

计算机组成原理大实验

# 支持指令流水的计算机系统的 设计与实现

陈可卿      孙艺瀚

计05      计05

2010011347      2010011356

December 7, 2012

## Contents

<b>1 实验目标</b>	<b>4</b>
<b>2 主要模块设计</b>	<b>4</b>
2.1 流水段设计 . . . . .	5
2.1.1 取指(IF) . . . . .	5
2.1.2 译码(ID) . . . . .	5
2.1.3 执行(EXE) . . . . .	6
2.1.4 访存(MEM) . . . . .	6
2.1.5 写回(WB) . . . . .	6
2.2 数据通路 . . . . .	6
2.3 指令流程表 . . . . .	7
<b>3 主要模块实现</b>	<b>9</b>
3.1 PC寄存器单元 . . . . .	10
3.2 branch跳转控制模块 . . . . .	11
3.3 IF/ID寄存器组模块 . . . . .	12
3.4 立即数扩展模块 . . . . .	13
3.5 寄存器划分模块 . . . . .	14
3.6 寄存器组模块 . . . . .	15
3.7 ID/EXE寄存器组模块 . . . . .	16
3.8 算术逻辑单元模块 . . . . .	17
3.9 EXE/MEM寄存器组模块 . . . . .	18
3.10 内存模块 . . . . .	19
3.10.1 UART模块 . . . . .	20
3.10.2 SRAM模块 . . . . .	22
3.10.3 VGA和显存模块 . . . . .	24
3.10.4 键盘模块 . . . . .	24
3.11 ME/WB寄存器组模块 . . . . .	24
3.12 写回模块 . . . . .	25

<b>4 扩展功能介绍</b>	<b>26</b>
4.1 冲突解决 . . . . .	26
4.1.1 数据旁路解决数据冲突 . . . . .	26
4.1.2 流水线暂停解决数据冲突 . . . . .	28
4.1.3 倍频解决结构冲突 . . . . .	28
4.1.4 软件解决控制冲突 . . . . .	29
4.2 中断 . . . . .	29
4.3 VGA和显存 . . . . .	30
4.3.1 VGA显示模块 . . . . .	30
4.3.2 VGA只读存储器模块 . . . . .	32
4.3.3 显存模块 . . . . .	32
4.4 键盘 . . . . .	33
4.5 双机通信 . . . . .	34
4.6 伪码解析 . . . . .	35
4.7 其它贡献 . . . . .	36
4.7.1 管脚自动绑定脚本 . . . . .	36
4.7.2 实验指导书修改 . . . . .	37
<b>5 实验成果展示</b>	<b>37</b>
5.1 运行kernel . . . . .	37
5.2 进入中断 . . . . .	38
5.3 在kernel里使用VGA和键盘响应 . . . . .	38
5.4 即时聊天工具 . . . . .	40
<b>6 实验心得体会</b>	<b>40</b>

## 1 实验目标

计算机硬件系统由中央处理器（CPU）、存储器、输入/输出系统等部件构成。本次实验中，我们将解析计算机硬件系统的基本组成、运行原理和协同工作机制，分析计算机组成对系统性能的影响，并设计一台简单的计算机，建立计算机整机系统的概念。我们将利用THINPAD教学机上的硬件系统，通过VHDL语言，设计一台执行基本MIPS指令的机器，并在上面运行监控器程序。同时，在监控器上运行第一次实验的程序和一些实例程序。利用THINPAD教学机，我们可以实现一个多周期或者流水线的计算机系统。考虑到实际背景下的应用，我们组选择了流水线。我们的主要目标分几个阶段：

1. 在THINPAD实验平台上用VHDL语言实现支持指令流水的CPU，能够支持THCO MIPS指令集中指定的部分指令，并能成功通过串口通信运行给定的kernel代码；
2. 在已有要求的基础上，在硬件层面增加一定的扩展部分。如数据旁路、中断处理、VGA和显存、键盘响应等；
3. 在已有指令集的基础上，配合VGA和键盘，在软件层面增加一定的扩展功能。如汇编器的伪代码解析、双机通信等。

## 2 主要模块设计

在本次实验中，我们将完成一个指令流水的CPU的设计。我们采用经典的五级流水，即每条指令的执行过程包括：取指（IF）、译码（ID）、执行（EXE）、访存（MEM）和写回（WB）五个阶段。每个阶段的控制信号由且仅由上一阶段产生或传递，实现指令流水。各个阶段之间用一个流水线锁存器的寄存器堆模块分隔，在每个时钟上升沿到来时，进行前一模块到后一模块数据的传输。在各个模块之间，我们尽量采取组合逻辑，减少if的嵌套，以with-select的选择器为主，以避免时序过于复杂带来的延迟等问题。

下面，我们将概括介绍我们的设计方案。

## 2.1 流水段设计

在这一部分，我们将对五个流水阶段做详细的阐述。这五个阶段将被四个不同的寄存器组隔开，具体实现中，我们会将指令、PC值和所有的控制信号在流水段上继续传递下去，方便后续的使用。这样每一个阶段都可以得到本条指令的完整信息，方便对通路的修改和添加。我们将指令依次传下去还有一个好处是可以在对应的单元对指令重新进行独立的解析。

下面我将对于这五个阶段需要做的事情进行详细的阐述。

### 2.1.1 取指(IF)

在该阶段，我们将根据PC值得到当前指令，并得到下一步的PC值。PC寄存器的值由一个多路选择器给出。若遇到跳转指令，则当前PC值由后面的阶段给出控制信号进行设置。否则，保留当前PC值，根据当前PC值从内存中获取相应指令，并使PC加一。同时，PC寄存器的值（其实是PC+1）将会继续向后传递，以便实现MFPC等指令。

### 2.1.2 译码(ID)

在这一阶段，我们将对取出的指令进行分析，取出需要用到的寄存器编号、内存地址、立即数等。在整个CPU的结构中，多数控制指令也由该阶段解析指令后产生。包括T寄存器的使能和赋值、是否有立即数操作、是否需要写回、是否是跳转语句，并划分寄存器。

该阶段译码产生的寄存器编号将被送往寄存器模块并在阶段内完成寄存器的访问。同样地，译码产生的立即数将被送往立即数扩展模块，内存地址将会沿着流水线向后传递。而各控制信号将会沿着流水线在相应阶段送往各控制器。

