硬件编程–机器指令编程

--利用机器指令控制冯诺依曼计算机执行程序。

学号：18342029

姓名：黄进

目录

1. 实验目标：

·理解冯·诺伊曼计算机的结构

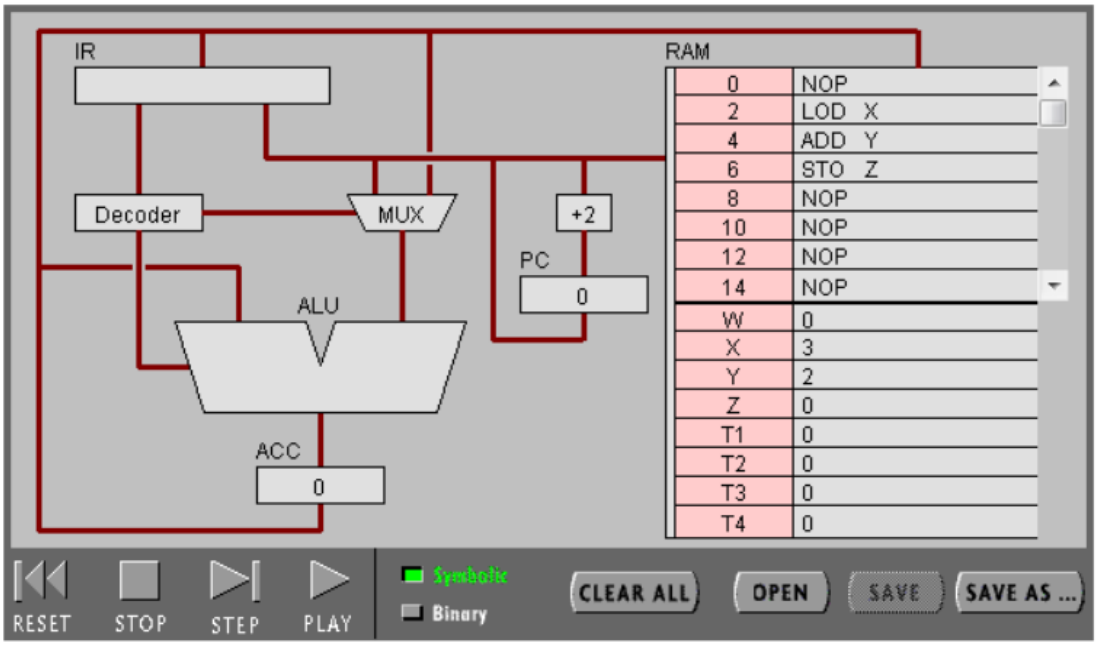
·理解机器指令的构成

·理解机器指令执行周期

·用汇编编写简单程序

2. 实验步骤及结果

任务 1：Add 2 number



A .

