**桂林航天工业学院学生实验报告**

**实验五**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 计算机组成与结构 | | | 实验名称 | | 计算机组成与机器指令周期（4学时） | |
| 开课教学单位及实验室 | | | | 计算机科学与工程学院 | | 实验日期 | 2024.11.26 |
| 学生姓名 | | 廉振威 | 学号 | | 2023070030615 | 专业班级 | 23软件工程6班 |
| 指导教师 | | | | 张亚红 | | 实验成绩 |  |
| 实验目的 | | | | **1) 通过总线将微程序控制器同运算器、存储器等连接起来，组成一台计算机**  **2) 使用微程序控制器控制模型机数据通路，运行由4条机器指令组成的简单程序**  **3) 理解微指令与机器指令的关系，理解计算机的指令周期** | | | |
| 实验要求 | | | | 1. **做好预习，复习微指令的格式和微指令控制器中各部件发挥的作用，了解实验中用到的器件和使用方法** 2. **按步骤完成实验，独立分析，按要求作好记录** 3. **完成实验报告** | | | |
| 一、实验电路    图1 整机原理图    **二、实验电路**  本实验将前面几个实验中的所有电路，包括运算器、存储器、通用寄存器堆、微程序控制器等模块组合在一起，构成一台简单的模型机。因此，在基本实验中，这是最复杂的一个实验。  在前面的实验中，实验者本身做为“控制器“，完成了对数据通路的控制。而在本次实验中，数据通路的控制器将交由微程序控制器来完成。TEC-5G从内存中取出一条机器指令到执行指令结束的一个指令周期，是由微程序来完成的，即一条机器指令对应一段微程序。   |  |  |  | | --- | --- | --- | | 74LS163 | 4位二进制计数器 |  | | 74LS374 | 8位正沿触发器存器 |  | | NOTgate | 非门 |  |   图2 实验电路图  **三、实验设备**   1. TEC-5G计算机组成实验系统1台 2. 逻辑测试笔一支（在实验台上） 3. 双踪示波器一台（公用） 4. 万用表一只（公用）   **四、实验任务**  整体结构延续了数据通路的架构，不同之处主要有以下几点：   1. 以微程序控制器替代原有的手动开关，控制信号由ROM发出，结合时序产生器，控制数据通路 2. 增加了如下几个计算机组成中不可或缺的部件： 3. 程序计数器PC用于指令寻址，同时，也作为本实验的数据寻址，用2片74163（4位计数器）串接起来； 4. 通用寄存器R0，在本实验中用作隐含操作数，由一片74374构成，该器件本身具有输出使能端，不需额外配备三态门742 5. 指令寄存器IR，在微程序控制器实验中，使用了开关表示IR，本实验使用一片8位锁存器74273构成 6. 以上部件均与总线相连   实验中用到四条机器指令，IN（输入），ADD（加法），STO（存数），JMP（转移），操作码分别为000， 001, 010, 011，需要注意的是，IN命令不需要操作数，长度为8位，其他3条指令均有一个操作数，因此本实验采用了变长指令格式，具体指令格式如下表所示：  表 1 指令格式   |  |  |  |  | | --- | --- | --- | --- | | 指令 | 机器码 | 长度 | 功能 | | IN | 00000000 | 8位 | SW->R0 | | ADD D | 00100000 D | 16位 | R0+(D)-> R0 | | STO D | 01000000 D | 16位 | R0->(D) | | JMP D | 01100000 D | 16位 | D->PC |   待执行的指令事先存储在RAM中，若干条指令构成一段计算机程序，RAM的内容如下：  表 2 RAM中的程序和数据   |  |  |  | | --- | --- | --- | | 地址（八进制） | 内容 | 含义 | | 00 | 00000000 | IN | | 01 | 00100000 | ADD | | 02 | 00001000 | D=(10)8 | | 03 | 01000000 | STO | | 04 | 00001001 | D=(11)8 | | 05 | 01100000 | JMP | | 06 | 00000000 | D=(00)8 | | 07 | 00000000 | 暂未使用 | | 10 | 00001011 | (D)=(13)8 | | 11 | 00000000 | (D)=(00)8 |   微指令流程图如下所示，每个流程对应的微指令地址在方框右上方：    图 3 微程序流程图  其中，3条指令的数据寻址在执行周期完成，完成执行周期后都返回到取指周期，除JMP命令外，下一条指令的地址均由上一个指令周期取址阶段经过PC+1给出，即顺序执行，而JMP命令修改了PC的值，因而能实现程序的跳转。  