

数字逻辑电路实验报告

第十三次实验：简易计算机系统

张铭方

161220169

16 级计算机系 5 班

161220169@smail.nju.edu.cn

许致明

161180162

16 级计算机系 5 班

161180162@smail.nju.edu.cn

2017 年 12 月 26 日

第一节 实验目的

本实验的目标是在 Nexys 4 开发板上实现一个简单的计算机系统，能够运行简单的指令，包括循环、整数计算、函数调用、递归等。这些指令使用 RISC 方式编写，存储在开发板的存储器（ROM）中。开发板的另一部分存储器（RAM）用来保存程序运行中所需的数据。此外，在完成后，开发板还具有一定的输出功能，程序输入通过向 ROM 中初始化相应的机器码实现。

第二节 实验原理

一 计算机系统简介

计算机系统主要由 CPU 和外部设备组成。CPU 是系统中最重要的一部分，它负责控制系统运行和信息处理。外部设备负责和外界进行交互，使得计算机的可以接受输入，产生相应的输出。本实验需要实现一个简化的计算机系统。下面对两种基本的系统结构做简要介绍。

第一种是冯·诺伊曼结构，这种结构被现代的大多数 CPU 所使用。在这种结构下，处理器使用同一个存储器，经过同一个总线传输，具有以下特点：

1. 结构上由运算器、控制器、存储器和输入/输出设备组成；
2. 存储器是按地址访问的，每个地址是唯一的；
3. 指令和数据都是以二进制形式存储的；
4. 指令按顺序执行，即一般指令按照存储顺序执行，程序的分支、循环由转移指令实现；
5. 以运算器为中心，在输入输出设备与控制器之间的数据传送都途径运算器。运算器、存储器、输入输出设备的操作以及它们之间的联系都由控制器集中控制。

第二种是哈佛结构，它使用两个独立的存储模块，分别存储指令和数据，并具有一条独立的地址总线和一条独立的数据总线，具有以下特点：

1. 每个存储块都不允许指令和数据并存，以便实现并行处理；
2. 利用公共地址总线访问两个存储模块（程序存储模块 ROM 和数据存储模块 RAM），公用数据总线则被用来完成程序存储模块或数据存储模块与 CPU 之间的数据传输；
3. 地址总线和数据总线由程序存储器和数据存储器分时共用。

数字信号处理一般需要较大的运算量和较高的运算速度，为了提高数据吞吐量，在数字信号处理中大多采用哈佛结构。本实验所构建的计算机系统就采用了哈佛结构。

二 RISC 架构简介

2.2.1 RISC 架构的基本特征

RISC, 即 Reduced Instruction Set Computer, 是精简指令集计算机的简称, 与它相对的是 CISC (Complex Instruction Set Computer)。RISC 架构主要具有以下特点:

1. 只包含一些使用频率较高的指令, 并用这些指令的组合来实现较为复杂指令的功能;
2. 指令长度固定, 指令格式、寻址方式比 CISC 少;
3. 只有加载、存储两条指令需要访问内存, 其他指令都是在寄存器和寄存器或寄存器和立即数之间进行操作;
4. CPU 中包含多个通用寄存器, 执行指令过程中的数据均暂存在寄存器中, 提高指令的执行速度;
5. 常常采用流水线技术, 这样大部分指令可以在一个时钟周期内完成。还可以采用超标量和超流水线技术, 使指令平均执行时间小于一个时钟周期;
6. 控制器采用组合逻辑的控制方式, 不使用微程序控制的方式。

CPU 是计算机中的核心部件。RISC 架构中的 CPU 进行信息处理时, 主要进行如下两个步骤:

1. 将数据和指令 (二进制串) 读入到计算机的存储器中;
2. 从第一条开始, 按顺序执行程序, 直至停机, 结束运行。

这一过程还可以用伪代码表示如下:

CPU-EXECUTE

```
1 while TRUE
2   FETCH instruction Instr[PC]
3   DECODE Instr[PC]
4   EXECUTE Instr[PC]
5   PC = PC + 1
```

