计算机组成原理大实验

16位MIPS CPU实验报告

计43 丁铭 2014011333

计43 沈天成 2014010646

计43 邹昊 2013011016

1. 实验目标

在本次实验中，我们在THINPAD教学计算机的硬件平台上，实现了能够执行THCO-MIPS指令集（MIPS16e的改编实现）的CPU。我们实现的指令共有30条，其中包括25条基础指令以及5条扩展指令。

CPU支持指令流水，能够以25MHz的主频运行；能够正确处理了流水线中可能有的各种结构冲突、数据冲突、控制冲突；能够运行监控程序，执行监控程序支持的各项操作。

此外我们实现了一个手写数字识别的扩展功能。我们接入了一个OV7670摄像头，能够拍摄一张手写数字的照片，CPU可以读取照片中的相关信息，执行识别程序进行计算。

2. 实现指令集

我们实现的25条基础指令是

ADDIU ADDIU3 ADDSP

ADDU AND B

BEQZ BNEZ BTEQZ

CMP JR LI

LW LW\_SP MFIH

MFPC MTIH MTSP

NOP OR SLL

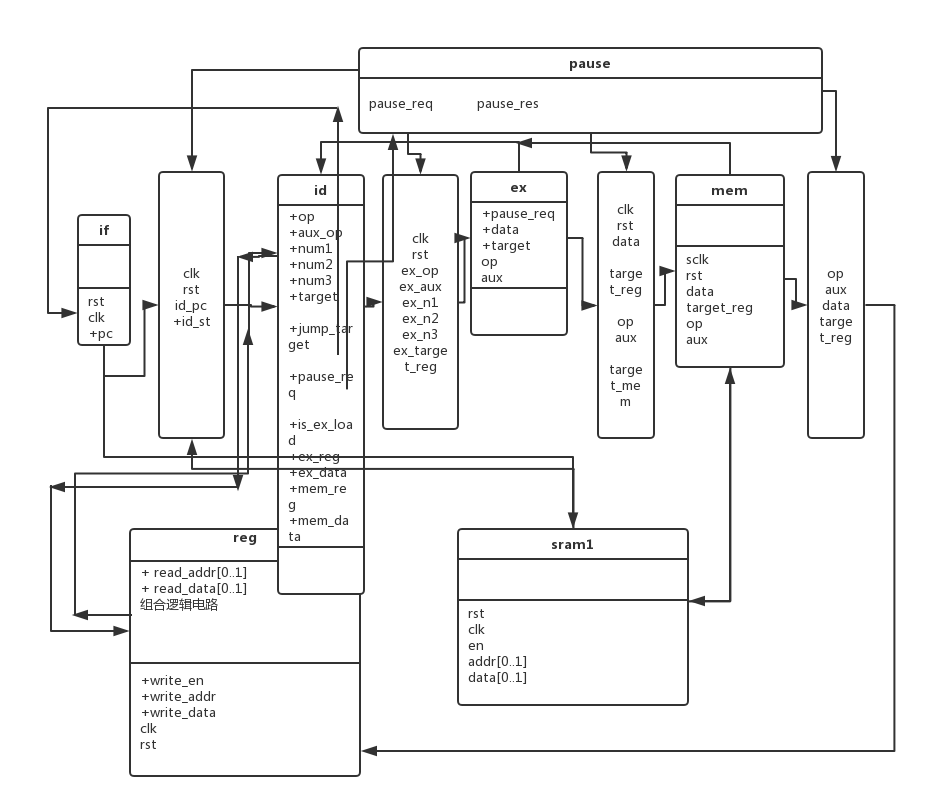
SRA SUBU SW

SW\_SP

另外5条扩展指令是

JRRA JALR ADDSP3 MOVE NEG

3. 数据通路设计



数据通路采用流水线的方法设计，其中在实现时，if模块与if/id段间寄存器实现在同一个模块中，wb模块略去。

4. 流水线设计

4.1 设计思想

流水线架构的设计思想是将每条指令的执行分为5步，每个周期执行一步。在设计中，各段间寄存器为时序逻辑电路，当时钟跳变到来时将前一个阶段的结果赋给下一个阶段的输入。各段均为时序逻辑，实际上，由于访问存储的要求，sram模块存在更高频率的时序逻辑。

顶层模块输入输出信号如下：

|  |  |  |
| --- | --- | --- |
| 名称 | 输入或输出 | 功能 |
| clk\_raw | 输入 | 时钟 |
| rst | 输入 | 重置 |
| output | 输出 | 连接led灯用于显示调试信息 |
| ram\_addr | 输出 | 访存地址 |
| ram\_data | 输入输出 | 读写的数据 |
| ram\_en、ram\_oe、ram\_we | 输出 | sram读写使能 |
| seri\_rdn | 输出 | 数据总线读使能 |
| seri\_wrn | 输出 | 数据总线写使能 |
| seri\_dataready | 输入 | 数据准备信号 |
| seri\_tbre | 输入 | 发送缓冲寄存器发送完毕 |
| seri\_tsre | 输入 | 发送移位寄存器发送完毕 |