对应的微指令代码存放在控制存储器中，如下表所示，此处，微地址采用8进制表示：  表 3 微程序二进制代码表   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | 地址 | S3 | S2 | S1 | S0 | M | -Cn | -CE | WE | -LOAD | LDR0 | LDDR1 | LDDR2 | LDIR | LDPC | LDAR | -ALU-B | -PC-B | -SW-B | -R0-B | P(1) | uA3 | uA2 | uA1 | uA0 | | 00 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | | 01 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | | 02 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | | 03 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | | 04 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | | 05 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | | 06 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | | 07 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | | 10 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | | 11 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | | 12 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | | 13 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | | 14 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | | 15 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |   **五、实验步骤与结果**  （此处填写实验步骤描述、验证的规律与运行结果截图）  整体过程：  单步执行微指令，观察程序、指令执行与微程序、微指令执行的关系  分解步骤   * 1. 导入实验电路   2. 打开电源开关，注意，导入实验电路后，电路预置之前必须打开电源开关，再进行电路预置，否则寄存器的输出值为高阻态，计算机无法启动。   3. 电路预置。DR1、 DR2和AR的MR置1，计数器的CR、ENT、ENP置1，时序发生器Step置1。微地址寄存器74175和指令寄存器IR的MR置1。此时微地址寄存器和IR已经初始化为零，计算机将从控制存储器的零地址开始运行。   *此处请贴预置后的电路图*  屏幕截图 2024-11-19 160056   * 1. 数据开关（SW7～SW0）上设置好程序起始地址（00000000）。   2. 单击1次时序发生器Start按钮，思考并回答问题：此时执行的是微程序流程图中的第几条微指令？作用是什么？   答：第一条微指令是NOP，通常用于初始化或占位。接下来是SW → PC'，这表示将开关（SW）的值加载到程序计数器的下一个值（PC'）。这是微程序流程图中的第二条微指令，作用是设置程序计数器，以便从用户指定的起始地址（00000000）开始执行程序。   * 1. 再单步执行2条微指令，思考并回答以下问题：这2条微指令的作用是什么？   答：• PC → AR：将程序计数器（PC）的值加载到地址寄存器（AR）。• PC+1：程序计数器自增1，指向下一个指令。这两条微指令的作用是准备读取内存中的下一条指令。首先，将当前PC的值送到AR，以便从内存中读取指令。然后，PC自增，为读取下一条指令做准备   * 1. 通过数据开关（SW7～SW0）设置操作数1的值为00010100。思考并回答问题：此设置是否可以提前？如果可以，最早应该在以上第几步之后？   答：操作数的设置通常在指令读取之后进行。在微程序流程中，这应该在“RAM → BUS”和“BUS → IR”之后，即在内存中的指令被读取到指令寄存器（IR）之后。根据流程图，这应该是在序号03之后。   * 1. 单击Start，执行微指令SW->R0，将操作数1保存到累加器R0中，此时总线上的值来自哪里？   