为了实现这些操作, CPU 至少需要具有以下功能:

1. 取指令: 当程序已经在存储器 (ROM) 时, 首先根据程序入口地址取出一条指令。需要 CPU 能发出正确的地址信息和产生控制读取存储器的信号;
2. 指令译码: 这一操作即需要分析出此二进制串的意义, 获得它指示的操作内容, 并且产生相应的操作控制命令;

3. 执行指令：根据指令译码得到的结果，产生相应的操作控制信号序列，控制运算器、存储器、输入输出设备的动作，完成这条指令的功能。其中包含对运算结果的处理（如设置标志位）以及下一条指令的地址的形成（如根据跳转指令更改 PC 的值）。

总而言之，CPU 做为计算机系统的核心，主要的任务就是取出指令，解释指令，然后根据得到的结果执行相应的指令。在这些过程中，可能涉及到存储器（例如内存、内部寄存器）的读取、写入，PC 内容更改（自增 1 或按条件跳转），以及和外部设备的数据交换（接收外设传来的输入信号，将输出信号发送给外设）。同时，CPU 也需要产生相应的控制信号，使得其他部件能正确的工作。

2.2.2 RISC 架构下 CPU 的基本构成

RISC CPU 主要包括三方面的功能：数据存储、数据运算、时序控制。与此对应的就是三方面的硬件设备：寄存器和内存、运算器 ALU、控制器。其基本结构如下面的图 1 所示：

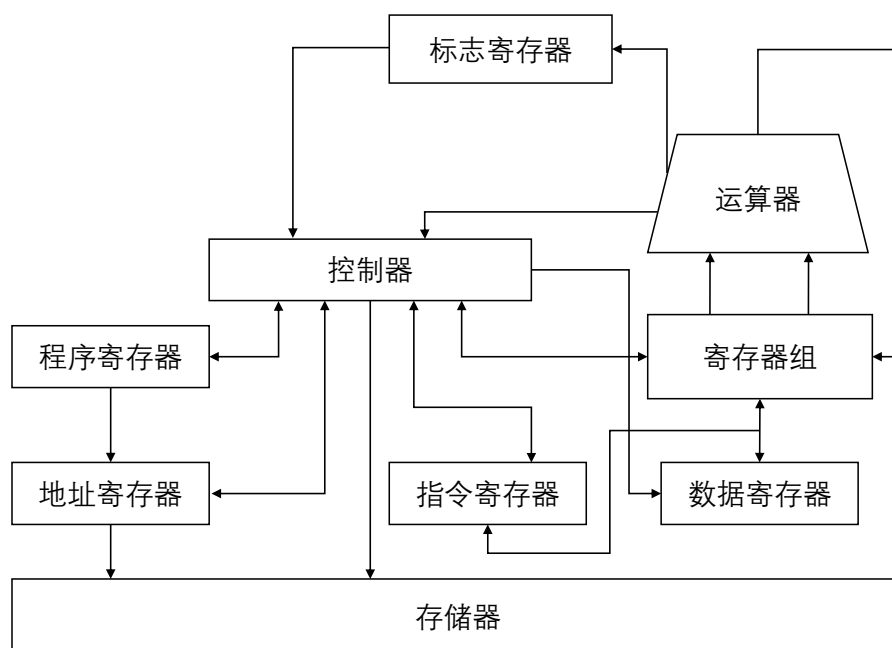


图 1: RISC 架构 CPU 的基本结构

寄存器用于存放指令和数据，在此 CPU 设计中，包含了较多寄存器，这符合 RISC 架构的主要思想，即将指令执行过程中的各种数据存入到不同的寄存器中，减少访问速度较慢的内存的次数，从

而提高处理器的运行速度。图中的箭头指明了数据在这条通路上的传输方向。通过多条通路，指令和数据能够独立传输、互不干扰，这样也能提高 CPU 的运行速度。

三 CPU 的指令系统设计

在此次实验中，我们完成了一个 16 位的 RISC 架构指令集，共包含 32 条 5 个二进制位的定长指令，具有算术运算、存储读写、输入输出和逻辑控制这些基本功能。具体如表 1 所示：