（1）指令寄存器（IR ）：

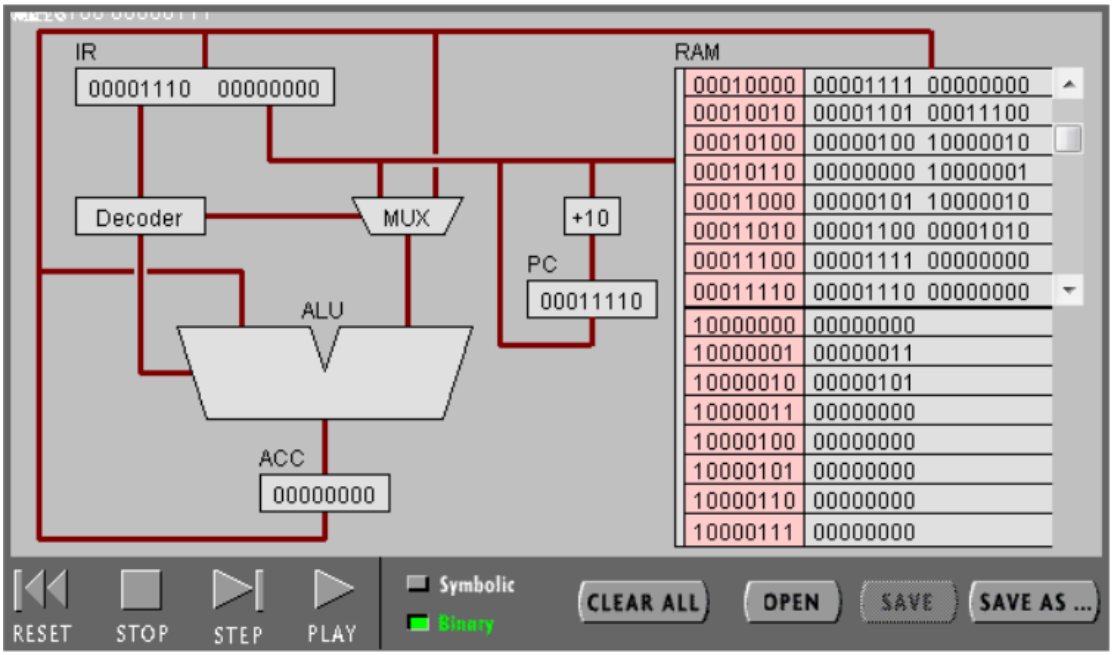
用来保存当前正在执行的一条指令。

程序计数器（PC）：

为了保证程序能够连续地执行下去，CPU 必须具有某些手段来确定下一条指令的地址。在程序开始执行前，必须将它的起始地址，即程序的一条指令所在的内存单元地址送入 PC，PC 中存放着正在执行的指令的地址码，计算机根据 PC 中的地址去 RAM 中进行寻址，找到相应地址中的内容，然后把它输送到指令寄存器 IR 中。

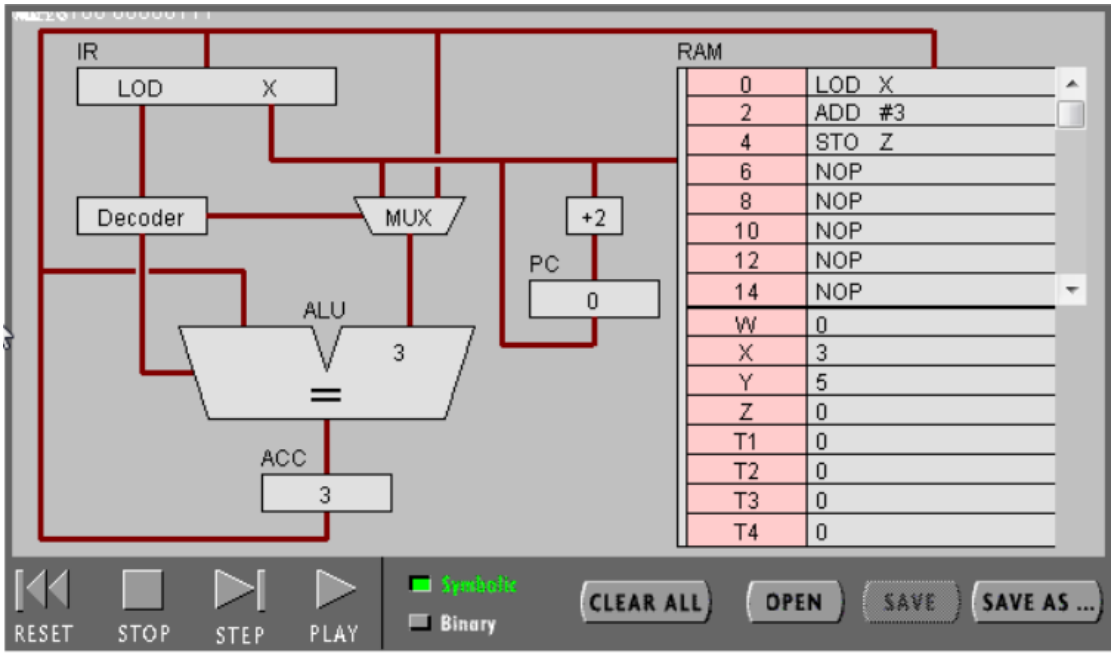
因此程序计数器（PC）的内容即是从内存提取的第一条指令的地址。当执行指令时，CPU 将自动修改 PC 的内容，即每执行一条指令 PC 增加一个量，这个

量等于指令所含的字节数，以便使其保持的总是将要执行的下一条指令的地址。当程序转移时，转移指令执行的最终结果就是要改变 PC 的值，此 PC 值就是转去的地址，以此实现转移。



（2）ACC（Accumulator）是累加器 A 缩写。

累加器 A 是一个二进制 8 位寄存器，专门用来存放操作数或运算结果。在 CPU 执行某种运算前，两个操作数中的一个通常应放在累加器 A 中，运算完成后累加器 A 中便可得到运算结果，然后回到 RAM 中储存为 Z。