答：当执行微指令“SW→R0”时，总线上的值来自数据开关（SW7SW0），因为这些开关被用来设置操作数1的值。   * 1. 继续单步执行之后的微指令，注意观察微指令的下地址，直到第一轮循环结束。在此过程中注意观察总线上数据灯的显示，并说明每个显示出来的数字的意义，将下表补充完整。   表 4总线数据表   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 总线上数据  （二进制） | 微指令编号  （八进制） | 意义（地址请用二进制表示） | | 1 | 00000001 | 02 | 当前PC值，即内存地址01 | | 2 | 00000010 | 02 | 递增1后PC的值 | | 3 | 00100000 | 03 | 内存地址01中的ADD指令操作码 | | 4 | 00000010 | 02 | 当前PC值，即内存地址02 | | 5 | 00000011 | 02 | 递增1后PC的值，即内存地址03 | | 6 | 00001000 | 03 | 内存地址02中的指令操作码，可能是NOP | | 7 | 00001011 | 03 | 内存地址03中的指令操作码，可能是JMP | | 8 | 00010100 | 07 | 内存地址04中的数据，此数据也是一个地址 | | 9 | 00011111 | 07 | 内存地址07中的数据 | | 10 | 00000011 | 02 | 当前PC的值，即内存地址11 | | 11 | 00000100 | 02 | 递增1后的PC值 | | 12 | 01000000 | 03 | 内存地址11中的指令操作码，可能是STA | | 13 | 00000100 | 12 | 当前PC的值，即内存地址100 | | 14 | 00000101 | 12 | 递增1后的PC值 | | 15 | 00001001 | 07 | 内存地址100中的数据，此数据也是一个地址 | | 16 | 00000000 | 07 | 内存地址1001中的数据 | | 17 | 00011111 | 03 | 内存地址07中的指令操作码，可能是JMP | | 18 | 00000101 | 02 | 当前PC的值，即内存地址101 | | 19 | 00000110 | 02 | 递增1后的PC值 | | 20 | 01100000 | 03 | 内存地址101中的指令操作码，可能是IN | | 21 | 00000110 | 13 | 当前PC的值，即内存地址110 | | 22 | 00000111 | 13 | 递增1后的PC值 | | 23 | 00000000 | 07 | 内存地址111中的数据 |   *此处任选两个序号，标注所选序号并贴出电路图*  *4*  屏幕截图 2024-11-19 172822  *5*  屏幕截图 2024-11-19 173314  六、思考题  1.如何实现程序的跳转？  **1. 直接跳转指令：** 跳转指令通过直接修改程序计数器（PC）的值来使程序流转移。程序计数器的值被设置为目标地址，程序将从这个新地址开始执行。直接跳转可以是无条件的，也可以是条件性的，根据是否满足某个特定条件来决定是否跳转。  **2. 条件跳转：** 条件跳转依赖于先前的运算或比较结果。只有在特定条件成立时，程序才会进行跳转。例如，某个标志位为真时，或者某个值满足特定比较时，才会执行跳转指令。  **3. 间接跳转：** 间接跳转与直接跳转不同，跳转的目标地址存储在寄存器或内存中，而不是在指令中硬编码。程序计数器会被设置为这些存储位置中的值，从而实现跳转。此方式常用于实现动态调用或虚拟函数。  **4. 子程序调用：** 当程序调用子程序时，当前程序计数器的值会被压入栈中，作为返回地址。接着程序跳转到子程序的起始地址。子程序执行完毕后，通过从栈中弹出返回地址，程序计数器被恢复，继续执行调用指令之后的代码。  **5. 异常与中断处理：** 异常和中断是程序运行中可能遇到的特殊情况。当中断或异常发生时，程序的正常执行流程会被打断，控制权转移到中断或异常处理程序。处理完毕后，通过特定的返回指令恢复到中断或异常前的执行状态。  **6. 循环与分支：** 在循环结构中，程序会重复执行某段代码，直到满足退出条件。通常，循环通过条件跳转指令来实现，即在每次迭代结束时检查条件，如果不满足，则跳转回循环开始位置继续执行。  **7. 微程序控制：** 在微程序控制的计算机体系中，跳转通过调整微指令序列中的下一个地址来实现。这种跳转是由微指令内的条件逻辑控制的，可以灵活地根据程序的当前状态决定下一步执行的操作。  **8. 硬件加速跳转：** 一些高级处理器提供硬件支持来优化跳转指令的执行，例如分支预测。通过预测跳转指令是否会被执行，处理器可以提前加载可能需要的指令，从而减少由于跳转带来的性能损失。 | | | | | | | |