4.2 冲突的处理

对于数据冲突，我们采取了数据旁路的方法，将后面几个周期算出的寄存器结果送到寄存器堆中，当译码阶段需要用到该寄存器将结果给出。一个例外是load相关，即执行阶段的指令是load同时目标寄存器在译码阶段被用到，因为此时load的结果没被计算出，因此需要请求pause模块，将前两个段暂停一周期。

对于结构冲突，我们对于访存暂停一周期就能解决。

对于控制冲突，我们在译码阶段就判断是否要跳转，并且直接将控制信号传给取指阶段。

4.3 模块设计

4.3.1 取指模块

取指模块在时钟跳变到来时更新pc值，同时在该周期中，sram会给出指令的具体数值。

|  |  |  |
| --- | --- | --- |
| 名称 | 输入或输出 | 功能 |
| is\_paused | 输入 | 接收pause模块是否暂停流水线的信号 |
| clk | 输入 | 时钟 |
| rst | 输入 | 重置 |
| id\_pc | 输出 | 段间寄存器给下一阶段的pc |
| id\_instruction | 输出 | 段间寄存器给下一阶段的指令 |
| addr | 输出 | 连接sram模块，访问指令存储的地址 |
| instruction | 输入 | 连接sram模块，取得的指令 |
| jump\_en | 输入 | 译码阶段是否要求pc跳转 |
| jump\_target | 输入 | 译码阶段要求跳转的pc值 |

4.3.2 译码模块

译码阶段是最复杂的阶段之一，需要同时与上阶段的段间寄存器、取指阶段、执行阶段、暂停模块、寄存器堆交互。

具体的信号如下：

|  |  |  |
| --- | --- | --- |
| 名称 | 输入或输出 | 功能 |
| read\_addr1 | 输出 | 与寄存器堆的交互接口，要取的寄存器编号1 |
| read\_addr2 | 输出 | 与寄存器堆的交互接口，要取的寄存器编号2 |
| read\_data1 | 输入 | 寄存器堆给出的数值1 |
| read\_data2 | 输入 | 寄存器堆给出的数值2 |
| id\_pc | 输入 | 段间寄存器给的pc |
| id\_op | 输出 | 计算出的指令类型 |
| id\_instruction | 输入 | 段间寄存器给的指令 |
| num1、num2、num3 | 输出 | 最多三个的操作数 |
| target\_reg | 输出 | 目标寄存器 |
| ex\_target\_reg | 输入 | 执行阶段的目标寄存器，用于判定load相关 |
| is\_ex\_load | 输入 | 执行阶段的指令是否为load |
| jump\_target | 输出 | 译码阶段要求跳转的pc值 |
| jump\_en | 输出 | 译码阶段是否要求跳转 |
| pause\_req | 输出 | 是否请求暂停 |

4.3.3 执行模块

执行模块的主要任务是根据不同的命令对操作数进行运算，所以使用一个多路选择器就能完成工作。

|  |  |  |
| --- | --- | --- |
| 名称 | 输入或输出 | 功能 |
| op | 输入 | 操作类型 |
| num1、num2、num3 | 输入 | 最多三个的操作数 |
| target\_reg | 输入 | 目标寄存器 |
| ex\_target\_reg | 输出 | 执行阶段的目标寄存器，传给译码段用于判定load相关 |
| is\_ex\_load | 输出 | 执行阶段的指令是否为load |
| data | 输出 | 计算结果 |
| target\_mem | 输出 | 如果是访问指令的话，目标地址 |

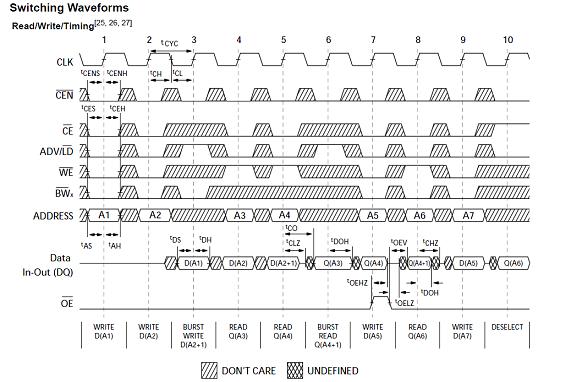
4.3.4访存模块

访存模块对于load、store两类指令计算结果。

|  |  |  |
| --- | --- | --- |
| 名称 | 输入或输出 | 功能 |
| Data | 输入 | 执行阶段的计算结果 |
| op | 输入 | 操作类型 |
| target\_reg | 输入 | 目标寄存器 |
| sram\_toggle | 输出 | 要求sram执行的命令：读、写、无 |
| sram\_addr | 输出 | 访存地址 |
| sram\_data\_in | 输出 | 要写的数据 |
| sram\_data\_out | 输入 | 读到的数据 |
| Data | 输出 | 计算结果 |
| Target\_mem | 输入 | 目标地址 |