表 1: 指令系统概要

汇编指令格式	操作数	功能描述	类型
ADC %SR, %DR	00000	$R[DR] + R[SR] + CF \rightarrow R[DR]$	算数逻辑
SBB %SR, %DR	00001	$R[DR] - R[SR] - CF \rightarrow R[DR]$	
MUL %SR, %DR	00010	$R[DR] * R[SR] \rightarrow R[DR]$	
DIV %SR, %DR	00011	$R[DR] / D[SR] \rightarrow D[DR]$ （无符号）	
ADDI \$IMM, %DR	00100	$R[DR] + \$IMM \rightarrow R[DR]$ （寄存器与立即数相加）	
CMP %SR, %DR	00101	$R[DR] - R[SR]$ （只改变标志寄存器，若相等，则 ZF=1）	
AND %SR, %DR	00110	$R[DR] \& R[SR] \rightarrow R[DR]$ （按位与）	
OR %SR, %DR	00111	$R[DR] R[SR] \rightarrow R[DR]$ （按位或）	
NOT %DR	01000	$\sim R[DR] \rightarrow R[DR]$ （按位取反）	
XOR %SR, %DR	01001	$R[DR] \wedge R[SR] \rightarrow R[DR]$ （按位异或）	
TEST %SR, %DR	01010	$R[DR] \wedge R[SR]$ （根据结果改变标志寄存器 ZF）	
SHL %DR	01011	$R[DR] \ll 1 \rightarrow R[DR]$ （逻辑或算数左移，最高位入 CF）	
SHR %DR	01100	$R[DR] \gg 1 \rightarrow R[DR]$ （逻辑右移，高位补 0，最低位入 CF）	
SAR %DR	01101	$R[DR] \gg 1 \rightarrow R[DR]$ （算术右移，高位补符）	
IN PORT, %DR	01110	$[PORT] \rightarrow R[DR]$	I/O
OUT %SR, PORT	01111	$R[SR] \rightarrow PORT$	
MOV %SR, %DR	01110	$R[SR] \rightarrow R[DR]$	数据传送
MOVL \$IMM, %DR	01111	$IMM \rightarrow R[DR][0..7]$ （8 位立即数移入寄存器高 8 位）	
MOVH \$IMM, %DR	10000	$IMM \rightarrow R[DR][8..15]$ （8 位立即数移入寄存器低 8 位）	
LOAD %SR, %DR	10001	$M[R[SR]] \rightarrow R[DR]$	访存
STORE %SR, %DR	10010	$R[SR] \rightarrow M[R[DR]]$	
PUSH %SR	10101	$R[SR] \rightarrow M[R[SP]], R[SP] - 1 \rightarrow R[SP]$ （SR 入栈）	栈操作
POP %DR	10110	$M[R[SP]] \rightarrow R[DR], R[SP] + 1 \rightarrow R[SP]$ （DR 出栈）	
JMP OFFSET	10111	$PC + OFFSET \rightarrow PC$ ，无条件跳转指令，OFFSET 有符号	控制转移
JMPC	11000	当 CF=1 时进行跳转	
JNC	11001	当 CF≠1 时进行跳转	
JMPZ	11010	当 ZF=1 时进行跳转	
JNZ	11011	当 ZF≠1 时进行跳转	
CALL	11100		
RET	11101		
NOP	11110	空操作， $PC + 1 \rightarrow PC$	处理器
HALT	11111	停机	

我们设计出的这个指令规定数据字长也为 16 位。指令格式固定，操作数寻址方式共有 3 种。大多数均为寄存器寻址和立即数寻址，只有 LOAD/STORE 两条指令涉及到内存寻址。指令的格式有图 2 所示的三种，对其中的符号做如下说明：

