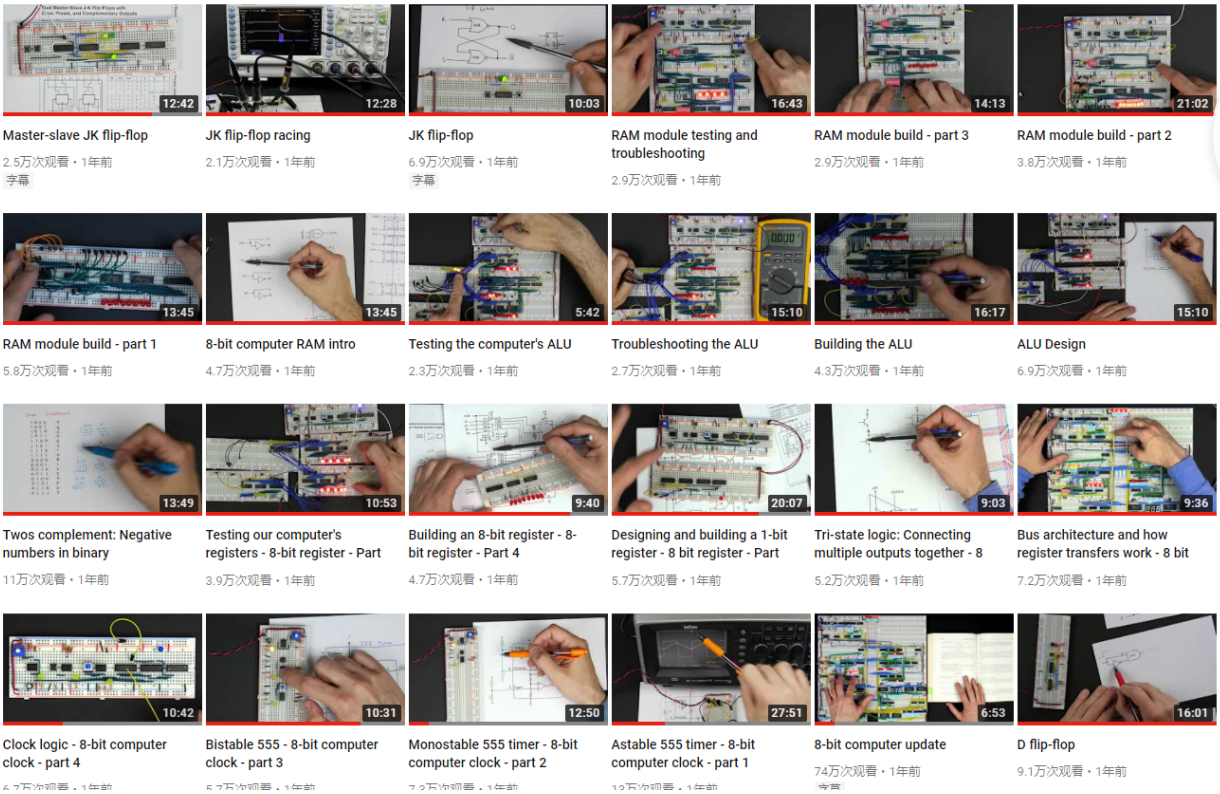
MENU

这个项目是在两天前完成的，想到近来好久没有更新，有愧于诸位，所以就整理了一番，为大家展示一下我搭建的 8 位计算机。

关于 Ben Eater

Ben Eater 是 youtube 上的一个 up 主，曾经发布过一系列视频来展示如何利用面包板来搭建一个 8 位的计算机，并利用之完成一些简单的计算。详见如下截图和本段起始的超链接。