在我们的实现中，由于instruction信号将沿着流水线传递下去，有一部分的译码工作将在各模块内进行。这样简化了数据通路，减轻了译码阶段的负担，同时保证了每一个部件工作时信息的完整性，便于扩展和调试。本阶段的另一个任务是决定下一阶段PC的值，供下一周期使用。由于B型和J型指令的存在，PC指令除了可以来自(PC+1)以外，还可能是寄存器中的值（即寄存器堆给出的第一个操作数A）或者(PC+1+IMM)。这两种情况都可以在本

阶段得到，并传给PC的多路选择器，再通过译码给出控制信号，得到下一阶段PC的值。

### 2.1.3 执行(EXE)

对于译码阶段给出的操作数和操作数进行算术逻辑运算，这一过程主要是ALU的运算。ALU的两个操作数都将通过多路选择器选择给出。同时，这一阶段将给出T寄存器的值，并将其余控制信号继续向后传播。

### 2.1.4 访存(MEM)

在本阶段将根据译码和执行阶段给出的控制信号进行内存的读写。我们的设计中，采取典型的“冯·诺依曼”结构，数据和指令不进行分离，而是将程序当做数据一样对待，因此会与取指阶段形成结构冲突。我们采取倍频的方式解决这个问题。内存中我们使用了THINPAD教学板上的50M时钟，而我们的主频将其四分频为12.5M。这样，这四个周期中，前两个周期将进行取指操作，后两个周期进行访存操作，避免了冲突。

另一个重要的方面是，若上一条指令涉及到LW操作，而当前指令又需要使用到上一条指令目标寄存器的值，则会造成写后读冲突。该冲突是不能通过旁路解决的。因此必须加一个气泡，让流水线暂停一个周期。这一过程将通过冒险检测单元（Hazard Detection Unit）来实现。在我们的设计中，该模块将被集成至旁路**bypass**模块。

此外，由于RAM1与CPLD共用基本数据总线和地址总线，使用时要把RAM1的总使能始终置‘1’。

### 2.1.5 写回(WB)

对于执行和访存得到的结果，将其写回寄存器堆。由于存在写后读冲突，为了提高流水线工作效率，该阶段存在对ID和EXE阶段的旁路。

## 2.2 数据通路

我们的数据通路如Figure 1所示。

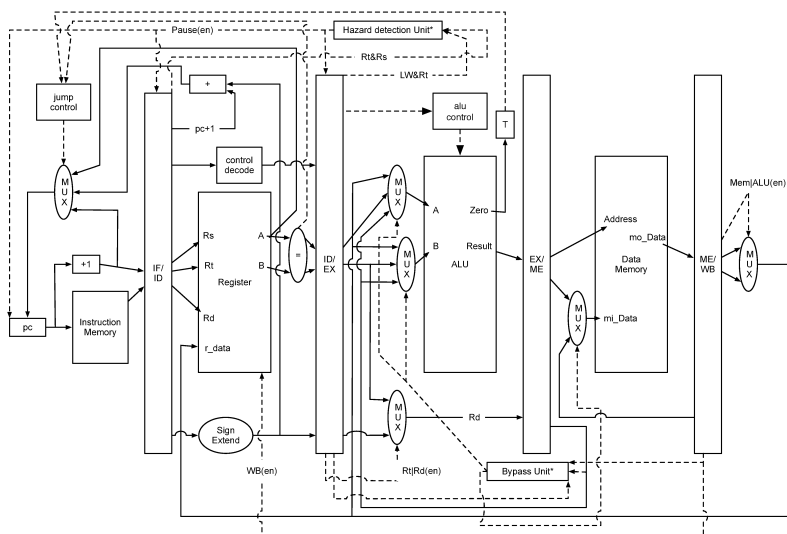


Figure 1: 数据通路

其中，虚线部分表示控制信号。我们在基础要求之上，在初始的数据通路里就增加了旁路单元（Bypass Unit）以解决数据冲突。它将后续阶段产生的一些信号尽快传送到需要使用它的位置。我们将在4.1.1中详细论述。对于不可避免的数据冲突（上一条指令是LOAD语句，它的写入寄存器和当前某一源寄存器相同），需要通过冒险检测单元（Hazard Detecting Unit）检测并停止流水线一个周期。关于流水线的暂停，我们将在4.1.2中详细介绍。

## 2.3 指令流程表

对于我们需要实现的指令，列举对应阶段要进行的操作如下：

	IF	ID	EXE	MEM	WB
<b>R</b>					
ADDU	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=R[y]	C:=A+B		R[z]:=C
AND	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=R[y]	C:=A&B		R[x]:=C

CMP	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=R[y]	T:=(A!=B)		
MFIH	IR:=MEM[PC] PC:=PC+1	A:=IH			R[x]:=A
MFPC	IR:=MEM[PC] PC:=PC+1	A:=PC			R[x]:=A
MTIH	IR:=MEM[PC] PC:=PC+1	A:=R[x]			IH:=A
MTSP	IR:=MEM[PC] PC:=PC+1	A:=R[x]			SP:=A
OR	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=R[y]	C:=A B		R[x]:=C
SLLV	IR:=MEM[PC] PC:=PC+1	B:=R[x] A:=R[y]	C:=A<<B		R[y]:=C
SLT	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=R[y]	T:=(A<B)		
SUBU	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=R[y]	C:=A-B		R[z]:=C
<b>I</b>					
ADDIU	IR:=MEM[PC] PC:=PC+1	A:=R[x]	C:=A+IMM		R[x]:=C
ADDIU3	IR:=MEM[PC] PC:=PC+1	A:=R[x]	C:=A+IMM		R[y]:=C
ADDSP	IR:=MEM[PC] PC:=PC+1	A:=SP	C:=A+IMM		SP:=C
ADDSP3	IR:=MEM[PC] PC:=PC+1	A:=SP	C:=A+IMM		R[x]:=C
LI	IR:=MEM[PC] PC:=PC+1				R[x]:=IMM
LW	IR:=MEM[PC] PC:=PC+1	A:=R[x]	C:=A+IMM	MR:=MEM[C]	R[y]:=MR
LW_SP	IR:=MEM[PC] PC:=PC+1	A:=SP	C:=A+IMM	MR:=MEM[C]	R[x]:=MR
SLL	IR:=MEM[PC] PC:=PC+1	A:=R[y]	C:=A<<IMM		R[x]:=C
SLTI	IR:=MEM[PC] PC:=PC+1	A:=R[x]	T:=A<IMM		
SRA	IR:=MEM[PC] PC:=PC+1	A:=R[y]	C:=A>>IMM		R[x]:=C
SW	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=R[y]	C:=A+IMM	MEM[C]:=B	
SW_SP	IR:=MEM[PC] PC:=PC+1	A:=SP B:=R[x]	C:=A+IMM	MEM[C]:=B	
<b>B</b>					



