**汇编语言实验报告**

学号：18342113

姓名：颜府

目录

[实验目标 3](#_Toc529034403)

[实验工具 3](#_Toc529034404)

[任务1 4](#_Toc529034405)

[任务2 7](#_Toc529034406)

[实验小结 10](#_Toc529034407)

实验目标

1. 理解冯·诺伊曼计算机的结构

2. 理解机器指令的构成

3. 理解机器指令执行周期

4. 用汇编编写简单程序

实验工具

本次实验将以一台名为PIPPIN的机器为工具，在其上编写汇编程序并运行。鉴于这只是一个概念上的虚拟机，实验使用因特网上的一个[Java applet](http://www.science.smith.edu/~jcardell/Courses/CSC103/CPUsim/cpusim.html)，这是一个PIPPIN机器的CPU模拟器，它模拟了CPU里面的所有基本组件，包括ALU、IR、PC等在内，能形象地演示出CPU是如何工作的。

任务1

先把以下用汇编语言写的程序输入至模拟器里，观察CPU模拟器的运行情况：

LOD #3

STO W

LOD #7

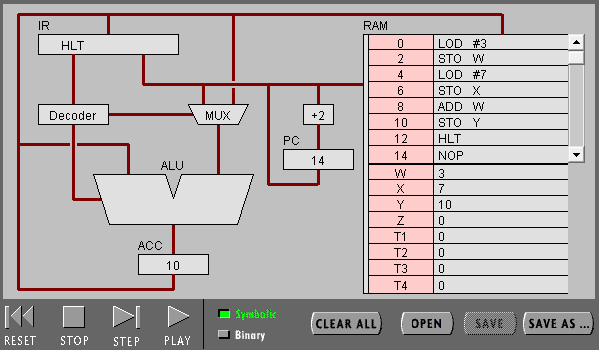
STO X

ADD W

STO Y

HLT

运行后的模拟器如图：



单步执行之后，我们可以发现：

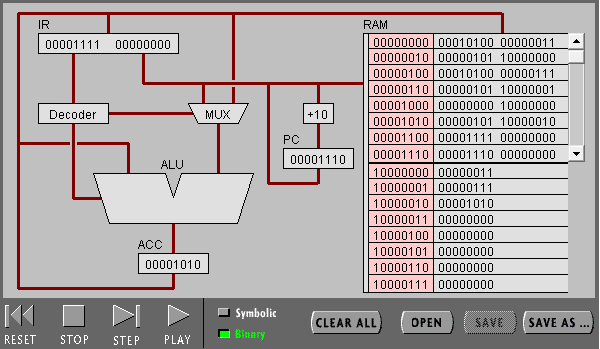
1. PC存储一个字节，表示下一条指令的地址；IR存储两个字节，表示当前指令的内容（PIPPIN机器的每条指令为2字节）。

2. ACC的全称是Accumulator，中文翻译是“累加器”。其作用是保存算术逻辑运算的中间值，如果没有ACC，那么就算下一条指令就要用到这次运算的结果，也不得不把这个结果先写回内存，到下一条指令的时候再重新读一次，这样子效率很低。

3. 由“LOD #3”指令我们可以看出，CPU先把指令Fetch到IR里，通过Decoder之后识别出这是一个LOD指令，然后直接把指令中表示“3”的内容发送到ACC里。

4. 由“ADD W”指令我们可以看出，CPU先把指令Fetch到IR里，通过Decoder之后识别出这是一个ADD指令，之后先把ACC里的值传给ALU中的加法器作为第一个操作数，再通过指令中的地址，获取内存中对应地址的值，传给加法器作为第二个操作数，接着执行加法操作。

把所有数据和指令转换为二进制表示，如图：



我们可以发现：

1. “LOD #7”的二进制表示为“00010100 00000111”。PIPPIN机器的指令由两个字节构成，第一个字节的高4位表示寻址模式，此处0001表示Immediate，说明operand（第二个字节）表示的就是将要操作的数值（而不是将要操作的数的地址）；第一个字节的低4位表示opcode，此处0100表示LOD操作；第二个字节表示operand，具体是一个数值还是地址取决于寻址模式，这里00000111显然表示的就是7的二进制。