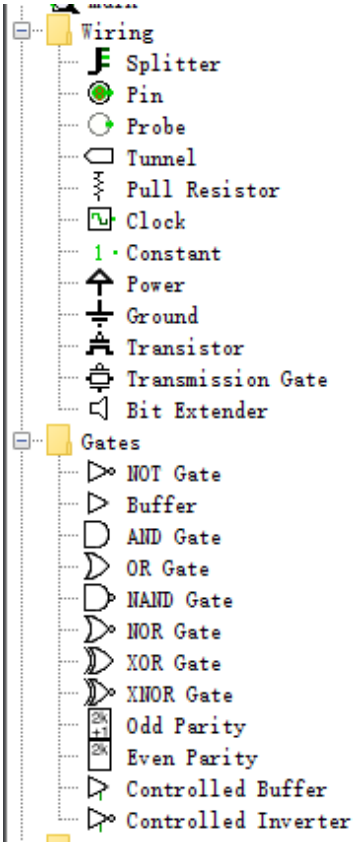
搭建 8 位计算机

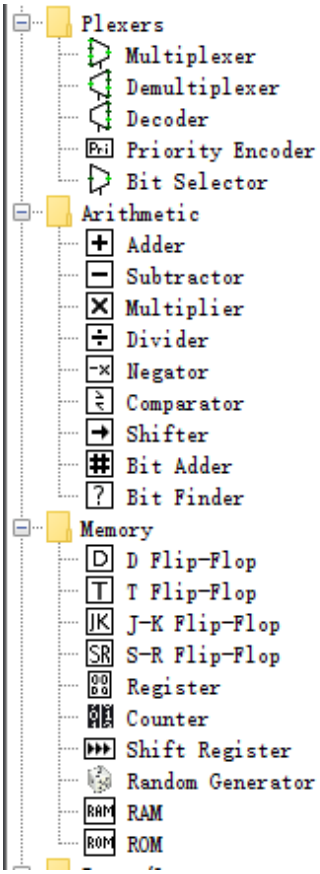
因为芯片难找，Ben 使用的 74 系列芯片有些是在 198X 年生产的，比较难找，再加上我懒，所以我决定使用软件模拟。而 AD、Multilism、Proteus 这些软件都因为芯片库不全而无法模拟。然而一个只有不到 7MB（windows 下）、由 Java 写成的开源软件就足以胜任了。

Logisim

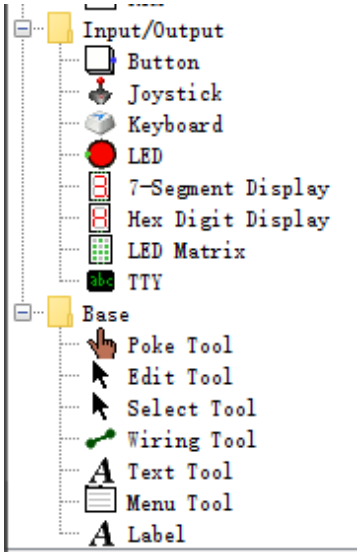
MENU

Logisim 是使用 Java 写成的，其跨平台性很好，目前最后更新是 2014 年 10 月 11 日，最新版 2.7.0。不过它只能模拟最基本的门电路，以及在其上构造的 D 锁存器、触发器等，详见下图：