B	IR:=MEM[PC] PC:=PC+1	PC:=PC+IMM			
BEQZ	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=0 if(A==B) PC:=PC+IMM			
BNEZ	IR:=MEM[PC] PC:=PC+1	A:=R[x] B:=0			
		if(!(A==B)) PC:=PC+IMM			
BTEQZ BTEQZ	IR:=MEM[PC] IR:=MEM[PC] PC:=PC+1	if(!T) PC:=PC+IMM			
BTNEZ BTNEZ	IR:=MEM[PC] IR:=MEM[PC] PC:=PC+1	if(T) PC:=PC+IMM			
<b>J</b>					
JR	IR:=MEM[PC] PC:=PC+1	A:=R[x] PC:=A			

上表中，A与B表示寄存器堆的输出，C表示ALU的输出，在数据通路中，可以清晰地看出。以下内容中，我们同样遵循这个约定。

### 3 主要模块实现

这一部分我们将主要介绍我们通过硬件语言具体实现各模块的方法，包括各部件的主要工作过程和各模块之间的层次关系。在我们的硬件代码部分中，我们主要增加了旁路、VGA和显存、键盘响应、软件中断、冒险检测和流水线暂停等扩展功能。在本章节，这些扩展功能的实现将仅作简要的概括和说明，具体实现将在第4章里详细阐述。

我们的顶层模块包括了顶层的管脚绑定，以及各模块之间的接口信息。方便起见，以下我们将介绍所有计算机系统中使用到的接口信息，不包括调试信息，因此会与程序中的PORT有一些微小的出入。顶层模块的接口如下：

信号名	类型	功能说明
-----	----	------

clk_right	in	50M时钟
rst	in	reset, 如果reset按下, 将PC清零, 寄存器清零, 所有状态机回到初始状态。
wrn, rdn	out	uart的输出信号
data_ready, tbre, tsre	in	urat的输入信号
ram1_data	inout	ram1的数据, 这里用作串口数据
ram1_en	out	ram1的使能, 由于使用串口, 这里将常置'1'
ram2_data	inout	ram2的数据, ram2将作为内存使用
ram2_addr	out	ram2的地址
ram2_en, ram2_oe, ram2_we	out	ram2的使能信号
hs,vs	out	VGA的行、场同步信号
red, green, blue	out	VGA的rgb信号
ps2data, ps2clock	in	键盘的输入数据和时钟信号

Table 2: 顶层模块的输入输出接口

该模块之下是若干基本模块的接口, 我根据其层次结构和组织一一介绍如下。

### 3.1 PC寄存器单元

PC寄存器单元将产生当前PC值, 并将PC值给出, 供取指使用。为了简化模块, 这个模块中, 我们集成了PC加一的加法器模块, 而并没有单独将PC加一的加法器写成单独的模块。PC的输入将由branch模块的输出信号`next_pc`给出, 它将对 $(PC+1)$ 和跳转到的PC值做出选择。该模块我们将在 3.2 中详细介绍。

本阶段的主要时序逻辑即是在每个时钟到来时, 将输入的PC的值锁存至寄存器。

本模块的输入输出端口如下所示：

信号名	类型	功能说明
clk	in	50M四分频后的时钟
rst	in	reset信号
pc_in	in	输入的PC寄存器值
pc_out	out	输出的下一周期的PC寄存器值
pc_plus1	out	PC加一的值
pc_en	in	关于冒险检测给出的PC使能。若该信号为0，则流水线停止一个周期
pc_int_pause	in	指令中断给出的PC使能。若该信号为0，说明处于中断中，PC值不变

Table 3: PC寄存器模块的输入输出接口

### 3.2 branch跳转控制模块

在本模块中，我们将对来自不同途径的PC值做出选择，并给出下一个周期的PC值。PC值可以来自以下几种途径：

1. 没有跳转， $next\_pc$ 为 $pc+1$
2. J型跳转，跳转至寄存器中数据（即寄存器堆的输出信号A）所指示的位置，即 $PC = A$
3. B型跳转，直接跳转至偏移量为IMM（IMM为B型指令中的立即数，下同）的位置。即 $PC = PC + 1 + IMM$
4. B型跳转，根据T寄存器的值决定是否跳转至偏移量为IMM的位置。即当T寄存器符合相应条件时， $PC = PC + 1 + IMM$
5. B型跳转，根据寄存器堆输出的信号A是否为0决定是否跳转至偏移量为IMM的位置。即当 $A = 0$ 时， $PC = PC + 1 + IMM$

我们用信号 $ae0$ 表示信号 $A$ 是否为0。根据以上分析，我们只需根据指令类型  
和 $ae0$ 以及 $T$ 的值，通过选择器即可得到 $next\_pc$ 。这个阶段是纯组合逻辑模  
块，不含时序逻辑部分。据此得到的输入输出端口如下所示：

信号名	类型	功能说明
imm	in	经过立即数扩展单元后得到的立即数
a	in	寄存器堆输出的信号A，即 $Rx$ 寄存器的值
pc	in	PC加一的值
pc2	in	phase_1输出的pc，考虑了中断保存的pc值
instruction	in	指令，用其前几位作选择控制信号
t	in	T寄存器的值
next_pc	out	输出的下一周期的PC寄存器值

Table 4: 跳转模块的输入输出接口

### 3.3 IF/ID寄存器组模块

该模块将IF和ID两个阶段隔离开，并将控制信号在流水线上传递下去。同时，由于译码之后要立即进入中断，我们的软件中断也在这个阶段实现。关于中断的详细阐述将在4.2里进行，这里仅对该模块做简要说明。

本阶段的主要时序逻辑有以下两个部分：

**interrupt** 中断的时序，为一个状态机，这一部分将在扩展部分4.2里详细介绍

**record** 每一个时钟周期到来时，将输入的信号传递到下一个流水段

这个模块的输入输出端口如下所示：

信号名	类型	功能说明
-----	----	------

clk	in	50M四分频后的时钟
enable	in	使能信号。若其为0，则处于流水线暂停阶段，保持所有信号一个周期
rst	in	reset信号
pc_in	in	输入的PC寄存器值
instruction_in	in	输入的指令
pc_out	out	输出的PC值，考虑中断情况，PC需要保持，否则将PC值在流水线上传递下去
instruction_out	out	输出的指令
pause_pc	out	中断标志，其为0表示在中断中

