即需要用到MEM时需要设置MIO REQ WE为001,MEM就会送到ALU的输入端D。  问题3：讲述一下在设计第二条传送指令ADDR2 -> DR+ADDR1的思路？  回答3：一共有两种思路，第一种就是将ADDR2的地址存进AR，再把ADDR2的值放进Q里面，再把DR+ADDR1地址对应的值赋值给Q，但是这个思路有一个bug，我是DR+ADDR1也就是DR+MEM -> AR发现行不通，因为DR是B口，MEM是D，功能表上只有D+Q,D+A,D+0,没有D+B，因此需要多做一步，即把DR放进Q里面，再Q+ADDR1，可是Q已经存储的是ADDR2地址对应的值，故这题必须需要两个寄存器，第二种思路也是在基于第一题的必须用两个寄存器基础上做的，只是先后顺序不同罢了，先是把DR+ADDR1对应的地址存储在另一个非Q的寄存器，这里我用的是通用寄存器R0，然后把ADDR2地址对应的值放在Q里面，把R0的值给AR，取出地址，Q -> MEM,放进地址里面，整体上看上去没有问题，思路也十分正确，杨老师提出了一个非常好的瑕疵，就是这里必须用两个寄存器嘛，一个是Q，ALU里面的，这个没有问题，用户不能直接修改和用Q里面的值，但是R0这是通用寄存器，用户可以随便使用，比如这题如果DR用的是R0，这题不就有bug了吗，然后我就说可以让用户不使用寄存器R0，使用R1,R2,R3R7,R8不就可以了吗，就是限制用户不使用R0，但杨老师提出如果是在实际生活中设计减法微指令，用户肯定不知道不能用R0，这必然会出现bug，不够严谨，后来在杨老师的指导下可以用IP寄存器，虽然IP是用于存放当前正在执行的机器指令的指令代码，但正在运行时可以运行IP的值被覆盖，因为此时执行Q+MEM -> IP这一指令时IP的功能已经实现，就是找到正在执行的机器指令的指令代码，可以在执行期间覆盖，运行完后IP又会变成下一个指令的地址，且用户不能修改IP的值，完美起到了保护的作用。经过杨老师的提示和指点，我知道了IP寄存器的妙处，如果受保护的寄存器用完情况且还需要使用一个寄存器的情况下，就可以考虑使用IP寄存器。 | | | |