（3）“LOD#3”指令执行过程

首先，计算机按照 PC 中的地址码（就是该指令的地址码），去 RAM 中寻找相应的指令，并将这条指令运送到了 IR 中。(Fetch the next instruction)

然后，指令从 IR 中被转移到了 Decoder 中，对指令进行解码过程，判断该指令要进行何种操作，并且判断该指令中用到的操作数在哪里。（即是在指令中就含有还是要再到 RAM 中读取）。（Decode the instruction)

因为 LOD #3 中的操作数本身就在这条指令中，所以不再需要额外从内存中读取。不进行 Get data if needed

最后，“3”这个数字通过数据选择器被存放到了 ACC 中。(Execute the instruction)

(4) “ADD W” 指令执行过程

首先，计算机按照 PC 中的地址码（就是该指令的地址码），去 RAM 中寻找相应的指令，并将这条指令运送到了 IR 中。(Fetch the next instruction)

然后，指令从 IR 被送到了 Decoder 中，对指令进行解码，发现这条指令中

的操作数并不在指令中，而是需要到内存中重新读取。(Decode the instruction)。

根据上一步的结果，W 的值需要到内存中再读取，并且同样通过数据选择器进入到 ALU 中。 (Get data if needed)

最后，在 ALU 中完成了算术运算后结果被保存在了 ACC 中。(Execute the instruction)

(5).“LOD #3” 与 “ADD W” 指令的执行在 Fetch-Execute 周期级别，有什么不同：

这两条指令的执行在获取数据的一步有明显的不同:

因为“LOD #3” 的指令经过译码后，计算机知道这条指令的操作数就在这条指令中，而不需要再到内存中读数.

但“ADD W”的指令就不同了，计算机只能在这条指令中读到“W”的地址码，并不能得到其具体数值，因而还多了一步重新回内存中读取数据的过程。

B .

（1）00010100 00000111

第一个字节中的第四位数代表了这个指令的寻址模式，其中“1”表示操作数是数值，即第二个字节表示了一个二进制数 7。如果是‘0’则表示操作数是地址所指的值，第一个字节中的后四位代表了操作码，其中 0100 表示这是一个 LOD 操作。

（2）.解释 RAM 的地址：

RAM 中的所有存储的数据和指令都有自己的地址，使计算机可以按照地址来读取指令和数据。

（3）.该机器 CPU 是几位的：

CPU 的位数是指 CPU 能一次同时寄存和处理二进制数码的位数，这和 CPU 中寄存器的位数对应，累加器中的位数是 8，这是一个 8 位的 CPU。

（4）.写出该程序对应的 C 语言表达：

```
int main()
{int y，x=3;
y=x+3;
return 0;}
```

任务 2：

A.

（1）

这是一个通过循环来计算从 1 累加到 n 的程序，并将最终的求和结果输出。

（2）

```
Int main(){
Int sum=0,i=1;
Int w=n;
for( i=1;i<=w;i++){
 sum+=i;}
return 0;}
```

B.

(1)

```
Int main(){
Int sum=0,i=1;
Int w=10;
for( i=1;i<=10;i++){
  sum+=i;}
return 0;}
```

(2)机器语言：

```
LOD #1      0
STO X       2
LOD #0      4
STO Y       6
JMP 16      8
LOD X      10
ADD #1     12
STO X      14
SUB W      16
JMZ 28     18
LOD Y      20
ADD X      22
STO Y      24
```

JMP 10　26

HLT　　28

（3）

机器语言是一种直接面向机器的语言，也就是计算机可以明白的语言，通过各种电路等硬件设备而实现它的功能，但是很复杂不便人们记忆和掌握。虽然可以直接对电路进行操作，但是想要用这种语言来编写复杂的程序简直就是要杀了程序员。

高级语言是建立在机器语言的基础之上的，通过编译等操作再将高级语言转为低级的语言从而使计算机可以执行。这种语言给程序员们提供了很大的便利，他们可以用简单的语句来实现较为复杂的程序。

3. 实验小结

通过使用 CPU 模拟器，我更加能理解冯·诺伊曼计算机的结构，一个指令是怎样执行的，需要经过什么步骤，硬件们做什么，而且会用汇编语言编一些简单的程序，而不是像以前一样一脸懵。感叹到方便易懂的高级语言的发明是有多么伟大，感谢前人的铺垫。