Table 5: IF/ID寄存器组模块的输入输出接口

### 3.4 立即数扩展模块

在16位MIPS中，对于输入的立即数，由于指令的不同，采取的形式（有无符号）、长度也各不相同，我们将在立即数扩展模块中将其扩展为16位二进制数输出。这一过程只需根据instruction信号，选择指令中的立即数字段扩展即可。值得一提的是，对于有符号数和无符号数需要区别对待，实现也非常简单，只需在指令中找出相应的符号位，记录下来作为扩展后的数字的前x位（即需要扩展的位数）即可。

此外，本阶段为纯组合逻辑阶段，不涉及时序逻辑。

这一部分的输入和输出端口如下：

信号名	类型	功能说明
instruction	in	当前指令
imm	out	扩展后的立即数

Table 6: 立即数扩展模块的输入输出接口

### 3.5 寄存器划分模块

该模块是译码过程中最重要的模块。虽然叫寄存器划分模块，该模块其实实现的是译码的工作。这一过程中，我们将根据不同的指令划分出寄存器的编号，同时，这个阶段给出了几个重要的控制信息：ALU运算的第二个操作数是立即数还是寄存器组的输出信号B、是否需要写回、是否要对T寄存器进行操作。同时，该阶段将给出T寄存器的控制信号，即是不等、判小于还是判无符号大于。最后，本阶段将给出三个寄存器 $Rs$ 、 $Rd$ 、 $Rt$ 的编号和上面提到的控制信号。同样，我们将整个指令作为输入。在分析的过程中，由于指令非常不规整，我们通过第一位、前五位、前八位依次区别，可以区分所有的指令。再根据指令的具体功能，用几个多路选择器给各个控制信号赋值即可。这即是一条指令的解析过程。

同样地，本阶段也为纯组合逻辑模块，不涉及时序。

这一过程中，输入输出的端口如下：

信号名	类型	功能说明
instruction	in	当前指令
rs, rt, rd	out	输出从指令中分析得到的三个寄存器的编号
write_back	out	写回使能 [0 : no][1 : yes]
choseBIMM	out	ALU的第二操作数选择寄存器组的控制信号 [0 : B][1 : IMM], 下同
t_en	out	T寄存器使能 [0 : no][1 : yes], 下同
t_chose	out	T寄存器运算类型控制信号 [00 : !=][01 : <][10 : u >], 下同

Table 7: 寄存器划分模块的输入输出接口

### 3.6 寄存器组模块

本阶段将对寄存器组进行访问，根据寄存器划分阶段得到的 $Rs$ 、 $Rd$ 、 $Rt$ ，访问寄存器组里的相应寄存器，得到寄存器的值并输出。

在我们的设计中，将IH和T寄存器当做普通寄存器，给以编号放在寄存器组中，而PC寄存器作为特殊寄存器单独用一个模块给出。同时，T寄存器的赋值和输出是一个时序的过程，当T使能为真，在每一个时钟，将接受输入的T值赋给T寄存器并输出。

注意到，当涉及SW操作时，需要从寄存器堆读出相应寄存器的值，并作为输入数据给内存单元。但此时寄存器组模块中正在执行的指令，即正在进行第二阶段ID的指令，距离正在进行访存MEM的语句已经走过了两个周期。此时取得的 $Rt$ 寄存器不是访存指令涉及到的寄存器。因此，要将phase3，即EXE/MEM分隔模块寄存器堆输出的 $Rt$ 引回寄存器模块为信号 $Rm$ ，将该寄存器内数据作为写入内存的数据读出，由于写回是流水线上的最后一个阶段，这一过程没有引入更多的冲突。

值得注意的是，虽然寄存器读出的值不一定是ALU最终的操作数，这一问题将通过旁路在执行阶段解决，即执行阶段将对寄存器的输出和旁路的输出进行选择。而在寄存器组模块不需进行额外的处理。此外，本阶段还要进行正常的写回操作。

这一阶段的时钟采取与内存相同的50M时钟。该阶段的时序逻辑主要有：

**reg\_write** 当寄存器写回信号为真，进行寄存器的写回操作

**reg\_t** 将输入的T值锁存进寄存器

本阶段的主要输入输出接口如下所示：

信号名	类型	功能说明
clk	in	50M时钟
rst	in	reset信号
rs, rt	in	输入的源寄存器的编号

rd	in	输入的目标寄存器的编号
rm	in	写内存的数据所来自的寄存器编号
r_data	in	需要写回寄存器的数据
a_out, b_out	out	两个源寄存器的数据输出
m_out	out	需要写入内存的数据输出
wr_en	in	写寄存器使能
zero_equal	out	第一个源寄存器数据与0比较的结果
t_in	in	输入的T寄存器值
t_out	out	输出的T寄存器值
t_en	in	T寄存器使能

Table 8: 寄存器组模块的输入输出接口

### 3.7 ID/EXE寄存器组模块

这一个模块承接了ID阶段和EXE阶段，将上一阶段的数据和控制信号在流水线上继续传递下去即可。

该阶段唯一涉及的时序逻辑部分是将所有的输入信号锁存至寄存器组并输出。

这一阶段的输入输出接口如下所示：

信号名	类型	功能说明
clk, rst	in	输入时钟和reset信号
enable	in	使能信号，恒置'1'
pc_in, instruction_in	in	输入的PC值和当前指令
pc_out, instruction_out	out	输出的PC值和当前指令
a_in, b_in, imm_in	in	输入的寄存器组的输出信号A、B和立即数



a_out, b_out, imm_out	out	输入的寄存器组的输出信号A、B和立即数
rs_in, rt_in, rd_in	in	输入的两个源寄存器编号和一个目标寄存器编号
rs_out, rt_out, rd_out	out	输出的两个源寄存器编号和一个目标寄存器编号
write_back_in	in	输入的写回使能信号
write_back_out	out	输出的写回使能信号
choseBIMM_in	in	输入的B信号和立即数的选择信号
choseBIMM_out	out	输出的B信号和立即数的选择信号
t_en_in	in	输入的T寄存器使能信号
t_en_out	out	输出的T寄存器使能信号
t_chose_in	in	输入的T寄存器操作类型控制信号
t_chose_out	out	输出的T寄存器运算类型控制信号

