

机器指令编程 实验报告

学号

18342022

姓名

郭凯杰

实验名称

利用 Pippin CPUSim 写程序

目录

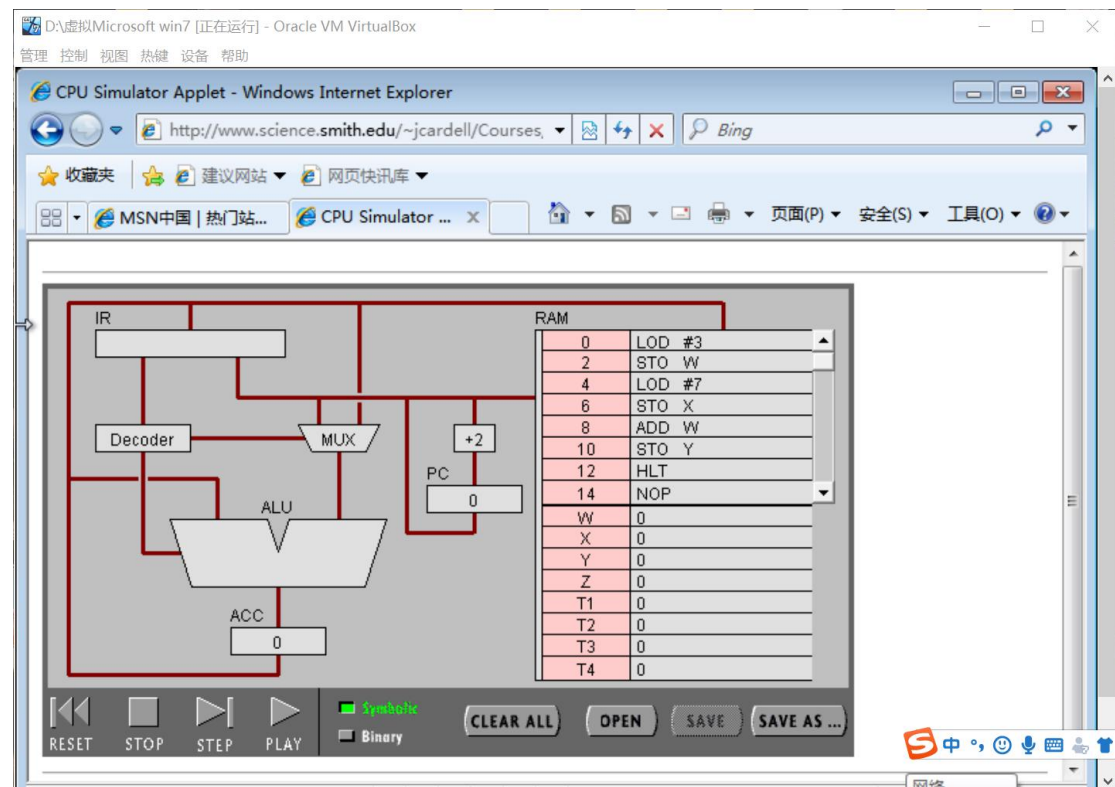
| | |
|---|---|
| 一. 实验目标: | 1 |
| 二. 实验步骤与结果: | 1 |
| 1. 任务一: Add 2 number..... | 1 |
| (1)简要解析以下问题: | 1 |
| 1) PC, IR 寄存器的作用:..... | 1 |
| 2) ACC 寄存器的全称与作用:..... | 2 |
| 3) 用“LOD #3”指令的执行过程, 解释 Fetch-Execute 周期:..... | 2 |
| 4) 用“ADD W”指令的执行过程, 解释 Fetch-Execute 周期:..... | 2 |
| 5) “LOD #3”与“ADD W”指令的执行在 Fetch-Execute 周期级别的不同: | 2 |
| (2) 点击“Binary”后简要解析以下问题: | 2 |
| 1)写出指令“LOD #7”的二进制形式, 按指令结构, 解释每部分的含义: . | 2 |
| 2)解释 RAM 的地址: | 3 |
| 3)该机器 CPU 是几位的? (按累加器的位数) | 3 |
| 4)写出该程序对应的 C 语言表达: | 3 |
| 2. 任务二: Simple loop..... | 4 |
| (1) 简要解析以下问题: | 4 |
| 1)用一句话总结程序的功能: | 4 |
| 2) 写出对应的 c 语言程序: | 4 |
| (2) 修改程序, 实现 1+2+...+10, 结果存到 Y: | 5 |
| 1)写出 c 语言的计算过程: | 5 |
| 2)写出机器语言的计算过程: | 5 |
| 三. 实验小结: | 7 |
| 用自己的语言, 简单总结高级语言与机器语言的区别与联系: | 7 |

