# 实验报告模板

华南理工大学

《计算机组成原理》课程实验报告

实验题目： 控制器

姓名： 陈映松 学号： 202130442563

班级： 21计科(2)班 组别：

合作者：

指导教师： 张齐

|  |
| --- |
| **实验概述** |
| 【实验目的及要求】  实验目的：   1. 掌握数据通路、微命令、微指令和微程序的概念 2. 理解微程序控制器的工作原理及其如何依次构建CPU 3. 理解硬布线控制器的工作原理及其与微程序控制器的差异   实验要求：   1. 学习微程序控制器和硬布线控制器的使用、 2. 学习微程序控制器中存储与IO访问的内容 3. 学习微程序控制器中运算器的内容 4. 学习微程序控制器中中断的内容   【实验环境】  Proteus 8.1  Digiblock  Java配置环境 |
| **实验内容** |
| 【实验过程】   1. 实验步骤：（“实验步骤”）   **实验1：微程序控制器和硬布线实验**  1. 如下图 4.1.1 所示的单总线架构 CPU 中，红色部分即为 CPU 的控制器。在 CPU 的任何一个时钟周期，控制器发出若干操作控制信号（称为控制流），操作若干指定的部件，使得信息（称为数据流）从某一个部件经过总线到另一个或若干个部件，则该信息的路径称为一个数据通路。例如一条指令从存储器 MEM 经过暂存器 DR 传送到指令寄存器 IR，形成一个取出指令的数据通路；该指令（假设是运算指令）被控制器解析后，开始执行：首先控制器控制通用寄存器堆 Regs 发送数据到运算器 ALU 及其数据暂存器 X，形成该指令执行过程的第一个数据通路；然后，控制器控制运算器 ALU 的结果暂存器 Z，将运算结果发送到通用寄存器堆 Regs 的某一个寄存器，形成该指令执行过程的第二个数据通路。注意：因为总线是唯一的，所用 CPU 的任何一个时钟周期，有且仅有一条数据通路存在。  2. 为简化起见，我们先构筑一个最小版本指令集，方便讨论数据通路与控制器。假设目前只有三条指令：NOP（空闲）、HLT（停机）与 JMP（跳转），NOP 和 HLT 指令是单字节指令，字节高 4 位是 OP 码；NOP 与 HLT 的 OP 码相同，靠字节最低位为 0/1 区分是 NOP/HLT指令。JMP 指令是双字节指令，第一字节高 4 位是 OP 码，低 4 位没有定义；第二字节是跳转地址。因为是指令内带有地址信息，所以是直接寻址。具体指令格式如下表所述：对应上述最小版本的指令集，我们可以构造一个极简的数据通路，如下图 4.1.2 左所示。用户通过手动操作三个控制信号（EN，PC\_INC 和 LDPC），在“取指->执行->取指->…”的不断周期循环中执行上述三条指令，每条指令不同周期的控制信号序列如下图 4.1.2 右所示。  3. 如下图 4.1.3 所示，微程序控制器的原理是把每条 CPU 指令看作一段微程序，组成程序的若干条微指令与指令的若干个指令周期相对应，所有的微指令存储在微指令存储器(控制存储器)中。因为上图 4.1.2 中，最小版本指令集的三条指令仅有两个不同的指令周期：公共的取指周期和 JMP 指令的执行周期。所以在微指令存储器地址[0000]中是公共的取指微指令，而地址[0001]中则是 JMP 指令的执行微指令。一条微指令由微命令字段，P 字段和下址字段组成。微命令字段的每一位都是控制 CPU  各个部件的微操作控制信号；下址字段指示当前微指令的下一条微指令地址；P 字段用于微程序的分支跳转，例如取指周期后，不同指令的执行周期分支跳转则通过 P 字段的 P1 信号和 OP 码决定，即下址=P1\*(I7,I6,I5,I4)。在一个微指令周期中，微指令的微命令字段所有位（微命令）并行发送，在 CPU 内形成数据通路，把数据从源部件经过总线发送到目的部件。  4. 在上图 4.1.2 的数据通路上，增加微程序控制器就构成一个“最小 CPU”（第一阶段），如下图 4.1.5 左所示。