Table 9: ID/EXE寄存器组模块的输入输出接口

### 3.8 算术逻辑单元模块

ALU是EXE阶段中最重要的部分，它将包含全部的算术逻辑运算，并将结果传递下去。同时它将对T寄存器进行设置。由于EXE和ID刚好是相邻的阶段，T寄存器的值可以在设置之后被下一条指令的寄存器访问阶段读出，不会产生冲突。

值得注意的是，这里的两个输入信号A和B不完全来自寄存器堆的输出，由于流水线的交叠，两个操作数还有可能来自于上一条指令或上上条指令的执行结果或访存结果。当操作数来自上条指令的执行结果或上上条指令的执行或访存结果时，可以通过旁路，将计算结果尽快送到这里，再通过一个多路选择器得到实际的操作数值。当该操作数来自上个阶段的访存结果时，操作尚未进行，不能通过旁路解决，此时必须暂停流水线。关于旁路和暂停流水线的具体

实现，我们将在4.1.1中进行详尽的介绍。在这里只需注意到，两个操作数的输入信号实际上都来自旁路的输出和寄存器输出的选择。

同时，整个ALU也是由组合逻辑构成的，不涉及时序逻辑。

其输入输出端口如下：

信号名	类型	功能说明
instruction	in	当前指令
op1	in	操作数1
b	in	操作数2
imm	in	立即数
t_chose	in	T寄存器运算类型控制信号
t_result	out	T寄存器运算结果
choseIMM	in	是否选择立即数
c	out	运算结果输出

Table 10: ALU模块的输入输出接口

### 3.9 EXE/MEM寄存器组模块

与第二阶段寄存器组类似，这一模块的主要功能是将EXE阶段的数据和控制信号向下传递。该阶段唯一涉及的时序逻辑部分是将所有的输入信号锁存至寄存器组并输出。

这一阶段的输入输出接口如下所示：

信号名	类型	功能说明
clk, rst	in	输入时钟和reset信号
enable	in	使能信号，恒设为1
pc_in, instruction_in	in	输入的PC值和当前指令
pc_out, instruction_out	out	输出的PC值和当前指令

a_in, b_in, c_in, imm_in	in	输入的ALU操作数A、B、立即数imm以及ALU运算结果C
a_out, b_out, c_out, imm_out	out	输出的ALU操作数A、B、立即数imm以及ALU运算结果C
rs_in, rt_in, rd_in	in	输入的两个源寄存器编号和一个目标寄存器编号
rs_out, rt_out, rd_out	out	输出的两个源寄存器编号和一个目标寄存器编号
write_back_in	in	输入的写回使能信号
write_back_out	out	输出的写回使能信号
choseBIMM_in	in	输入的B信号和立即数的选择信号
choseBIMM_out	out	输出的B信号和立即数的选择信号
t_en_in	in	输入的T寄存器使能信号
t_en_out	out	输出的T寄存器使能信号
t_chose_in	in	输入的T寄存器操作类型控制信号
t_chose_out	out	输出的T寄存器操作类型信号

Table 11: EXE/MEM寄存器组模块的输入输出接口

### 3.10 内存模块

由于RAM1与UART共享数据总线和地址总线，在我们的设计中，我们使用RAM2作为内存，将RAM1使能恒置为'1'，并将串口与内存同时包括在内存模块下。同时，我们采取“冯·诺依曼”计算机结构，将所有的程序看做数据，并存放在同一块内存中。这意味着第一阶段中的取值和第四阶段的访存可能会造成冲突，我们通过倍频的方法解决了这一问题，具体的方案在3.10.2中做了详细的阐释。

内存模块顶层涉及到的输入和输出信号主要是：

信号名	类型	功能说明
clk, rst	in	输入50M时钟和reset信号
ram2_data	inout	RAM2的数据
ram2_en, ram2_we, ram2_oe	out	RAM2的使能信号
ram2_addr	out	RAM2的地址信号
ram1_data	inout	串口的数据信号
tbre, tsre, data_ready	in	与CPLD通信信号
wrn, rdn	out	与CPLD通信信号
address_pc	in	PC地址
data_pc	out	PC值输出
address	in	要访问的显存地址
data_in	in	写入数据值
instruction	in	当前指令
data_out	out	输出数据
hs,vs	out	行场同步信号
r,g,b	out	VGA输出的rgb值
ps2data, ps2clock	in	键盘的时钟和数据信号

Table 12: 内存模块的输入输出接口

下面我们将一一介绍其下包含的四个子模块：UART模块、SRAM模块、VGA和显存模块和键盘模块。

### 3.10.1 UART模块

在我们的程序里，我们将通过UART模块与串口通信。串行接口是一类使用相对简单的接口，THINPAD教学计算机上也配置了基本的串口，作为连接终端设备的接口。

THINPAD教学计算机的FPGA芯片通过基本总线连接了存储器芯片RAM1以及被配置为UART的扩展芯片CPLD。UART连接RS-232接口，作为串口与其

他设备连接。

异步接收/发送器（*UART*），可完成并行数据和串行数据之间的相互转换，还能检测串行通信在传送过程中可能发生的错误。*UART*主要由数据总线接口、控制逻辑、波特率发生器、发送部分和接收部分等组成。本实验主要涉及*UART*中最重要的发送部分和接收部分，其功能包括发送缓冲器（*tbr*）、发送移位寄存器（*tsr*）、帧产生、奇偶校验、并转串、数据接收缓冲器（*rbr*）、接收移位寄存器（*rsr*）、帧产生、奇偶校验、串转并。

数据的发送由*UART*中的微处理器控制，微处理器给出*wrn*信号，发送器根据此信号将并行数据*din*[7..0]锁存进发送缓冲器*tbr*[7..0]，并通过发送移位寄存器*tsr*[7..0]发送串行数据至串行数据输出端*sdo*。在数据发送过程中用输出信号*tbre*、*tsre*作为标志信号，当一帧数据由发送缓冲器*tbr*[7..0]送到发送移位寄存器*tsr*[7..0]时，*tbre*信号为1，而数据由发送移位寄存器*tsr*[7..0]串行发送完毕时，*tsre*信号为1，通知CPU在下个时钟装入新数据。