一. 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

二. 实验步骤与结果：

1. 任务一：Add 2 number



(1)简要解析以下问题：

1) PC，IR 寄存器的作用：

IR 指令寄存器，用来保存当前正在执行的一条指令。

PC 程序计数器，存放下一条要执行指令在内存中的地址。

2) ACC 寄存器的全称与作用:

全称是 The accumulator(A register), 用于存放操作的数据和结果, 是一种特殊的存储寄存器。

3) 用“LOD #3”指令的执行过程, 解释 Fetch-Execute 周期:

从 RAM 获取指令及数据 (LOD #3) → IR 寄存器 → Decoder (Decode instruction (LOD #3)) → 数字 3 存在 MUX (数据选择器) 中 → Decoder 将 LOD 指令传给 ALU → ALU 从 MUX 得到数字 3 → 累加到 ACC 上 → PC 累加 2, 用于存放下一条指令的地址

4) 用“ADD W”指令的执行过程, 解释 Fetch-Execute 周期:

从 RAM 获取指令及数据 (ADD W) → IR 寄存器 → Decoder (Decode instruction (ADD W)) → MUX (数据选择器) 从 RAM 中获取 W 对应地址的内容, 即数值 3 → Decoder 将 ADD 指令传给 ALU → ALU 从 MUX 得到数字 3 → 累加到 ACC 上 → PC 累加 2, 用于存放下一条指令的地址

5) “LOD #3”与“ADD W”指令的执行在 Fetch-Execute 周期级别的不同:

“LOD #3”在经过 IR 寄存器时, 直接将数据 3 传给 MUX, 以供 ALU 直接提取, 赋给 ACC; 而“ADD W”在经过 IR 寄存器时, 将 ADD 操作信号传给 ALU, 而 MUX 还需在访问 RAM 获取 W 的内容, 即数值, 再供 ALU 提取, 将 W 的值累加到 ACC 上。

(2) 点击“Binary”后简要解析以下问题:

1) 写出指令“LOD #7”的二进制形式, 按指令结构, 解释每部分的含义:

(LOD #7) → 00010100 00000111

其中第一个字节为指令说明符中的操作码 (Opcode), 对应 LOD 操作, 即 Load the operand into the A register (ACC), 第二个字节为操作数说明符, 在这里是数值 7 的二进制码, 整个 Instruction 是将数值 7 载入 ACC 累加器中。