1. *Opcode* 为操作数，即指令表中对应的第二栏；
2. *%SR* 是源操作数所在的寄存器，*%DR* 是目的操作数所在的寄存器；
3. *\$IMM* 是立即数，受指令长度限制，所有的立即数均为 8 位二进制数；
4. *Offset* 是偏移量，为有符号数，符号由 *Offset* 域的最高位决定，0 为负，1 为正，由此给定后 8 位立即数的符号。

指令类型一：

<i>Opcode</i>	<i>%SR</i>	<i>%DR</i>	<i>Undefined</i>
---------------	------------	------------	------------------

指令类型二：

<i>Opcode</i>	<i>%DR</i>	<i>\$IMM</i>
---------------	------------	--------------

指令类型三：

<i>Opcode</i>	<i>Offset</i>
---------------	---------------

图 2: 指令格式类型

第三节 实验器材和环境

一 硬件

Nexys 4 开发板

二 环境

运行在 Windows 10 下的 Vivado 2016.4

第四节 实验设计思路

一 RISC CPU 的数据通路图

我们设计的 CPU 数据通路如下图所示:

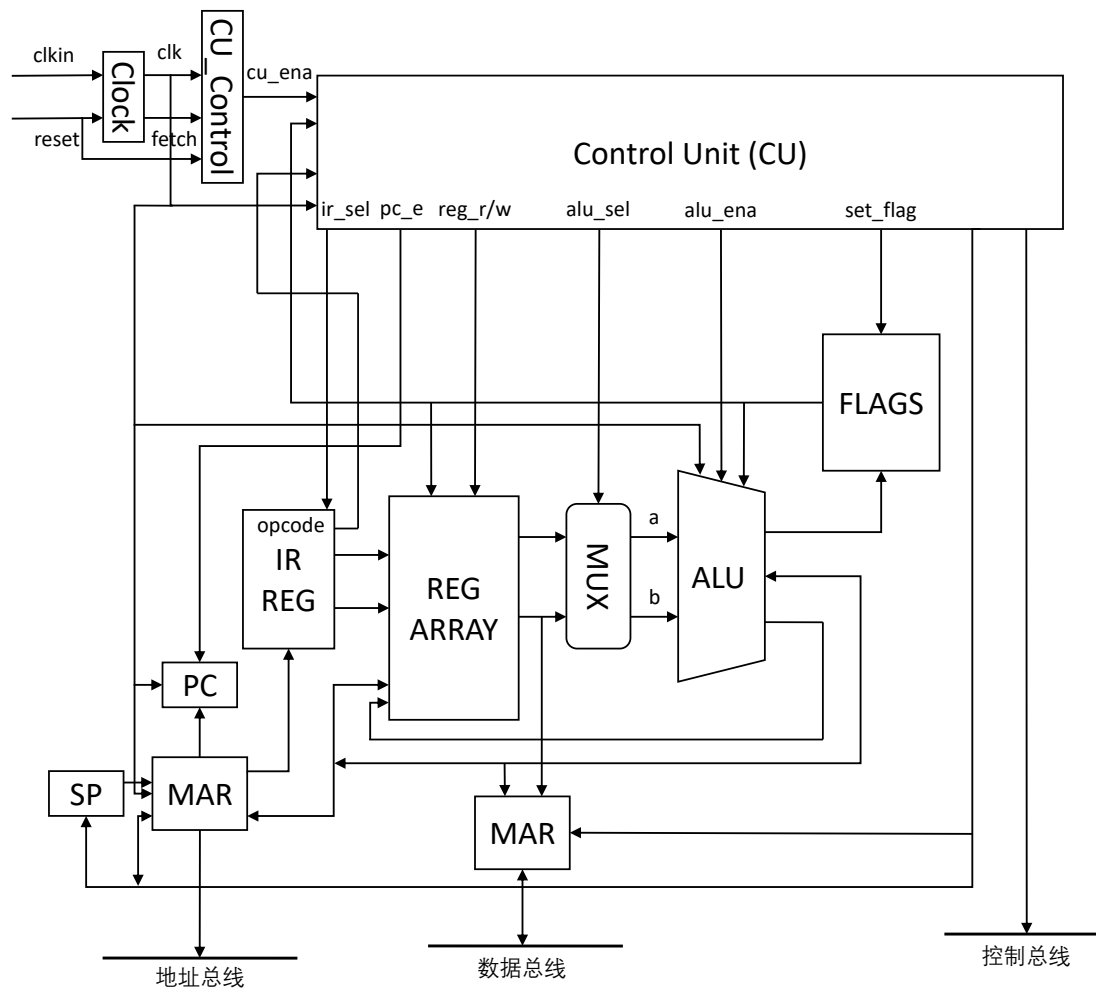


图 3: CPU 的数据通路

我们在这个 CPU 中设计了 8 个 16 位的按照 16 位可读写的通用寄存器组（图中 REG_BANK）。除此之外，还包含指令寄存器（图中 IR REG），地址寄存器（图中 MAR），数据寄存器（图中 MDR），栈指针寄存器（图中 SP, Stack Pointer）以及标志寄存器（图中 FLAG）。运算器为算术逻辑单元（图中 ALU），负责进行数学运算和逻辑运算。程序计数器在图中为 PC。控制部分为图中的 Control Unit (CU)，时钟信号由图中 Clock 发生。

二 模块划分

按照功能，我们将 CPU 划分为如下 11 个功能模块，下面逐个介绍这些模块的功能：

1. 时钟发生器 (Clock)：时钟发生器的功能是根据外部时钟信号，产生一系列的特定时钟信号，送往其他部件，控制它们进行操作；