4.3.5 Sram模块

Sram模块接受一个50M的时钟，并且在时钟的下降沿工作。这样做的好处是恰好与流水线本身错开信号赋值时间。



sram需要在一个周期内同时完成访存和取址任务，由于可能在同一片sram上，所以为了时序只能在有访存任务的时候暂停一个周期。

对于串口的读写，只需要将BF00映射到串口，并根据data\_ready等信号给出BF01的正确值即可。

4.3.6 Pause模块

Pause模块是一个纯粹的组合逻辑电路，功能十分简单。该模块接受两种请求：

①sram段因为访存在一个周期内做不完的暂停请求，这种情况下应该暂停所有流水线。

②id段因为load相关无法取得正确的操作数的暂停请求，这种情况下应该暂停取指、译码两个阶段，并将执行阶段的输入置为NOP。

4.4 其他设计细节

4.4.1 指令的译码

定义枚举类型operation以及相应的函数get\_op，该函数以指令的相关位为输入，以枚举类型operation为输出。在流水线各阶段间数据传递时，以operation的形式传递操作，增加了代码可读性，也是代码编写更加简单清晰。

4.4.2 立即数的处理

指令中包含的立即数有4位、5位、8位、11位等类型，需进行符号扩展（也有少部分为无符号扩展）后才能继续参与运算。为此，针对不同位数的立即数分别定义符号扩展函数，如sign\_extend4、sign\_extend5等，以便于在流水线中处理指令中的立即数。

4.4.3 寄存器编码

用5个二进制位来表示不同的寄存器，其中第一位表示寄存器编码是否有效，0为有效，1为无效（reset或操作与寄存器无关时即为无效），第二位表示是否为通用寄存器，0为通用寄存器，此时后三位表示通用寄存器的具体编号，1位特殊寄存器，编码为：

SP寄存器：01000

IH寄存器：01001

RA寄存器：01010

T寄存器：01011

基于以上编码，可以通过寄存器模块进行对寄存器的读写。

5. CAM2VGA模块具体设计

5.1 整体设计思路

CAM2VGA模块的整体设计思路是将整个模块分为三个模块：getcam模块，cam2ram模块以及ram2vga模块。

Getcam模块的作用是读取来自摄像头输出的信号，解析出其中的灰度值信号并将灰度值以及对应的像素地址输出。

Cam2ram模块承接getcam模块，将来自getcam模块的像素灰度值信号写入sram2中的对应地址；接受来自ram2vga模块需要的像素地址，从sram2中读出像素灰度值输出；以及接受cpu读取28X28的识别计算矩阵的地址，输出对应位置上的矩阵值。

Ram2vga模块负责从cam2ram中获取像素信号，向vga显示器输出vga信号。

5.2 整体设计图



5.2 内部组件详细设计

5.2.1 getcam模块

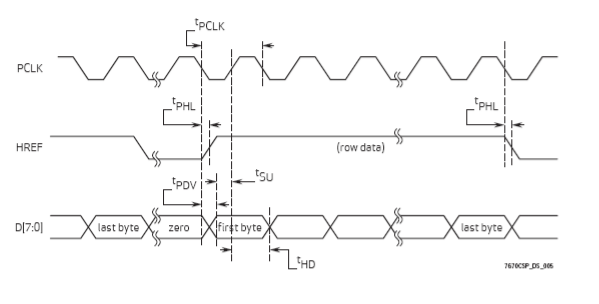
getcam模块的输入输出信号如下：

|  |  |  |
| --- | --- | --- |
| 信号名 | 输入或输出 | 作用描述 |
| fromcamclk | input | 摄像头输出时钟，用于接收摄像头的其他数据时同步 |
| vsync | input | 摄像头场同步信号，当其为1时表示换帧 |
| href | input | 摄像头行同步信号，当其为0时表示换行 |
| d | input | 来自摄像头的数据信号 |
| addr | output | 输出给cam2ram模块的地址 |
| dout | output | 输出给cam2ram模块的数据 |
| ready | output | 表示摄像头是否准备好输出灰度值数据 |
| en | output | 表示输出信号是否有效（像素是否在捕捉框内） |

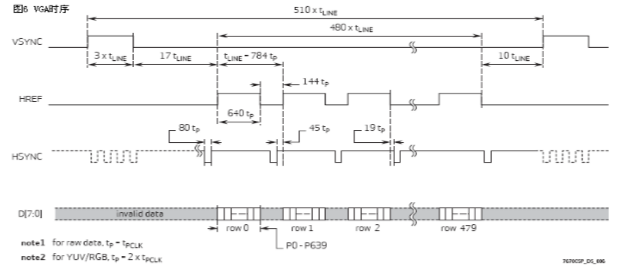
摄像头的型号为OV7670。使用方法如下：两个引脚分别接3.3V电源和地，我们使用了FPGA板上JTAG2的电源和地。向摄像头输入一个25M的时钟，作为它的工作时钟，摄像头的另一个引脚会输出一个同频率的数据时钟，摄像头的数据会伴随这个数据时钟同步给出。

我们在FPGA板的LED灯处焊上了16个引脚，用于摄像头和板子通信接口。我们需要对摄像头输出的信号进行解析。输出时序图如下。

行时序图：



场时序图：



D数据输出的格式是YUV格式，输出顺序是U1&V1、Y1、U2&V2、Y2、U2&V2、Y2...我们需要Y数据作为灰度值，UV数据均可忽略。

因此在vsync=0，href=1时，即一行输出时，每次fromcamclk上升沿时，读取d中数据，同时将ready在0、1之间变化，当ready为1时，表示输出的是灰度值信号，当ready为0时，d收到的是U、V信号在本次实验中不被使用，这个时候更新下次输出的地址。另外还加了一个判断，摄像头拍摄到的画面是640X480像素的，我们只截取了280X280的一块，行号在200到480之间，列号在100到380之间。当输出这一块的信号时，en为1，否则为0。