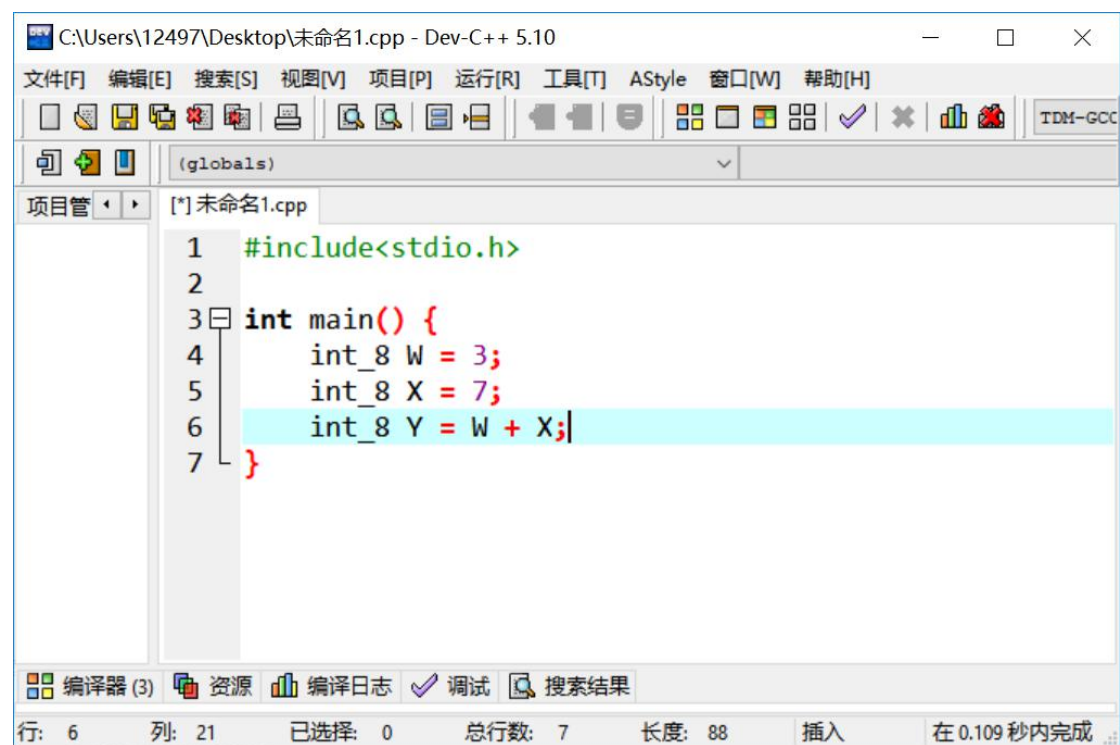
2)解释 RAM 的地址:

RAM 的地址被分为两部分，其中第一部分地址用来储存指令，包括指令说明符以及操作数说明符；第二部分地址用来标识变量的地址，具体存储变量的值。

3)该机器 CPU 是几位的？（按累加器的位数）

8 位

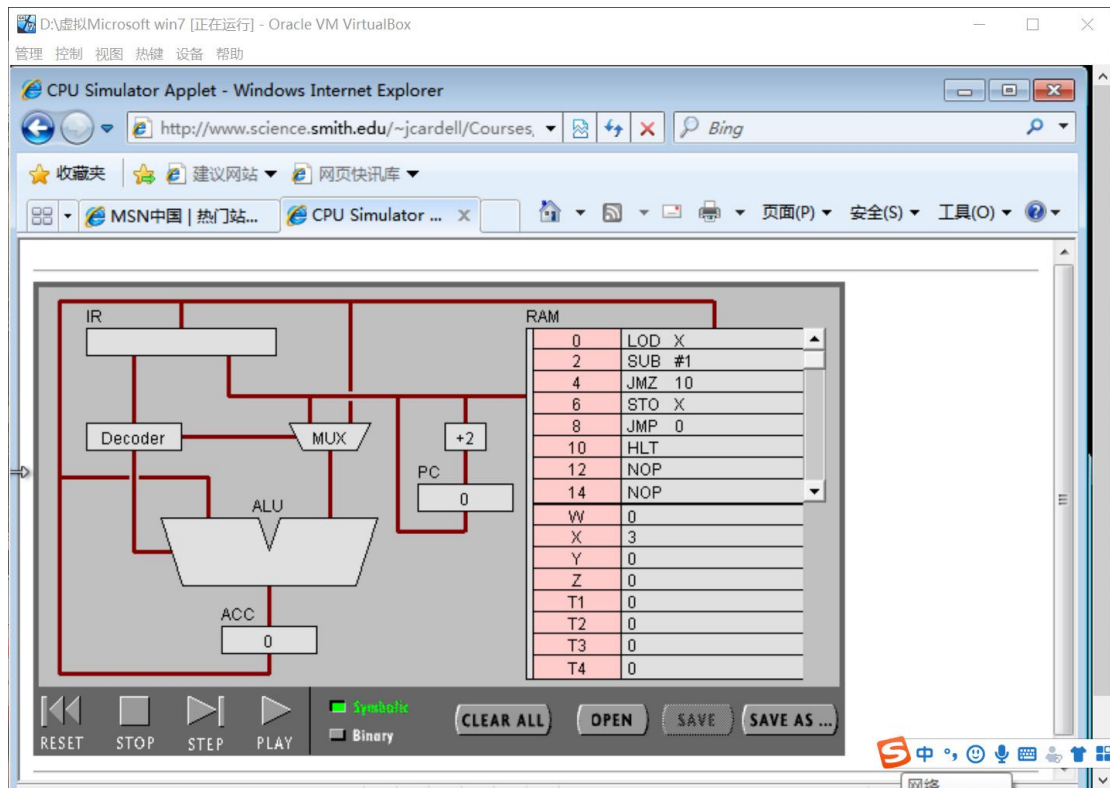
5) 写出该程序对应的 C 语言表达:



```
1  #include<stdio.h>
2
3  int main() {
4      int_8 W = 3;
5      int_8 X = 7;
6      int_8 Y = W + X;
7  }
```