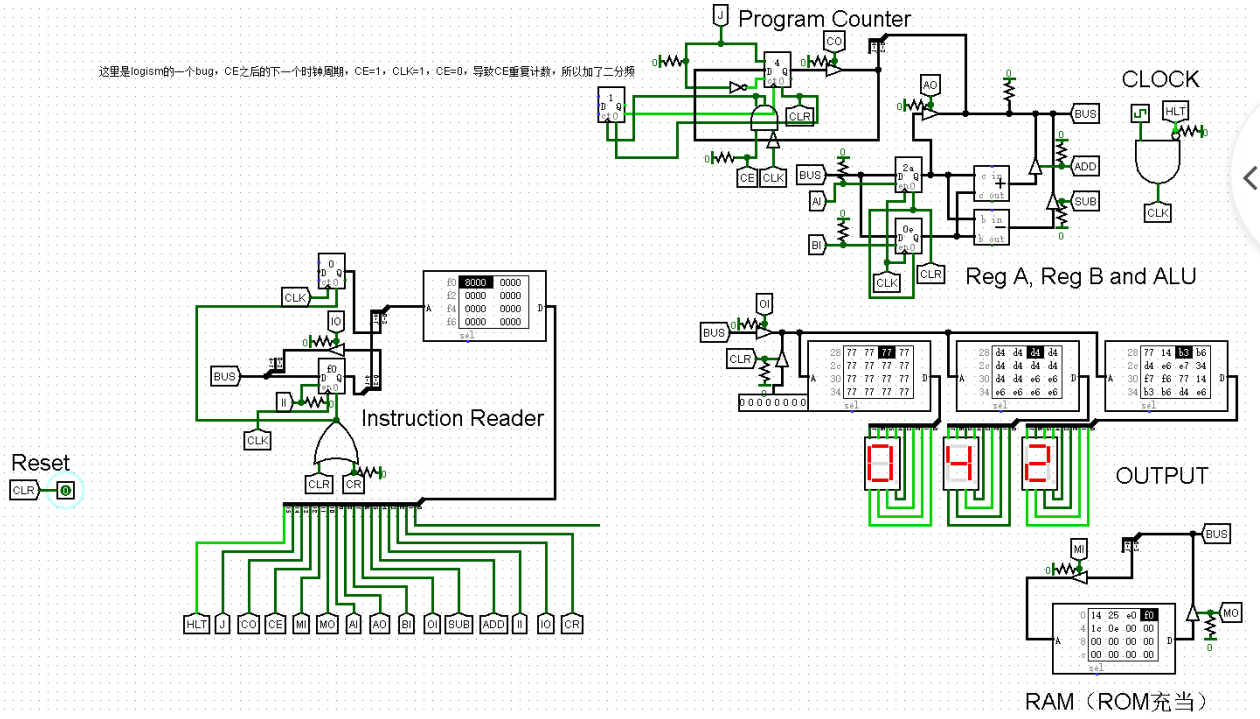


MENU



搭建

如下是我搭建的 8 位计算机，其中 Tunnel 功能类似于 Multilism 这些仿真软件中的网络标签，极其好用，也令我的搭建方便、简洁了不少。



我在每个部件旁边都加了标签，而跳线基本上是按照 Ben Eater 的来的。对于没有看过 Ben Eater 视频的，我将对部件进行详述。

总线 / BUS

顾名思义，总线是各个组件交换信息的媒介，根据逻辑控制部分的统一调度，同一时刻只能有一个组件往总线上写数据，否则多个数据将被混为一谈，读取的组件无法分辨。当然，你可以使用多了总线，将数据总线和地址总线分开。根据 Ben 的设计，总线的低四位是地址总线（XXXXYYYY 中的 Y），而全部 8 位

又作为数据总线使用，因此需要额外的指令周期进行调度。总线不用时应当被下拉电阻拉至逻辑 0 的电平。在寄存器 A、B 和 ALU 处可见我已经将总线拉至 0.

程序计数器 / Program Counter

正如其名字，他是负责记录程序运行到哪里的。程序需要依靠正确的顺序来执行，而每一条指令按照写入的位置拥有其指定的编号（地址）。Program Counter 可以受逻辑控制部分按需将地址放入到总线的低四位以供其他组件读取。同时也可以接受逻辑控制单元的 J 信号，将总线上的地址读入，来完成 JUP 操作。

寄存器 / Register

寄存器用来储存数据的。就目前的设计而言，寄存器 A 可供写入读出，而寄存器 B 用于指令后面数据的寄存，只可写入。

ALU

ALU 全名 arithmetic and logic unit，中文名算术逻辑计算单元，用来计算的。Logisim 中提供了加减乘除和位移，而我只用了加和减。我让它能够从寄存器 A 和 B 读取数据，计算后将结果写入总线。

时钟 / Clock

负责提供时钟信号。是所有组件行动的标准，各个组件按照时钟信号的上升沿和下降沿行动，逻辑控制单元也根据时钟信号执行对应的指令。同时时钟也接受逻辑控制单元的 HLT 信号来停止时钟输出，让电脑冻住。

输出 / OutPut

负责将总线上的数据转化成十进制数字显示出来。使用了三块 EEPROM 芯片，预先对其编程，将总线上的数据转化成对应的地址直接输入给 ROM，令其读取对应地址上的数据，这里我们根据其所在位数和数码管连线预先设计好该如何显示，这样就充当了译码器的工作。

```
public static void dex() throws IOException {
    String[] hex = new String[]{"77", "14", "b3", "b6", "d4", "e6", "e7", "34", "f7", "f6"};
    FileWriter fw1 = new FileWriter(".//1");
    fw1.write("v2.0 raw\n");
    for(int x = 0; x <= 0xff; x++){
        fw1.write(hex[x/100] + " ");
    }
    fw1.close();

    FileWriter fw2 = new FileWriter(".//2");
    fw2.write("v2.0 raw\n");
    for(int x = 0; x <= 0xff; x++){
        fw2.write(hex[x/10%10] + " ");
    }
    fw2.close();

    FileWriter fw3 = new FileWriter(".//3");
    fw3.write("v2.0 raw\n");
    for(int x = 0; x <= 0xff; x++){
        fw3.write(hex[x%10] + " ");
    }
    fw3.close();
}
```

这里使用了 Java，其他编程语言都行，由于 Java 不支持二进制，所以我转化成了 16 进制并按照 Logisim 支持的格式输出成文件，以便直接导入 Logisim 作为 ROM 的数据。

内存 / RAM

我使用了 ROM 作为内存，因此只读。原因是 Logisim 在每次重置模拟后都会清空寄存器和内存里的数据，因此我试用了 ROM 来储存程序，以免去每次都要重新载入程序的麻烦。内存组件会接受逻辑控制单元的控制，从总线上读取地址，然后并且按需将内部储存的程序机械码或数据写入总线，供指令解释器或寄存器读取。