在vsync=1时，换帧时，将输出的addr变为0。

当vysnc=0，href=0时，将行号vy加1，vx变回0。

5.2.2 cam2ram模块

cam2ram模块的输入输出信号如下：

|  |  |  |
| --- | --- | --- |
| 信号名 | 输入或输出 | 作用描述 |
| addr | input | 来自getcam的地址 |
| requestaddr | input | 来自ram2vga要求像素的地址 |
| d | input | 来自getcam的数据 |
| mnistdata | output | 送给CPU的矩阵元素0/1 |
| mnistaddr | input | CPU要求的矩阵地址，0XFC00之后的地址映射过来 |
| pclk | input | 摄像头输出时钟，作为驱动时钟 |
| rst | input | 初始化信号 |
| switch | input | 拍照信号 |
| pixready | input | Getcam过来的表示是否准备好数据的信号 |
| outr | output | 给ram2vga的要求像素的r值 |
| outg | output | 给ram2vga的要求像素的g值 |
| outb | output | 给ram2vga的要求像素的b值 |
| ram2Oe | output | Sram2的Oe |
| ram2En | output | Sram2的En使能 |
| ram2We | output | Sram2的We使能 |
| ram2Addr | output | Sram2的地址 |
| ram2Data | inout | Sram2的数据 |

cam2ram模块有两个状态：等待拍照状态和拍照保持状态。当rst信号过来时，进入等待拍照状态，outr，outg，outb都输出111，此时vga显示器上显示一片白色。同时对于getcam送过来的addr和d数据，当pixready=1时，ram2Data<=d， ram2Addr <= addr，ram2We<=1。当pixready=0时，此时的d数据是无用的，这个阶段正好用来ram2We<=0，写入数据到sram2。

当switch信号来时，继续维持等待拍照状态直至当前帧写入完毕，此时转入拍照保持状态。拍照保持状态下，sram2进入读状态，ram2Oe设为0，ram2We设为1，ram2Data设为高阻状态。

outr <= ram2Data(7 downto 5);

outg <= ram2Data(7 downto 5);

outb <= ram2Data(7 downto 5);

ram2Addr <= requestaddr;

即可读出sram2中对应像素的灰度值，读取灰度值数据取高3位作为vga的输出。

同时对在等待拍照状态时读取的当前帧的数据进行结块处理，即将280X280的像素矩阵划分成28X28个10X10的矩阵，每个10X10矩阵进行100个像素的灰度值加和处理，若和大于一个阈值则认为对应的28X28矩阵中的这个点是1（1认为是有数字的颜色，0表示背景色），否则认为是0。

另外有一个组合逻辑模块，接收mnistaddr的28X28矩阵的地址，输出相应的0/1元素。

由于sram2无法同时进行读和写操作，所以我们只能做到拍照（写sram）并显示那一帧（读sram），而无法做到实时显示摄像头画面。

5.2.1 ram2vga模块

ram2vga模块是作为给显示器输出信号供显示用的，它的输入输出信号如下：

|  |  |  |
| --- | --- | --- |
| 信号名 | 输入或输出 | 作用描述 |
| rst | input | 初始化 |
| clk | input | Vga的时钟信号，25M |
| vs | output | 场同步信号 |
| hs | output | 行同步信号 |
| requestaddr | output | 传给cam2ram模块，需要的像素地址 |
| getr | input | 接收的cam2ram发来的r值 |
| getg | input | 接收的cam2ram发来的g值 |
| getb | input | 接收的cam2ram发来的b值 |
| r | output | 发送给显示器的r值 |
| g | output | 发送给显示器的g值 |
| b | output | 发送给显示器的b值 |

Ram22vga模块内置了x变量和y变量，表示当前扫描到的像素位置的xy方向坐标。

每当clk上升沿时，x和y变量都相应的变化，表示前进一像素。

X变化：若X=799，则变为x=0，否则x=x+1

Y变化：若X<799，则不变，否则则若y=524，则y=0，否则y=y+1

当x在0-280之间，y在0-280之间时，就要求cam2vga中存放x在200-480之间，y在100-380之间的对应像素。当x=0，y=0时requestaddr就设为0，之后每当位置更新且在输出范围内，则requestaddr增加1。