行: 6 列: 21 已选择: 0 总行数: 7 长度: 88 插入 在 0.109 秒内完成

2. 任务二：Simple loop



(1)简要解析以下问题：

1)用一句话总结程序的功能：

给 X 一非零初值，循环减去 1，直到 x 为 1 时 ACC 寄存器存着 0，此时结束程序，否则持续减 1。

2) 写出对应的 c 语言程序：

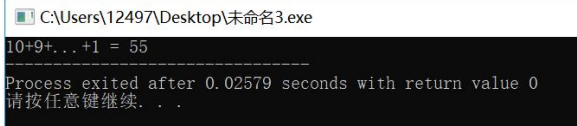
```
1  #include<stdio.h>
2
3  int main() {
4      int_8 x = 3;
5      x = x - 1;
6      while(x > 0) {
7          int_8 acc = x;
8          x = x - 1;
9      }
10 }
```

(2)修改程序，实现 $1+2+\dots+10$ ，结果存到 Y：

1)写出 c 语言的计算过程：

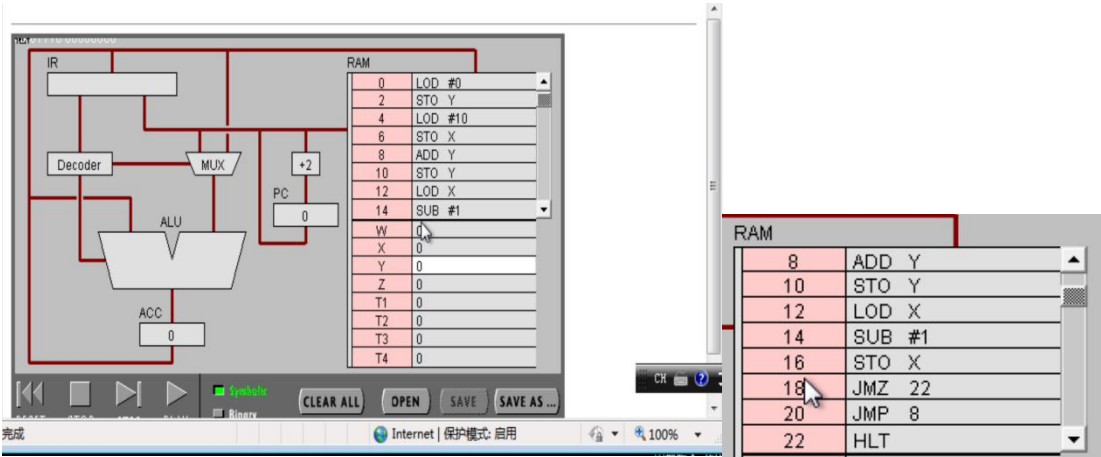
用 C 语言是创建两个整型变量 Y 和 i，分别赋初值 0 和 10，再用一个 for 循环，将 Y 当前值加上 i 再赋值给 Y，i 自减，直到 i 为 0 时跳出循环，此时 Y 的值就是 $10+\dots+2+1$ 的值。如图所示：

```
1  #include<stdio.h>
2
3  int main() {
4      int Y = 0,i;
5      for(i = 10; i >= 0; i --) {
6          Y = Y + i;
7      }
8      printf("10+9+...+1 = %d",Y);
9      return 0;
10 }
```