2. 控制器 (Control Unit, CU)：控制器是 CPU 最核心的部件。CPU 通过它来产生控制时序和控制信号，用来控制 CPU 中其他模块（例如 ALU、寄存器）按照一定的时序关系工作。这一部分由两部分组成：状态机 (CU) 和状态机的控制器 (CU_Control)；
3. 指令寄存器 (IR REG)：指令寄存器用来存放当前正在执行的 (PC 指向的) 指令；
4. 通用寄存器组 (REG ARRAY)：这个寄存器组全部用于存储数据，它由 8 个 16 位寄存器组成。此部件支持双端口读出、双端口写入操作。这些寄存器用来保存程序执行过程中需要的数据、中间变量和最后的结果，在访问时，必须一次性读出全部 16 位数据；
5. 程序计数器 (PC, Program Counter)：程序计数器保存下一条指令的地址，CPU 使用它在相应的 ROM 中取出指令；
6. 算术逻辑运算器 (ALU, Arithmetic and Logic Unit)：算术逻辑运算器是一个 16 位定点运算器，支持基本的加减乘除、与或非等 14 算数、逻辑运算；
7. 运算输入控制部件：此部件用来控制运算器的输入数据。送入 ALU 的数据只有两种：来自数据寄存器的 16 位数据和来自立即数的 8 位数据；
8. 标志寄存器 (FLAG)：标志寄存器共有 8 位，但是只使用了 4 位。高 4 位未定义，低 4 位存储运算过程中产生的标志位。低 4 位由高到低依次为：进位 (CF)、零 (ZF)、溢出 (OF) 和符号 (SF)；
9. 地址寄存器 (MAR, Main Address Register)：地址寄存器用来存储 CPU 访存时给出的地址，访存时的地址来源于此部件；
10. 数据寄存器 (MDR, Main Data Register)：数据寄存器用来存储需要向地址总线输出的数据，或存储从总线上读取到的数据。它是双向输入输出的；
11. 栈指针寄存器 (SP, Stack Pointer)：栈指针寄存器用于存储当前 1 的 16 位栈顶地址。每次出栈或入栈操作后，都要更新它的值。

这些模块的具体实现将在 4.4 节介绍。

三 指令执行流程

为了更清楚的描述此 CPU 的运行过程，下面介绍指令的执行流程。总体上，所有指令的执行均包含 3 个阶段：取出指令、指令译码和取出操作数、执行和回写。这其中每个阶段又分为 3 个小阶段，所以执行中共经历了 9 个阶段。负责此模块实现的控制单元（CU）中的状态机也就在 9 个状态间进行转移。