2. RAM的地址就是某个字节的地址，PIPPIN机器内存为256（也就是28）字节，刚好用8位来表示每个字节的地址。

3. 由ACC的位数可以看出，PIPPIN机器的CPU为8位的。

4. 该程序改为C语言如下，int\_8 表示的是8位的整型：

**int\_8** W = 3;

**int\_8** X = 7;

**int\_8** Y = 7 + W;

*// 因为ADD W前没有LOD X指令，所以这里应该理解为7加上W，而不是X + W*

任务2

先把以下用汇编语言写的程序输入至模拟器里，观察CPU模拟器的运行情况。

LOD X

SUB #1

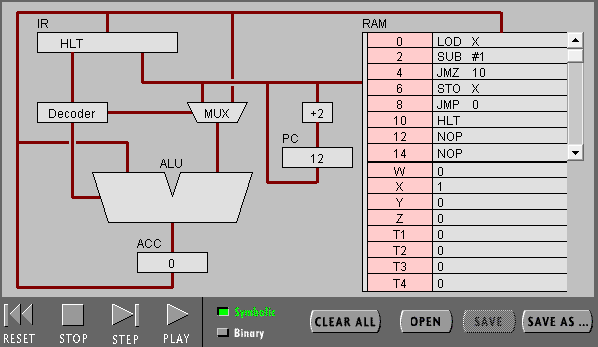
JMZ 10

STO X

JMP 0

HLT

运行后的模拟器如图：



我们可以发现：

1. 这个程序的功能就是根据X的初始值，循环X次，虽然“循环体”里面没有内容。

2. 该程序对应的C语言程序为：

**int\_8** X = 3;

**while** (X – 1 != 0) {

X = X – 1;

}

现在稍微修改以下程序，来计算10 + 9 + … + 1的结果。

1. 先写一个C语言的程序：

**int\_8** X = 10;

**int\_8** Y = 0;

**while** (X – 1 != 0) {

Y = Y + X;

X = X – 1;

}

2. 接下来改写成对应的汇编语言。观察后发现，原来的程序的循环结尾处，先STO了X，下一次循环的开头又LOD一次，这浪费了时间，因此这里把它改掉：

LOD X

ADD Y

STO Y

LOD X

SUB #1

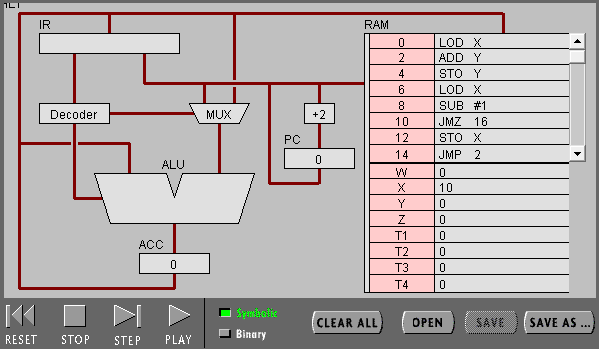
JMZ 16

STO X

JMP 2

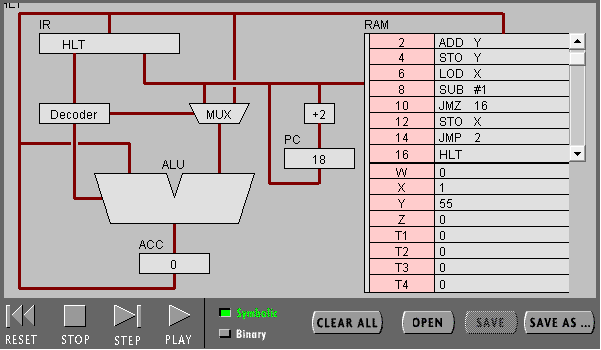
HLT

输入进模拟器，如图：



（内存为16的指令为“HLT”，模拟器显示不下）

运行后的结果如图：



可以发现内存Y中存了正确的结果55。

3. 高级语言与机器语言（或汇编语言）既有区别，也有联系。区别在于，机器语言可以直接对系统硬件进行操作，比如可以改变寄存器里的值；而高级语言一般不可以对硬件进行操作。联系在于，高级语言的一条语句或表达式一般对应着多条机器语言的指令。

实验小结

通过一个PIPPIN机器的CPU模拟器，可以看到冯·诺伊曼结构中最重要的组件——CPU中的各种更详细的组成部分，包括ALU、IR、PC、ACC、Ram、Decoder、MUX在内，它们之间通过总线来通信。

机器指令在机器中与数据是同等地位的，从某种角度，指令其实也是数据，它们与数据一样，也是用一串二进制码来表示的。一条指令的不同部分分别代表了一些指令的信息，包括寻址模式、opcode、操作数。

机器的Fetch-Decode-Execute可以通过模拟器的动画清晰地观察：首先控制器先根据PC的值，从内存中Fetch一条指令，保存到IR里。接着，Decoder把这条指令解码，CPU便识别出这条指令的内容了。最后，CPU根据指令，进行对应的操作。

在上述几例，已经使用PIPPIN机器的机器语言（或其对应的汇编语言）编写了一些简单的程序，并能完成一些简易小功能。

总之，通过PIPPIN机器的CPU模拟器，已经大致了解了计算机底层硬件的工作原理。