微指令存储器（控制存储器）MROM 由微指令地址寄存器 MAR 驱动，输出微指令的微命令字段直接形成微操作信号（PC\_INC 和 LDPC），而下址字段[0-3]则与指令的 OP 码一起输入选择器，由 P1 决定：当 P1=1 时刻（公共取指微指令,当前微指令地址[0000]），下一条微指令地址不是下址字段，而是 OP 码所表示的地址（JMP 指令 OP 码 0001，所以下一条微指令的地址是[0001]）；当 P1=0 时刻（JMP 指令的执行微指令，当前微指令地址[0001]），下一条微指令地址是下址字段（地址[0000]，返回下一条指令取指）。同样，第一阶段硬布线控制器版本“最小 CPU”如下图 4.1.5 右所示，由上图 4.1.2 的数据通路和硬布线控制器构成。在硬布线控制器中，状态 Mx 循环转移由 M 状态机控制，受指令 OP 码影响：当且仅当 NOP 和 HLT 指令时，状态转移是{M1->M1}，其余情况均是{M1->M2->M1->…}。微操作信号 PC\_INC 和 LDPC，以及停机信号 hlt 均由指令及其状态 M共同形成，反映了该信号出现在特定指令及其下特定状态的所有可能性。  此外，停机信号 hlt 由 HLT 指令([7-4]位和[0]位)硬件生成。信号 hlt=1 时刻，下一个clk 上升沿则信号 halt=1，阻塞 clk（恒为 1）；若 reset=1 则时钟高电平时刻 halt=0 清零,恢复 clk 信号。Counter 计数器受微程序控制器的 P1 信号或硬布线控制器的 M1 信号驱动,用来指示当前指令是程序中第几条指令（NOP 和 HLT 指令除外，不在统计中）  5. 根据最小版本指令集可以编写简单的二进制机器语言程序，如下表所示：程序  jump.hex 总共包含了四条指令，实现了循环“跳转”的功能。图中左列的汇编助记符是中间列注释的英文缩写（后来演化成为汇编语言），对应右列的指令二进制机器代码。请将上述 jump.hex 二进制代码分别加载到上图 4.1.5 左右所示的微程序控制器和硬布线控制器版本的“最小 CPU”电路中。然后，硬布线控制器版本 CPU 可以直接运行，而微程序控制器版本 CPU 还需在微指令存储器 MROM 加载上图 4.1.3 所示的微程序二进制代码。完成加载后，启动运行微程序控制器版本 CPU，PC=0，ROM 输出第一条指令 JMP 第一字节；OP=0001；此时，微程序控制器内的 MAR=0000,表示当前是取指微指令。随后单步执行，用表格记录每一次 clk 后，ROM 存储器的输入（地址 PC）和输出（指令），以及 MROM 存储器的输入（MAR）和输出（微指令）。同理，完成加载后，启动运行硬布线控制器版本 CPU，ROM 输出第一条指令 JMP 第一字节；OP=0001；此时，M1=1,表示当前是取指周期。随后单步执行，用表格记录每一次 clk后，ROM 存储器的输入（地址 PC）和输出（指令），以及状态机 Mx 的变化。  **实验2：微程序控制器—存储与IO访问实验**  1. 如下图 4.2.1 所示，在上述最小版本指令集（NOP/HLT/JUMP）基础上，扩充以下三类指令：存储器赋值指令：MOV 寄存器间赋值/SET 寄存器立即数赋值；IO 读写指令：IN101读端口/OUT 写端口；存储器操作指令：LAD 读存储器/STO 写存储器，POP 堆栈出栈/PUSH 堆栈入栈（注：此处堆栈是以存储器为物理载体，通用寄存器为“指针”的软堆栈）  2. 与上述指令集扩展对应的微程序控制器 CPU（第二阶段）如下图 4.2.2 所示：原有的PC 跳转通路和微程序控制器基本不变，而输出的微操作信号更多了，因为需要操作的电路增加了寄存器电路、数据通路选择电路和存储器读写电路。此外，指令存储器 ROM 的访问范围是 00H-FFH（PC 计数范围），可读写数据存储器 Dram 的访问范围是 00H-7FH，只读数据存储器 Dram 的访问范围是 80H-FFH。  3. 上述 CPU 的指令周期流程图，微指令格式和微指令存储器中的所有微指令编码如下  图 4.2.3 所示：指令周期流程图保持不变，仍是取指和执行两个周期。因为指令集扩充至 11条指令，所以微指令扩充到 8 条：取指微指令地址[0000]就是 NOP/HLT 指令 OP 码，其余7条微指令地址对应JMP指令和上述新增指令的OP码。