另外还要考虑行同步信号hs和场同步信号vs。X在656-752之间，则行同步信号hs为0，否则为1，y在490-492之间，则场同步信号vs为0，否则为1。

当x在0-280之间，y在0-280之间时则输出来自cam2ram的rgb信号，否则rgb都输出000，显示黑色。

1. 实现效果

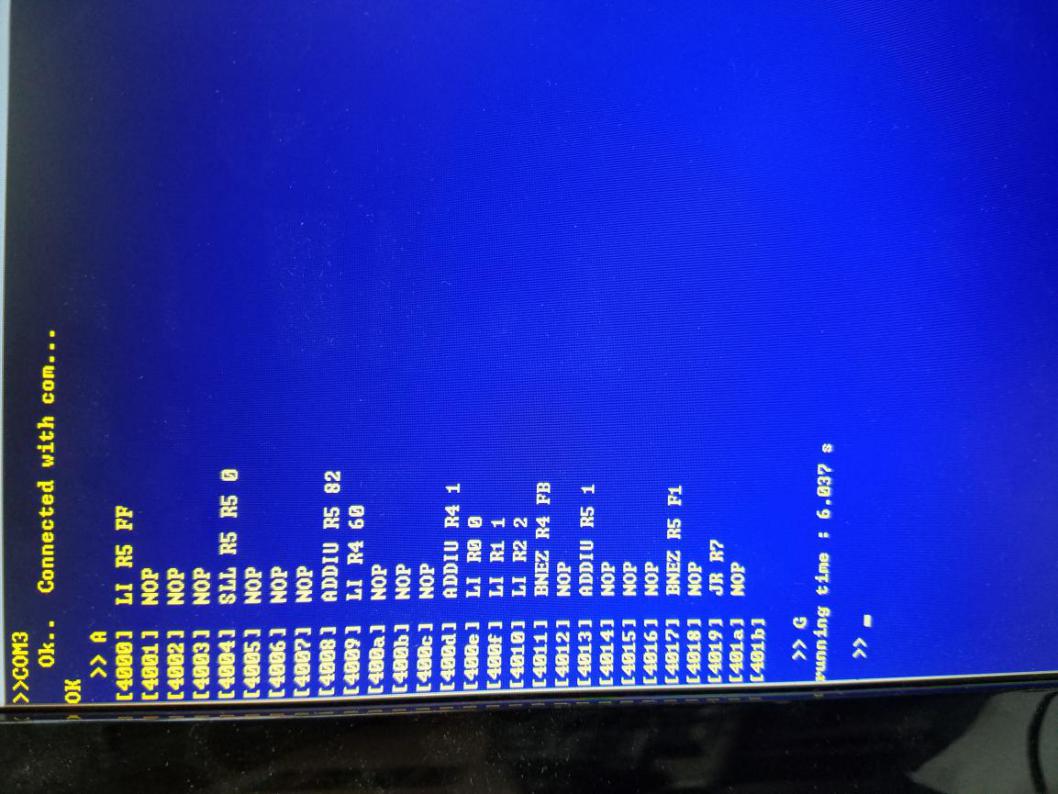
6.1 基本功能效果

我们实现的CPU能够支持在25M的主频下很稳定地运行监控程序。执行A、U、R、G、D5条命令都很正确。

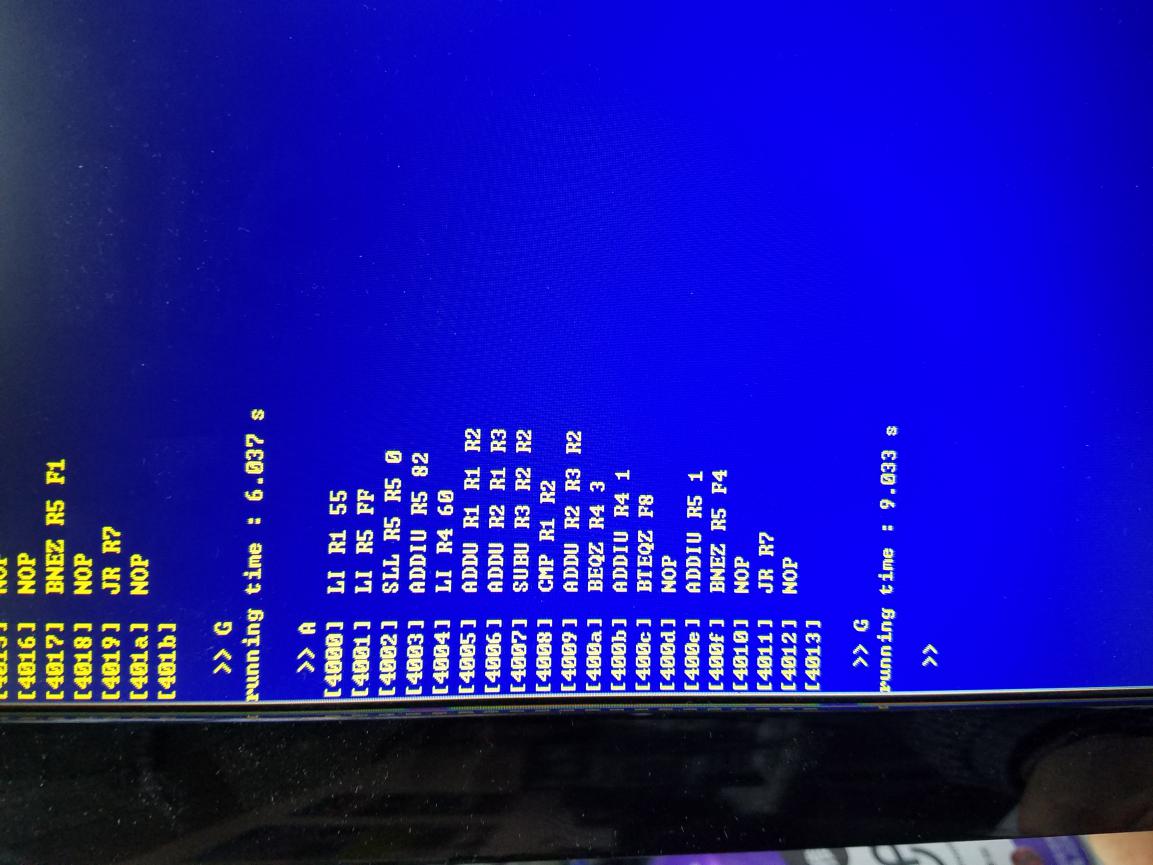
在监控程序下执行5段测试程序，都能顺利运行退出。

以下是我们的CPU执行测试程序的结果。可以看到我们的CPU运行时间比较短，速度还是比较快的。

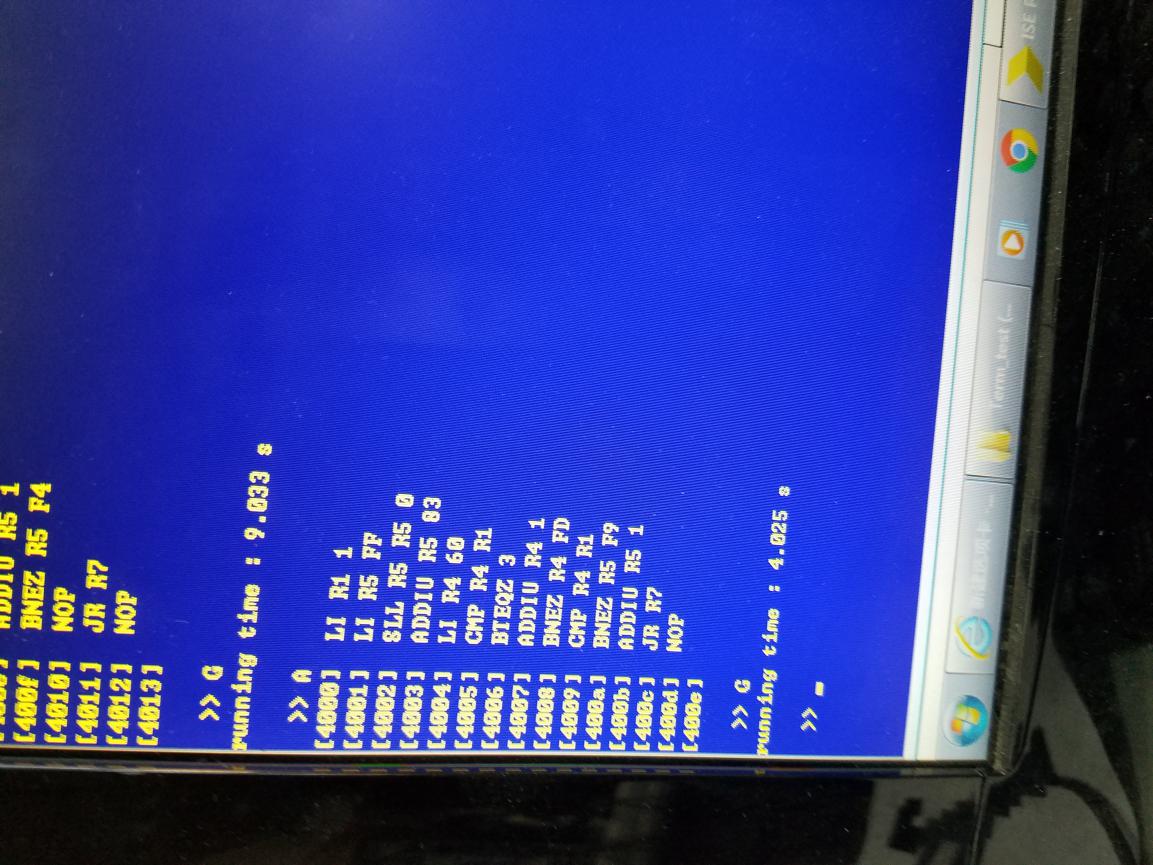
第一段程序：