1. 算数逻辑运算指令：先把数据从寄存器中取出，然后经过运算器处理后再将结果写回寄存器，或设置相应的标志位。下面以带进位的加法指令 ADC 为例，介绍执行过程的各个阶段：¹

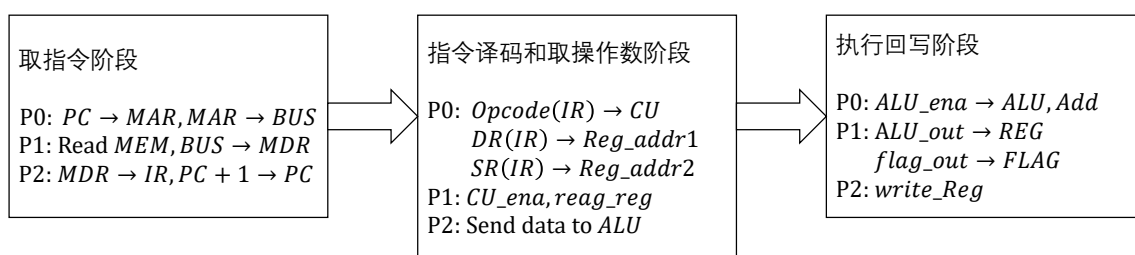


图 4: 算数和逻辑运算指令执行过程，以 ADC 为例

2. 数据传送类指令：此类指令包含在寄存器之间传输数据的 MOV，寄存器的高 8 位或低 8 位的加载指令（MOVIH/L \$IMM, %DR）。下面以寄存器间的传送指令 MOV 为例，介绍执行过程的各个阶段：

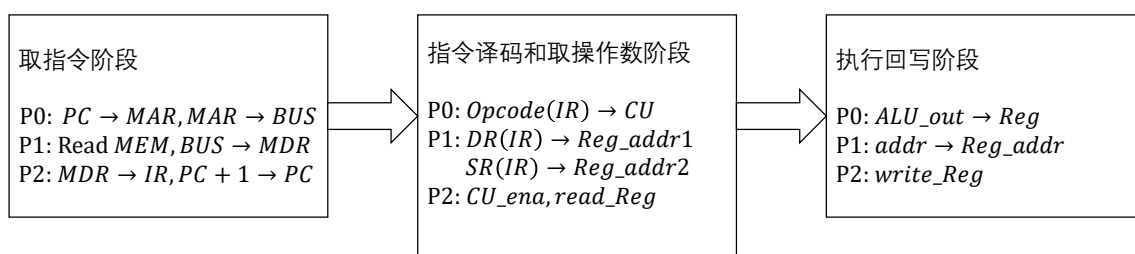


图 5: 数据传送类指令执行过程，以 MOV 为例

3. I/O 类指令：此类指令包含写 I/O 端口指令（OUT %SR, PORT）和读 I/O 端口指令（IN PORT, %DR）。下面以 OUT 指令为例，介绍执行过程的各个阶段：

¹与 ADC 执行过程相似的指令还有：SBB, DIV, MUL, AND, NOT, OR, XOR, SHL, SHR, SAR, CMP, TEST 和 ADDI 等。

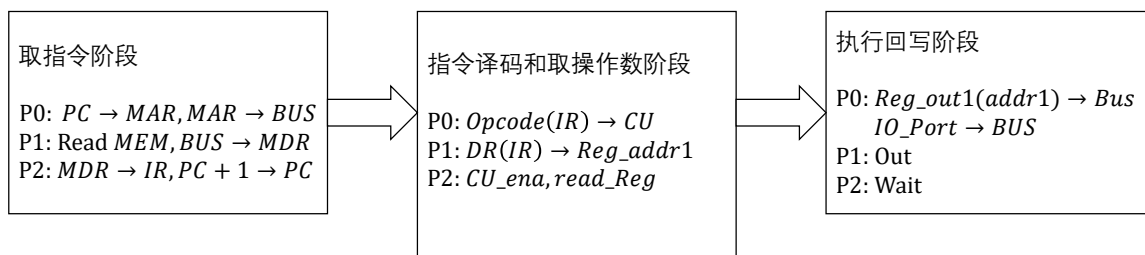


图 6: I/O 指令执行过程, 以 OUT 为例

4. 控制转移类指令: 可以分为有条件转移指令²和无条件转移指令³两大类。下面以 JMPC 为例, 介绍执行过程的各个阶段:

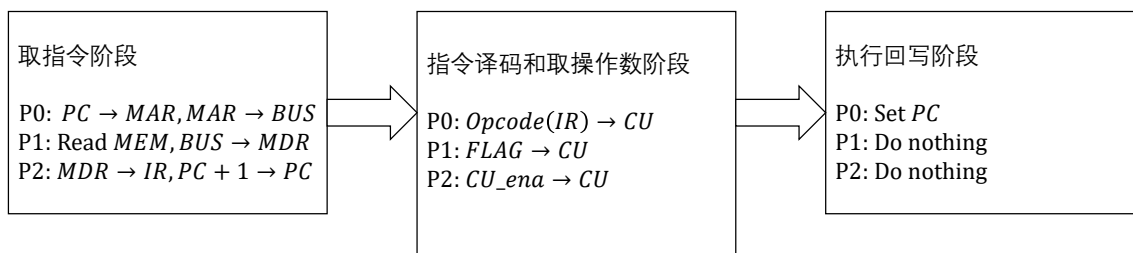


图 7: 条件转移类指令执行过程, 以 JMPC 为例

5. 栈操作指令: 包含入栈 (PUSH %SR) 和出栈 (POP %DR) 指令。下面以出栈指令 POP 为例, 介绍执行过程的各个阶段:

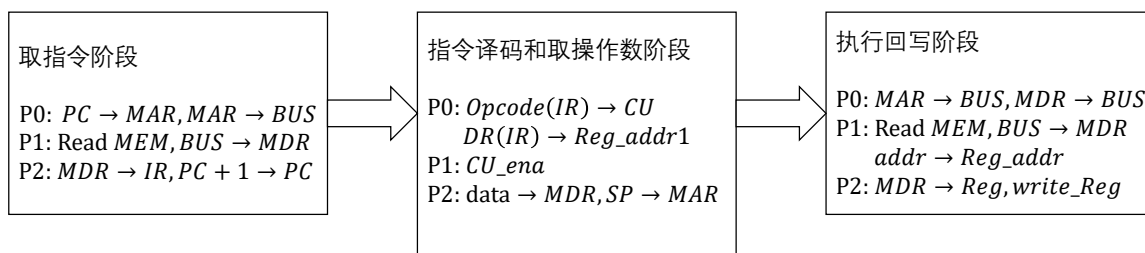


图 8: 栈操作指令执行过程, 以 POP 为例

6. 访问内存类指令: 包含取操作数指令 (LOAD %SR, %DR) 和存操作数指令 (STORE %SR, %DR)。

²此类指令包括 JMPC, JNC, JMPZ 和 JNZ。

³此类指令为 JMP

LOAD 指令的作用是将源操作数寄存器中的数据当作地址，将这个内存地址中的数据装入目的寄存器%DR；STORE 指令的作用是将目的操作数寄存器中的数据当作地址，将源操作数中的数据存入此地址指向的内存。LOAD 和 STORE 的执行过程分别如下所示：