指令寄存器 / Instruction Reader

指令寄存器读取高四位总线，与指令周期计数器一起组成地址交给 ROM，ROM 里面根据不同的指令和不同的指令周期，按需让组件写入、读取总线以完成指令。这里你可以自定义每个机械码代表的指令，可以自定义每个指令都做什么。

```
public static final int HLT = 0x8000;
public static final int J   = 0x4000;
public static final int CO  = 0x2000;
public static final int CE  = 0x1000;
public static final int MI  = 0x0800;
public static final int MO  = 0x0400;
public static final int AI  = 0x0200;
public static final int AO  = 0x0100;
public static final int BI  = 0x0080;
public static final int OI  = 0x0040;
public static final int SUB = 0x0020;
public static final int ADD = 0x0010;
public static final int II  = 0x0008;
public static final int IO  = 0x0004;
public static final int CR  = 0x0002;

public static int[][] ops = new int[][]{
    {CO|MI, MO|II|CE, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, CR},//0x00 NOP
    {CO|MI, MO|II|CE, IO|MI, MO|AI, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x01 LDA*
    {CO|MI, MO|II|CE, IO|MI, MO|BI, ADD|AI, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x02 ADD*
    {CO|MI, MO|II|CE, IO|MI, MO|BI, SUB|AI, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x03 SUB*
    {CO|MI, MO|II|CE, AO|BI, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x04 ATB
    {CO|MI, MO|II|CE, HLT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x05
    {CO|MI, MO|II|CE, HLT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x06
    {CO|MI, MO|II|CE, HLT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x07
    {CO|MI, MO|II|CE, HLT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x08
    {CO|MI, MO|II|CE, HLT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x09
    {CO|MI, MO|II|CE, HLT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x0A
    {CO|MI, MO|II|CE, IO|J, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x0B Jump*
    {CO|MI, MO|II|CE, IO|MI, MO|OI, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x0C OTM* (output Memory)
    {CO|MI, MO|II|CE, OI, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x0D OTB (output Bus)
    {CO|MI, MO|II|CE, AO|OI, CR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},//0x0E OTA(output A)
    {HLT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0} //0x0F HLT
};
```

这里我定义了全部的 16 个机械码。其中 1、2、3、E、F 经过测试，其他的尚未经过测试。以 LDA 为例，首先是让程序计数器输出，内存输入，让内存定位到程序所在的地址；之后内存将数据写入总线，指令寄存器读入，程序计数器 + 1。这是每个指令都必须要做的，因为必须将机械码读入指令寄存器才能针对不同的指令做出不同的操作。随后指令寄存器可以将低四位输出到总线上，令内存读入以定位到要写入寄存器 A 的数据所在的地址；接着就是要让内存输出，寄存器 A 读入。最后一步可选，为了提高运行速度，我将跳过为用到的指令周期，直接结束当前的指令，让计算机继续执行下一个指令。

ROM 编程代码如下：

```
public static void main(String[] args) throws IOException {
    FileWriter fw = new FileWriter(".//in");
    fw.write("v2.0 raw\n");
    for(int x = 0; x <= 0xf; x++) {
        for(int y = 0; y < ops[x].length; y++){
            if(ops[x][y] - 0xFFF > 0)
                fw.write(Integer.toHexString(ops[x][y]) + " ");
            else if(ops[x][y] - 0xFF > 0)
                fw.write("0" + Integer.toHexString(ops[x][y]) + " ");
            else if (ops[x][y] - 0xF > 0)
                fw.write("00" + Integer.toHexString(ops[x][y]) + " ");
            else
                fw.write("000" + Integer.toHexString(ops[x][y]) + " ");
        }
    }
    fw.close();
}
```

给电脑编程

我们设计一个简单的程序，就是经典的 28+14。首先我们简单的设计一下要干啥：

程序地址	指令	地址
00	LDA	04
01	ADD	05
02	OTA	00
03	HLT	00
04	数据 28 的高四位	数据 28 的低四位
05	数据 14 的高四位	数据 14 的低四位

将其翻译成对应的机械码，就是

程序地址	机械码
00	14
01	25
02	E0
03	F0
04	1C
05	0E

写入 RAM，重置计算机，开始运行模拟即可。

总结

我搭建这个计算机，总共花费了 3 个小时。全程也没有查什么资料，看过 Ben 的视频之后就凭着感觉搭建调试了。欢迎各位也尝试动手做一做，然后留个评论什么的。

相关文件下载

这里 Logisim 就不提供了，只提供 Logisim 的工程文件，还有文中提到的 Java 代码。

谷歌云端硬盘

8 位计算机 Ben Eater Logisim

最后编辑于: 2020 年 03 月 10 日

返回文章列表 文章二维码 打赏