注意：LAD与POP指令共用地址[0110]微指令，STO 与 PUSH 指令共用地址[0111]微指令，由 P2 信号和指令第一字节的[0,1]位（即Rb 字段）区分指令 LAD 与 POP，以及指令 POP 与 PUSH。  4. 如上图 4.2.2 所示，因为微指令结构中，微命令字段增加了 Asel 和 Ssel 字段，其作用如下图所示。在不同指令的执行过程中，需要建立一条数据通路，选择某一个源部件输出数据，写入到另一个指定的目标部件。因此，Ssel 字段从以下端口/数据中选择源部件：通用寄存器输出 Da/Db，存储器输出 Mout，输入端口 IN，指令第二字节立即数。Asel 字段则从以下端口或数据选择目标部件（及其有效写入信号）：通用寄存器输入端口 Rin(写入使能信号 Regw)，存储器输入端口 Min(写入使能信号 memw)，输出端口 OUT(使能信号 Enout)。如上图 4.2.2 所示，由于当前指令的 OP 码与其使用的通用寄存器地址 Ra/Rb 是在指令103第一字节，然而该字节出现在指令的取指微指令周期，要等到下一个周期才能根据 OP 码取出相应的执行微指令操作通用寄存器。所以，在下图中，Ra/Rb 字段必须锁存后延迟一个周期出现，即在该指令的执行微指令周期有效。与此同时，在存储器操作指令中，地址[0110]/[0111]的执行微指令都增加了 P2 字段，用来根据指令第一字节的[0,1]位（即 Rb 字段）判断是双字节指令 LAD/STO 还是单字节指令 POP/PUSH：若 Rb=00,表示当前是双字节指令LAD/STO 的执行微指令周期，ROM 当前输出指令第二字节 IMM，需要 PC+1，使得 ROM在下一个周期取出下一条指令的 OP 码；若 Rb≠00，则表示当前是单字节指令 POP/PUSH的执行微指令周期，ROM 当前输出下一条指令的第一字节，所以不需要 PC+1(如下图所示)。  5. 采用上述扩充的指令集，编写了一段赋值程序 mem.hex，实现了通用寄存器赋值，存储器的立即寻址访问和寄存器寻址访问（即堆栈指令），如下图所示。请将上述 mem.hex 二进制代码加载到上图 4.2.2 所示的第二阶段微程序控制器版本 CPU电路中。然后还需在微指令存储器 MROM 加载上图 4.2.3 所示的微程序二进制代码。完成加载后，启动运行第二阶段微程序控制器版本 CPU，PC=0，ROM 输出第一条指令 SET第一字节，OP=0011；微程序控制器内的 MAR=0000,表示当前是取指微指令，P 字段的 P1=1，P2=0，指示下一条微指令地址由 OP 码[0011]决定。随后单步执行，用表格记录每一次clk104后，ROM 存储器的输入（地址 PC）和输出（指令），MROM 存储器的输入（MAR）和输出（微指令），以及存储器 Dram、寄存器堆 regs 和输入/输出端口 IN/OUT 的变化.  **实验3：微程序控制器—运算器实验**  1. 如下图 4.3.1 中所示，在上述扩充指令集（寄存器、IO 和存储器访问）基础上，继续扩充以下三类指令：算术逻辑运算指令：包括 ADD/SUB/MUL 算术运算指令和 AND/OR/XOR 逻辑运算指令，其中每一类运算指令均有双寄存器运算和“寄存器+立即数”运算（-I）版本；移位指令：逻辑/循环左（右）移位指令（共享 OP 码，由指令[0,1]位区分）；JUMP 指令扩充成 4 个类型（共享 OP 码，由指令[0,1]位区分）：无条件跳转指令 JUMP，有条件（溢出 C/零标志 Z/符号位 S）跳转指令 JC/JZ/JS。注意：MUL 指令的相乘结果仅保留低 8 位[0,7]，所有移位指令仅 IMM[0,3]位有效。  2. 与上述指令集扩展对应的微程序控制器 CPU（第三阶段）如下图 4.3.2 所示。图中可见第二阶段的数据通路和微程序控制器基本不变，仅输出的微操作信号更多了，因为数据通路不再是从源部件到目标部件的单纯数据传输，而是中间经过了运算和移位电路，需要增加控制运算和移位类型的微操作信号。此外，运算的 C/Z/S 标志位保存到寄存器 ALU\_Flag，以提供有条件跳转指令 JC/JZ/JS 判断使用。  3. 上述 CPU 的指令周期流程图，微指令格式和微指令存储器中的所有微指令编码如下  图 4.3.3 所示。