2)写出机器语言的计算过程：

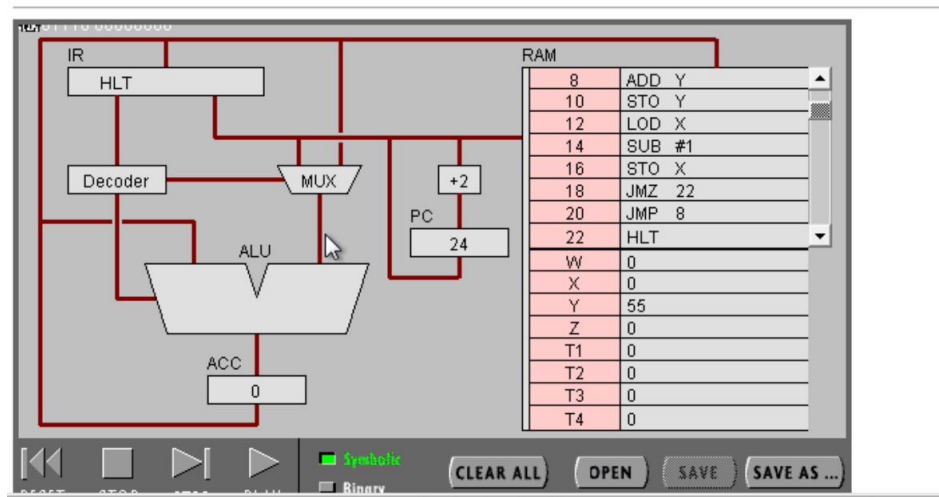
用机器语言是创建变量 Y 和 X，分别赋初值 0 和 10，再用 Loop，每次给 ACC 赋上 X 的当前值，ADD Y 然后将该值赋给 Y，X 每次减去 1，每次均设条件跳转 JMZ 22 语句判断 ACC 当前值是否为 0，若为零，则跳出循环，此时 Y 即为结果，若不为零，则经过无条件跳转语句 JMP 8 回到 ADD Y 那一步，直至满足跳出循环的条件。如图所示：



其中机器语言为：

| RAM | | |
|----------|----------|----------|
| 00000000 | 00010100 | 00000000 |
| 00000010 | 00000101 | 10000010 |
| 00000100 | 00010100 | 00001010 |
| 00000110 | 00000101 | 10000001 |
| 00001000 | 00000000 | 10000010 |
| 00001010 | 00000101 | 10000001 |
| 00001100 | 00000100 | 10000001 |
| 00001110 | 00010001 | 00000001 |
| | | |
| 00010000 | 00000101 | 10000001 |
| 00010010 | 00001101 | 00010110 |
| 00010100 | 00001100 | 00001000 |
| 00010110 | 00001111 | 00000000 |

最后运行结果如预料所示(由于用不了 JNZ，只能用 JMZ 与 IMP 搭配)：



三. 实验小结：

用自己的语言，简单总结高级语言与机器语言的区别与联系：

区别：高级语言简洁易懂，贴近生活中的表达，使人能够更容易接受，程序出现错误时容易找到错误，而机器语言晦涩难懂，且一旦出错，找到错误的时间代价是远远大于高级语言的，且 0, 1 这样的表达方式，的确是难以让人有舒适的敲码体验。

联系：即便机器语言难懂，但它却是人类与计算机沟通的最底层语言，是计算机能够直接领会到程序员目的的语言。而后面的高级语言，则是对机器语言的实现的高度抽象，抽象到不必指引计算机如何进行各种运算操作，而是可以直接以人脑思维进行编码，而不必关心机器语言是怎样的，计算机是如何实现的，编译器是这两者的桥梁，高级语言最终都需要通过编译器转换成机器语言，让计算机能够明白程序员到底做了什么。