在之前的小实验中，我们已经初步了解了串口的工作步骤。需要注意的是高阻态的设置，当写使能为'0'时，需要置data为高阻态。

这里涉及到的时序逻辑通过一个状态机来解决，每一个CPU周期内，将根据不同的指令，从IDLE状态选择进入读或写状态，再进行相应的操作即可。该状态机大致工作流程如下：

UART模块状态机	
IDLE	设rdn和wdn为1 转入choose状态
WRITING	rdn设为0 转入IDLE_READ
WRITING	wrn设为0 转入IDLE
IDLE_READ	输出数据设为data，rdn设为1 转入IDLE

其中*choose*信号是根据当前指令得到的，它的值为WRITNG、READING或IDLE。

它的输入输出端口如下：

信号名	类型	功能说明
clk, rst	in	输入时钟和reset信号
data	inout	串口数据
tbre, tsre, data_ready	in	与CPLD通信信号
wrn, rdn	out	与CPLD通信信号
data_in	in	输入数据
data_out	out	输出数据
wr_en	in	写使能信号
enable	in	URAT的使能信号

Table 13: URAT模块的输入输出接口

### 3.10.2 SRAM模块

我们使用RAM2作为内存，这样就避免了和与串口的冲突。SRAM模块需要解决的问题是，由于我们采取了经典的“冯·诺依曼”计算机结构，将程序和数据同等对待，放在同一块内存地址中，对于第一阶段的取指操作和第四阶段的访存操作会产生同时读写内存的冲突。虽然它们的地址相互独立，不会涉及到对同一块内存的同时读写，但是由于在同一块RAM上，冲突是存在的。对于这个问题，我们采取的方法是，将内存频率设为50M时钟，而将CPU主频置为50M时钟的四分频，这样在内存读写的过程中，只需约定前两个周期进行取指，后两个周期进行访存即可，整个过程通过一个状态机来实现，这样有效地避免了冲突，且没有严重影响CPU的性能。

同时，我们通过一个state信号标识当前状态机处于哪一个状态下，作为控制信号，对地址信号进行选择。这个状态机的运转大致如下：

---

**SRAM模块状态机**


---

RO(Read Only)	设oe为0, state为00 转入IDLE1
IDLE1	state设为10, 输出数据设为ram_data, oe设为1 转入WR
WR(Write&Read)	state设为11, 根据输入设定we和oe的值 转入IDLE2
IDLE2	state设为01, 输出数据设为ram_data 转入RO

---

同时有效的访存地址将根据state选择, 相应的VHDL代码如下:

```

1 with state select
2     ram_addr <=
3         "00" & ram_addr_ro when "00",
4         "00" & ram_addr_wr when "11",
5         (others => '1') when others;
```

可以看出, 四分频后, CPU的一个时钟周期刚好在内存单元跑过四个时钟周期, 这四个周期中, 前两个将进行根据PC值取指的只读操作, 后两个将根据不同的指令进行内存的读写操作。二者不会冲突, 且可以在一个CPU时钟内同时完成这两项工作。

它的输入输出接口如下:

信号名	类型	功能说明
clk_high, rst	in	输入时钟和reset信号
ram_en, ram_we, ram_oe	out	RAM的使能信号
ram_addr	out	RAM地址
ram_data	inout	RAM数据
ram_data_in	in	RAM输入数据
ram_addr_ro	in	RAM输入只读的地址, 即取指用到的PC值
ram_addr_wr	in	RAM输入的访存地址, 可读可写

ram_data_out_ro	out	RAM得到的只读数据，即PC读到的当前指令
ram_data_out_wr	out	RAM得到的访存数据
wr_en	in	写使能

Table 14: SRAM的输入输出接口

### 3.10.3 VGA和显存模块

扩展模块。将在4.3中详细介绍。

### 3.10.4 键盘模块

扩展模块。将在4.4中详细介绍。

## 3.11 ME/WB寄存器组模块

与2、3阶段的阶段寄存器相同，这个阶段也只需将之前的数据和控制信号向下传递。

同样的，这一部分的时序只需在每一个时钟周期将输入信号锁存并向下传递即可。

该阶段的输入输出接口如下：

信号名	类型	功能说明
clk, rst	in	输入时钟和reset信号
enable	in	使能信号，恒设为1
pc_in, instruction_in	in	输入的PC值和当前指令
pc_out, instruction_out	out	输出的PC值和当前指令
a_in, b_in, c_in, imm_in	in	输入的ALU操作数A、B、立即数imm以及ALU运算结果C



a_out, b_out, c_out, imm_out	out	输出的ALU操作数A、B、立即数imm以及ALU运算结果C
mem.data_in	in	输入访存阶段得到的数据
mem.data_out	out	输出访存阶段得到的数据
rs_in, rt_in, rd_in	in	输入的两个源寄存器编号和一个目标寄存器编号
rs_out, rt_out, rd_out	out	输出的两个源寄存器编号和一个目标寄存器编号
write_back_in	in	输入的写回使能信号
write_back_out	out	输出的写回使能信号
choseBIMM_in	in	输入的B信号和立即数的选择信号
choseBIMM_out	out	输出的B信号和立即数的选择信号
t_en_in	in	输入的T寄存器使能信号
t_en_out	out	输出的T寄存器使能信号
t_chose_in	in	输入的T寄存器操作类型控制信号
t_chose_out	out	输出的T寄存器操作类型信号

Table 15: MEM/WB阶段寄存器组模块的输入输出接口

### 3.12 写回模块

在这一模块，我们将得到的寄存器数据写回寄存器堆。具体分析每一条指令，寄存器的数据可能来自：

- 立即数imm
- ALU的第一个操作数A
- ALU的运算结果C
- 访存结果

- PC寄存器的值

综合上述分析，即可得到写回模块的输入输出接口如下：

信号名	类型	功能说明
imm	in	立即数
a	in	ALU的第一个操作数A
c	in	ALU的运算结果C
memread	in	访存结果
pc	in	PC寄存器的值
instruction	in	当前指令
data	out	输出需要写回的数据

Table 16: 写回模块的输入输出接口

## 4 扩展功能介绍

### 4.1 冲突解决

#### 4.1.1 数据旁路解决数据冲突

在MIPS流水线中，唯一可能发生的数据冲突是写后读（RAW）冲突。写后读冲突是指，指令j的执行需要使用指令i的计算结果，但是当它们在流水线中重叠执行时，指令j可能在指令i将其计算结果写入之前就先行对保存该计算结果的寄存器进行了读操作，这样指令j读出的寄存器值就是错误的。解决写后读冲突的一个常用办法是采用旁路（定向）技术。