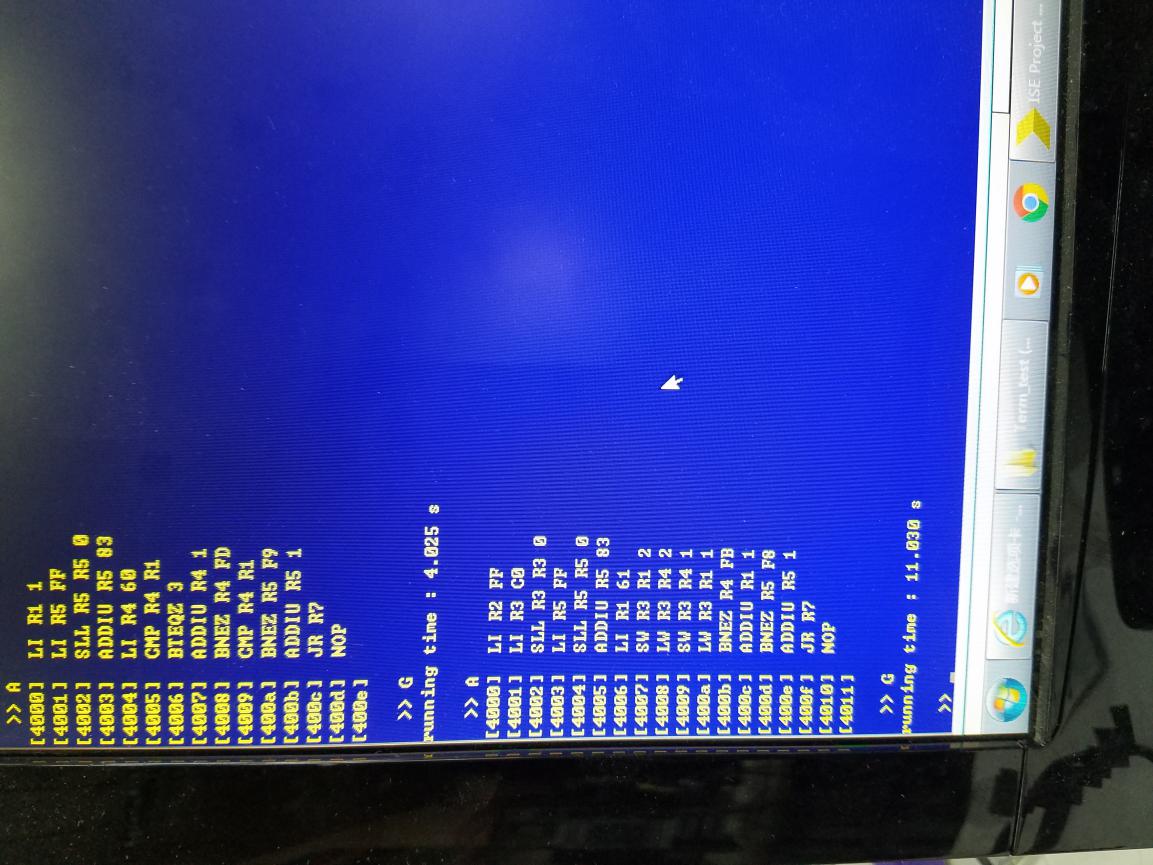
第二段程序：



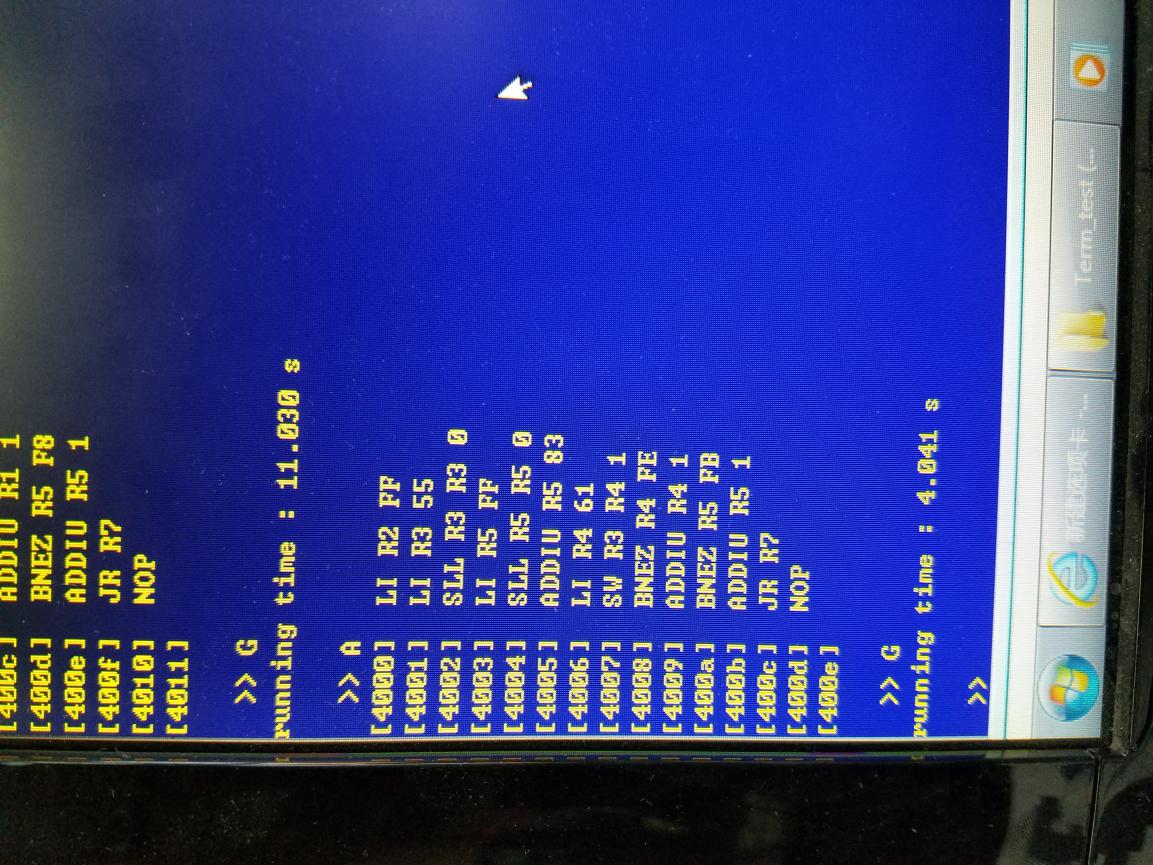
第三段程序：



第四段程序：



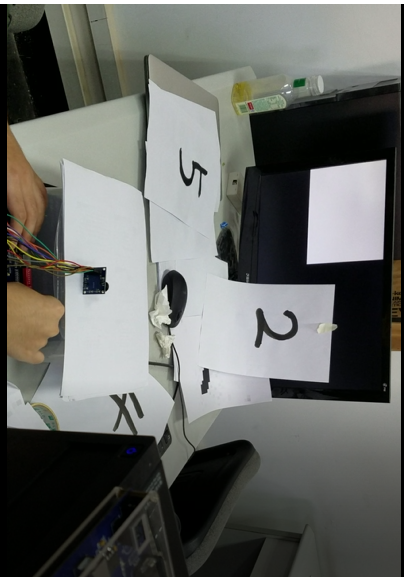
第五段程序：



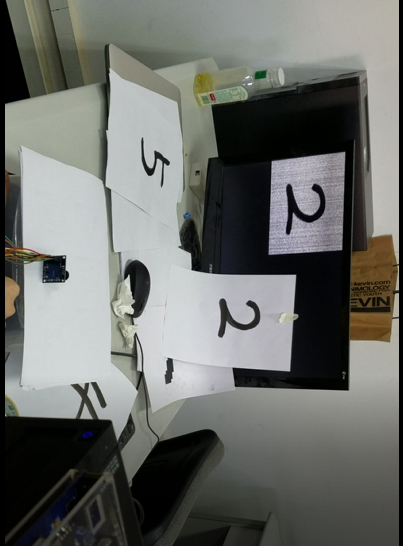
6.1 扩展功能效果

我们实现了一个用摄像头拍照获取手写数字的照片，并进行识别将识别结果输出到终端上的功能。效果如下：

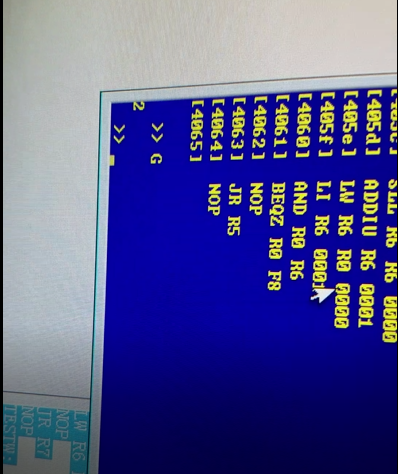
这是等待拍照状态，此时显示器屏幕上一片空白。



按下快门按键之后，屏幕上就显示了我们拍到的数字。



运行识别程序，得到识别结果。



由于硬件实现条件有限，汇编代码编写困难，我们的识别算法是一个经过简化的算法，识别效果并不是100%正确，会受到数字摆放位置，数字字体风格的影响。根据我们的测试，我们的识别正确率大概在50%-60%左右，也已经是一个不错的效果。