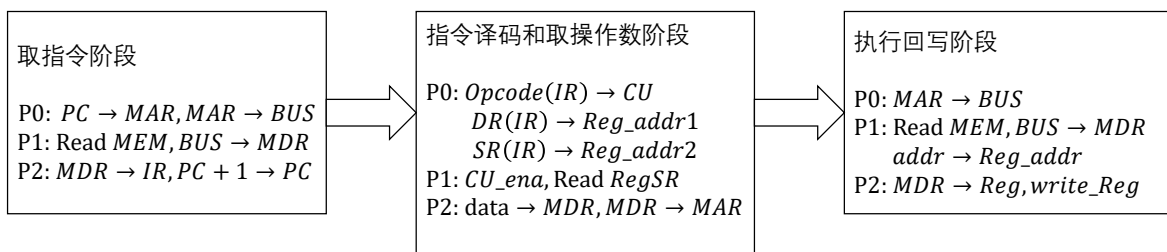


图 9: LOAD 指令执行过程

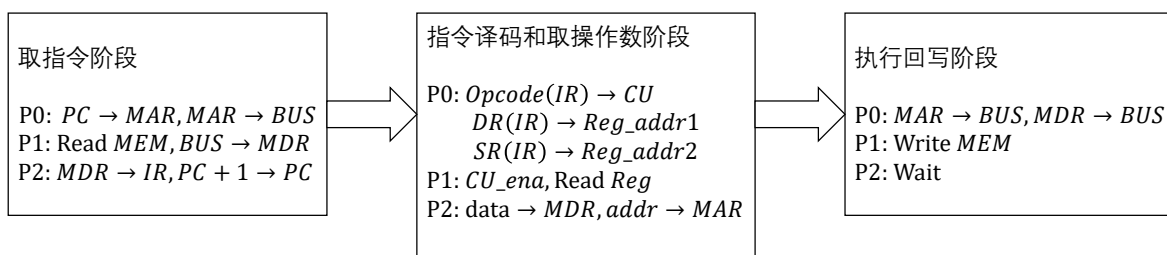


图 10: STORE 指令执行过程

7. 处理器控制类指令：包含空操作（NOP）和停机（HALT）两个指令。NOP 不执行任何操作，只将 PC 增加 1；HALT 指令负责停止 CPU 的运行。它们的执行过程分别如下图所示：

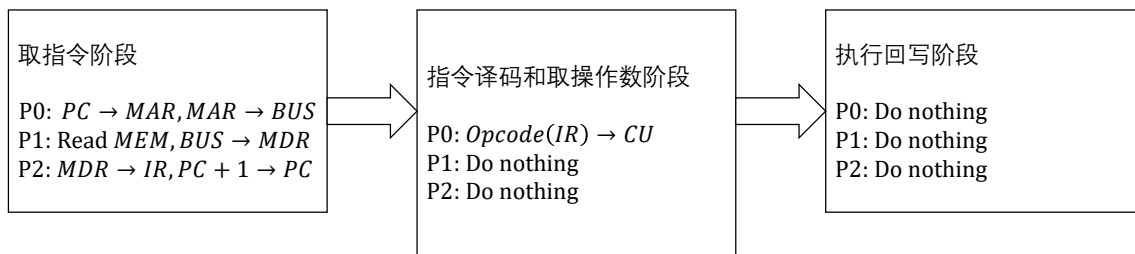


图 11: NOP 指令执行过程

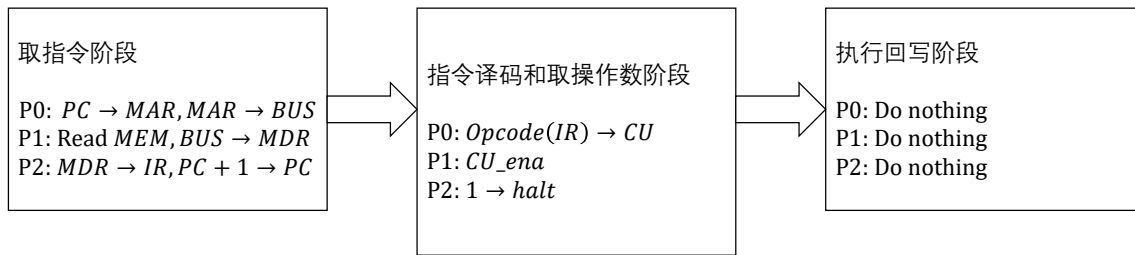


图 12: HALT 指令执行过程