旁路将把计算得到的结果尽快传送到它需要的位置。由于MIPS流水分为五段，且在最后一个阶段WB将对寄存器进行写回，而需要对寄存器进行访问并取得寄存器的值的操作是在第二阶段和第三阶段。因此，旁路将存在于以下几处：

- 上一条指令的EXE阶段对当前指令的ID段的旁路

- 上上一条指令的EXE阶段对当前指令的ID段的旁路
- 上上一条指令的MEM阶段对当前指令的ID段的旁路
- 上一条指令的MEM对当前指令的EXE段的旁路
- 上上条指令的WB对当前指令的EXE段的旁路

当上一条指令是LW指令，且当前指令的源寄存器是上一条指令的目标寄存器时，该数据冲突不能通过数据旁路解决，必须插入一个气泡。这一部分我们将在4.1.2中阐述。

因此，对于数据旁路模块，其输入输出端口如下：

信号名	类型	功能说明
a_1, a_2, a_3	in	当前和前两个周期的寄存器输出信号A
b_2	in	当前周期的寄存器输出信号B
c_2, c_3	in	上个周期和上上周期的运算结果
imm_2, imm_3	in	上个周期和上上周期的立即数
pc_2, pc_3	in	上个周期和上上周期的PC寄存器值
memread_3	in	上上阶段的访存结果
instruction_1, instruction_2, instruction_3	in	当前和前两个周期的指令
write_back_data	in	写回数据
rs_1, rs_2, rt_2, rd_2, rd_3, rd_4	in	当前阶段和前两个阶段涉及的寄存器编号
writeback_2, writeback_3 writeback_4	in	当前和前两个阶段的写回使能
a1_out	out	MEM和EXE阶段ID段的旁路给出的A信号
a2_out	out	MEM和WB阶段到EXE阶段的旁路给出的A信号

b2_out	out	MEM和WB阶段到EXE阶段的旁路给出的B信号
pause_signal	out	流水线暂停标志

Table 17: 旁路模块的输入输出接口

#### 4.1.2 流水线暂停解决数据冲突

我们将控制流水线暂停的Hazard Detection Unit集成在旁路模块中，至此解决了所有的数据冲突。通过检测符合以下两个要求可以获知需要暂停流水线一个周期：

1. 上一个指令是Load指令
2. 上一个指令的写入寄存器和当前指令的某一源寄存器相同

此时设置旁路模块中的输出信号`pause_signal`为1，并将其连接到PC寄存器单元和IF/ID寄存器组单元作为使能信号，执行以下操作：

1. 让当前指令的控制信号全部为0，即不进行任何写入操作
2. 让PC值保持不变
3. 让IF/ID段寄存器保持不变

即实现了暂停流水线的操作。

#### 4.1.3 倍频解决结构冲突

如果指令和数据放在同一个存储器，取指阶段和访存阶段会造成存储器争用冲突，这个问题在3.10.2里已经有论述。即将CPU主频改为内存的四分频，并约定在每个周期的前一段进行取指，后一段进行访存。具体可以参见3.10.2里的介绍。

#### 4.1.4 软件解决控制冲突

对于分支语句，若进行分支预测，即使准确率达到90%以上，也不可能全部命中。分支预测带来的效率问题一般是在有循环语句时出现。对于这个问题，我们采用的方法是，对于所有branch delay slot中的语句，默认被执行。事实上在MIPS中，也是这么实现的。我们尝试使用了一些MIPS的模拟器，如QtSpim，发现branch delay slot里的语句是一定执行的。这样在软件设计时，考虑到MIPS的这一特性，在大多数程序中，在不影响原意的前提下。可以将循环语句内最后一句移出，放在跳转语句之下。这样这一句话始终会被运行，不需要进行分支预测。程序将正常运转，且由于没有预测失败的成本，其效率会比进行分支预测的程序要高。同样地，对于跳转语句，也可以通过软件设计，将一些一定要运行的语句放在跳转语句的后一句，让branch delay slot始终执行有效的语句。

实际上，在我们所编写的软件中，我们也是这么操作的。把这个问题交给软件，有效地简化了硬件的设计。

## 4.2 中断

在我们的设计中，我们实现了软件中断。即实现了指令集中的INT指令。该过程是在IF/ID寄存器组阶段完成的。在这一阶段，已经取得指令，立即可以分析出是否是中断指令。若是，则进入一个状态机，将当前PC值和中断号压栈，进入中断处理（kernel第7行，DELINT行号标识）。该状态机的状态标识如下：

```

1  with state_int select
2  instruction_int <=
3      "1110111001000000" when "0001", -- <ee40>  MFPC R6
4      "0100111000000000" when "0010", -- <4e00>  ADDIU R6 00
5      "0110001111111111" when "0011", -- <63ff>  ADDSP FF
6      "1101011000000000" when "0100", -- <d600>  SW_SP R6 00
7      "011011100000" & imm_tmp when "0101", -- <6e00>  LI R6 00
8      "0110001111111111" when "0110", -- <63ff>  ADDSP FF
9      "1101011000000000" when "0111", -- <d600>  SW_SP R6 00
10     "0110111000000111" when "1000", -- <6e07>  LI R6 07
11     "1110111000000000" when "1001", -- <ee00>  JR R6

```

```
12 | instruction_in when others;
```

我们在每一个语句后面加上了汇编代码的注释，整个处理的过程可以非常清晰地看出。

这里的imm\_tmp是中断指令中立即数的值。每一个时钟周期到来的时候，状态会加一，直至处理完整个中断过程。

### 4.3 VGA和显存

在我们的设计中，我们加入了VGA模块和显存，这一部分被继承至内存模块。

在VGA的顶层模块，我们涉及几个子模块：VGA显示模块、显存模块、字符集ROM模块。VGA显示模块将扫描屏幕上的每一个点，根据其坐标，去显存模块取得当前位置的需要显示的字符编码。再根据字符编码，去ROM的相应位置取得该像素点的rgb值并显示。顶层模块的输入输出端口如下所示：

信号名	类型	功能说明
address	in	输入显存地址
data	in	输入显存数据
enable	in	使能信号
clk_right,rst	in	输入时钟和reset信号
hs,vs	out	行场同步信号
r,g,b	out	rgb色彩输出信号