指令周期流程图保持不变，仍是取指和执行两个周期，因此硬布线控制器状态机也保持不变：M1 取指状态和 M2 执行状态。因为指令集扩充至 31 条指令，所以微指令扩充到 15 条（仅地址[1000]微指令未做定义）：由 P2 信号和指令[0,1]位区分指令 LAD 与POP，指令 POP 与 PUSH，（双寄存器运算）指令ADD/SUB/MUL/AND/OR/XOR 与（寄存器+立即数运算）指令 ADDI/SUBI/MULI/ANDI/ORI/XORI  4. 如上图 4.3.2 所示，微指令结构中，微命令字段增加了 ALUop 字段，其作用如下图所示。Ssel 字段选择不同的源部件作为运算电路的输入 Xin，然后根据 Rb 字段为全 0 或非全0 选择立即数 IMM 或通用寄存器输出 Db 作为运算电路的输入 Yin。ALUop 字段选择以下八种运算电路：Xin 直通（兼容第二阶段指令运算），移位（由 Rb 字段选择移位类型）以及 Xin与 Yin 的 ADD 加法，SUB 减法，MUL 乘法，AND 与，OR 或，XOR 异或。最后运算结果经 Asel 选择目标部件输出。107 如上图 4.3.2 所示，运算电路根据运算结果自动给出了溢出标志位 C（加法运算独有），零标志位 Z(所有运算)和符号标志位 S（所有运算）。在下图中，每一次指令的取值周期的P1 上升沿跳变时刻，将三个标志位锁存到寄存器 ALU\_Flag，提供给有条件跳转指令判断。在跳转指令（OP 码 0001）的执行微指令周期，LDPC=1 且根据指令第一字节的 Rb 字段判断是否跳转。当 Rb=00/01/10/11，分别是无条件跳转/C=1 跳转/Z=1 跳转/S=1 跳转指令。  5. 采用上述扩充的指令集，编写了如下图所示的运算程序 ALU.hex，实现了从 1 到 9 的连续相加（采用了运算指令，移位指令和有条件跳转指令）。  请将上述ALU.hex 二进制代码加载到上图 4.3.2所示的第三阶段微程序控制器版本 CPU 电路中。然后还需在微指令存储器 MROM 加载上图 4.3.3 所示的微程序二进制代码。完成加载后，启动运行第二阶段微程序控制器版本 CPU，PC=0，ROM 输出第一条指令 SET 第一字节，OP=0011；微程序控制器内的 MAR=0000,表示当前是取指微指令，P 字段的 P1=1，P2=0，指示下一条微指令地址由 OP 码[0011]决定。随后单步执行，用表格记录每一次 clk后，ROM 存储器的输入（地址 PC）和输出（指令），MROM 存储器的输入（MAR）和输出（微108指令），以及寄存器堆 regs 和输出端口 OUT 的变化。  **实验4：微程序控制器—中断实验**  1. 在上述完整版本指令集基础上，增加单级中断的微程序控制器版本 CPU（最终版本）  如下图 4.4.1 所示。图中可见第三阶段的数据通路和微程序控制器基本不变，仅增加了硬件的中断电路和中断返回信号 ret 生成电路。  2. 因为进入中断是硬件实现，而返回中断需要扩充一条指令，所以，必须增加一条中  断返回指令 IRET（单字节，OP 码 1000），对应的执行微指令地址[1000]。IRET 指令与 HLT指令相似，没有使用任何微操作信号，由 OP 码 1000 硬件生成中断返回信号 ret，如下图 4.4.2左所示。由下图 4.4.2 右的中断处理过程时序图可见：任何时刻，经由按键 key 模拟的外部中断被 D 触发器锁存，然后在某条指令 n+1 的取指周期(P1=1)，由 P1 上升沿触发形成中断信号int=1，然后在下一个周期的 clk 上升沿清除中断信号 int，即中断信号 int=1 仅维持一个周期。当 int=1 的周期，PC 值从指令 n+1 对应的 PCint 切换成由外部 ISR 端口设置的中断服务程序入口 isr；同时，指令 n+1 的第一字节也被替换成中断服务程序入口指令的第一字节。在下一个周期（isr 指令的执行微指令周期），PCint 打入断点寄存器 BP\_pc，标志位寄存器ALU\_Flag 锁存的标志位信息打入断点寄存器 BP\_Flag；与此同时，int 清 0，PC 从中断服务程序开始运行。