1. 实验心得与体会

7.1 良好的设计

我们组在造计算机的三星期中，开工时间相比起其他组来说是属于比较晚的，但我们组最后还是能不用赶进度就很顺利地完成全部实验内容，从没有出现过可能完不成实验的“危机情况”。这一切都归功于我们良好的CPU设计。

我们的CPU从编写完成到能正常运行监控程序的调试工作只花费了一至两天的时间，跟其他组相比属于很短的。调试发现的bug基本都是疏忽大意写错或漏写了几句语句，没有出现过需要对设计和代码进行大刀阔斧地修改的情况。这让我们意识到动手开始实验之前，需要对实验的方法和过程做好充分的思考设计工作，然后再动手，而不是先凭感觉随意写代码，然后再改，这样能减少调试工作的时间，而调试的时间往往是实验时间中最多的一块。这不仅仅是这次实验，其他的工作都是如此。

7.2 勇于尝试

我们此次实验实现的扩展功能是用摄像头拍摄手写数字然后进行识别的功能。这项扩展功能是一项比较有新意的功能，大多数同学都不会想到。外设选择上我们也放弃了很多组都会用到的键盘外设，而选择了摄像头这个外设。摄像头的使用方法没有实验书的指导，也没有学长前辈们的经验参考，需要我们自己查询摄像头的手册来尝试摸索。摄像头的接入方法我们还咨询了李山山老师，特地请他帮我们给实验板子焊上GPIO接口。在实验的开始阶段，我们的想法还充满了未知数，可能会遇到各种意想不到的困难，幸运的是我们最后克服了这些困难，把我们最初设想的功能都实现，虽然效果（准确率）比期望的略差一些，但也令我们很有成就感。这次实验让我们感觉到，勇于尝试新的事物，是一件很有意思的事情。

7.3 总结

“奋战三星期，做台计算机”过去了，这确实是一个很令我们受苦的实验。在这段时间里，我们几乎搁置了其他所有事情，把所有精力都投入其中。我们投入了很多，但也学习到了很多。我们既学习了CPU的工作原理，硬件设计的知识，也磨炼了我们的耐心和韧性。这些都是很宝贵的财富。