此外，当中断服务程序结束时，末尾的 IRET 指令的取指周期，OP 码 1000 硬件生成中断返回信号 ret=1。当下一个周期（IRET 指令的执行周期），断点 mepc 弹回 PC 寄存器，标志位断点准备写回 ALUflag 寄存器。  3. 为验证上述 CPU 的中断功能，我们改写了运算程序 ALU\_int.hex，添加了中断服务程序输出当前运算结果（寄存器 R0 的值），如下图所示：  请将上述 ALU\_int.hex 二进制代码加载到上图 4.4.1 所示的第三阶段微程序控制器版本CPU 电路中。然后还需在微指令存储器 MROM 中补充图 4.4.2 所示的微指令二进制代码。完成加载后，启动运行具有中断功能的微程序控制器版本 CPU，PC=0，ROM 输出第一条指令SET第一字节，OP=0011；微程序控制器内的 MAR=0000,表示当前是取指微指令，P字段的 P1=1，110 P2=0，指示下一条微指令地址由 OP 码[0011]决定。随后自动运行到断点 break。然后按下 key 提供一个外部中断，被 D 触发器锁存。手动使 CPU 从断点 break 跳出继续运行，待到当前指令执行完毕，中断信号 int=1，PC 已经切换到中断服务程序入口（PC=20H），ROM 输出中断服务程序第一条指令 OUT 的第一字节，OP 码 0101。微程序控制器内的MAR=0000,P 字段的 P1=1，P2=0，当前是中断服务程序第一条指令 OUT 的取指微指令。用表格记录每一次 clk 后，ROM 存储器的输入（地址 PC）和输出（指令），MROM 存储器的输入（MAR）和输出（微指令），以及寄存器堆 regs 和输出端口 OUT 的变化。 同时记录中断信号 int，断点寄存器 BP\_pc 和 BP\_Flag，中断返回信号 ret，ALU\_Flag 寄存器的变化情况。    二、实验数据：（“实验步骤”里的列表部分）  **实验1：微程序控制器和硬布线实验**  **1.1微程序控制器**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | clk输入 | ROM输入 | ROM输出 | MROM输入 | MROM输出 | | 1 | **1** | **0** | **0b0001** | **0x1100** | | 2 | **3** | **1** | **0b0000** | **0x120** | | 3 | **4** | **3** | **0b0000** | **0x1100** | | 4 | **5** | **4** | **0b0001** | **0x120** |   **1.2硬布线控制器版本**   |  |  |  |  | | --- | --- | --- | --- | | clk输入 | ROM输入 | ROM输出 | 状态机Mx | | 1 | **1** | **0** | **M1** | | 2 | **3** | **1** | **M2** | | 3 | **4** | **3** | **M1** | | 4 | **5** | **4** | **M1** |   **实验2：微程序控制器—存储与IO访问实验**    **实验3：微程序控制器—运算器实验**    **实验4：微程序控制器—中断实验** |
| **小结** |
| 这是第四次计组实验了，也即将接近尾声。这次实验给我带来的最大感受就是：太难了！跟以往三次实验的难度完全不一样，前三次都是填表格的形式，而这次的四个小实验都是自主设置记录数据的格式，而且还有将汇编代码转换到电路图的操作，真的是让我一开始一头雾水，所以又要花大把时间来搞懂这些知识。其中我觉得最难的是将.hex文件中的汇编代码内容设置到电路中，说实话当时看懂指令我就花了好长时间，之后也是慢慢推导菜做完了所有内容。总之，这次实验对我来说是一次巨大的挑战，但我还是坚持做完它了，这也是一种胜利！ |
| **指导教师评语及成绩** |
| 评语：  成绩：           指导教师签名：                                                 批阅日期： |