Table 18: VGA顶层模块的输入输出接口

#### 4.3.1 VGA显示模块

VGA显示的工作流程大致如下：

1. 对输入的50MHz信号二分频，产生一个25MHz的信号

2. 根据需要的屏幕像素，产生行同步和场同步的计数矢量。对于 $640 \times 480$ 的像素，考虑消隐区，每行像素数800，每场行数525
3. 由行同步和场同步计数矢量产生行同步信号和场同步信号
  - 行同步：656-752为低电平，其余为高
  - 场同步：490-492为低电平，其余为高
4. 色彩输出
  - 在显示区域给每个像素点RGB进行赋值
  - 在整个消隐区，RGB均赋值为0

据此我们根据当前像素点的坐标，计算出它属于屏幕的哪一个分块，即得到相应的显存地址。再从显存中得到相应的字符编号，去ROM中读取每一个像素上的rgb值输出。设定输入输出接口为：

信号名	类型	功能说明
address	out	输出当前像素颜色在RAM的地址
ram_data	in	输入的字符编号
rom_address	out	输出的字符编号，需要在ROM中寻址
reset	in	reset信号
q	in	得到的当前像素点的rgb信号
clk_0	in	输入时钟
hs,vs	out	行场同步信号
r,g,b	out	rgb色彩输出信号

Table 19: VGA显示模块的输入输出接口

### 4.3.2 VGA只读存储器模块

在VGA模块中，我们需要一块ROM，存储我们的字符集。我们采用了Xilinx里自带的core generator生成一个片内的ROM，里面存储了0-9的数字、a-z小写字母和一些色块，共64个字符。字符的编码由我们自己设定，并使用mif文件将其初始化。

这个ROM是通过它的generic map实现的。输入和输出端口如下：

信号名	类型	功能说明
clka	in	输入时钟
addra	in	输入地址
douta	out	输出数据

Table 20: VGA只读存储器模块输入输出接口

### 4.3.3 显存模块

同样地，我们使用了generic map中的RAM来作为显存。同时我们将我们的显存映射到内存中从0xF800开始的一段地址。这样访存的时候，如果地址值的前5位都为1，我们就当做是对显存进行的读写操作。而在软件的编写过程中，我们也只需要将给屏幕某一块显示某字符的过程用一个基本的访存语句SW实现，不需设计额外的指令。

显存模块的输入输出接口如下：

信号名	类型	功能说明
clka	in	VGA显示部分（只读）的输入时钟
wea	in	VGA显示部分（只读）的写使能（置0）
addra	in	VGA显示部分（只读）的地址
dina	in	VGA显示部分（只读）的输入数据
douta	out	VGA显示部分（只读）的输出数据



clkb	in	VGA写入部分的输入时钟
web	in	VGA写入部分的写使能端
addrb	in	VGA写入部分的地址
dinb	in	VGA写入部分的输入数据
doutb	out	VGA写入部分的输出数据

Table 21: 显存模块的输入输出接口

## 4.4 键盘

PS/2键盘中，每按一个键会产生如下数据：

- 按下一个键产生一组串行数据
- 松开产生两组串行数据

每组串行数据有11位，从低位开始传输。每组数据包括一位起始位（低电平），8位数据位（扫描码），1位奇偶校验位（1的个数为奇数取0，否则取1）和一位结束位（高电平）。

扫描码分为通码和断码，当一个键被按下就发送通码，被释放则产生断码。

PS/2键盘接口程序主要分为两个部分：

- 键盘数据接收及分析
  1. 滤波:键盘产生的数据具有毛刺，需要滤波
  2. 接收数据
    - 接收11个串行数据，串并转换
    - 奇偶校验
    - 提取其中8为扫描码
- 键盘数据译码及输出

**译码** 对接受来的数据注意区分是通码还是断码

**输出** 将译码得到的字符输出

简化起见，我们的字符集里只设计了0-9和a-z，不需要考虑上档键shift等问题。

同时我们将键盘模块设计成类似串口的形势，用BF02和BF03与其通信。并根据扫描码解析就可以得到当前按下的键。该模块的输入输出接口如下：

信号名	类型	功能说明
ps2data	in	键盘输入信号
ps2clock	in	键盘时钟
clk_right, rst	in	输入时钟和reset信号
rdn	in	读入标志
ready	out	数据准备标志
data	out	输出解码后的数据

Table 22: 键盘模块输入输出接口

## 4.5 双机通信

在顺利跑出kernel程序，并加入了VGA和键盘端口以及中断等操作，我们的硬件编程工作告一段落。在我们的软件创新中，我们用汇编编码写了一段即时通信系统，利用串口完成了双机通信的任务。

在双机通信的过程中，我们用两台THINPAD教学实验机，都烧入我们设计的计算机系统。同时，两台实验机的串口相连，这样通过串口，一台实验机就可以接受另一台实验机发送的数据并保存下来。此后，我们又将VGA模块和键盘模块加上，使输入来自键盘的PS/2接口，而输出将存入显存并定时刷新。这样，我们在双机通信的基础上，基本实现了一款即时聊天系统软件的功能。

具体的实现成果，我们将在5.4中展示。由于程序太长，不在实验报告里贴出。一并附在最终代码中（提交文件夹下的programs/IM.asm），并有相对完整的注释。

## 4.6 伪码解析

THCO MIPS指令集中，只有一些基本的指令，对于每一个组需要实现的指令来说，也并不是完整的指令集。在编写软件的过程中，经常需要用到一些常用的语句，如常见的PUSH、POP、MOVE等，并没有出现在指令集中或不在我们组实现的指令集内。为了编写软件的方便，我们设计了几个伪指令并且编写了相应的脚本对其进行编译。这些伪指令的形式和对应的THCO MIPS指令如下：

伪码格式	解析后的汇编码
PUSH R0	ADDSP 0xFF SW_SP Rx 0x0
Rx = Ry	ADDIU3 Ry Rx 0x0
Rx = 0xABCD	LI Rx 0xCD SLL Rx Rx 0x0 ADDIU Rx 0xAB
POP R0	LW_SP R0 ADDSP 0x1

Table 23: 伪指令说明

伪指令是汇编语言到高级语言的第一步，有了伪码解析器，我们的软件编写可以更接近高级语言。便于编写、阅读和调试。

相应的脚本见提交文件夹的accessory/assembler.py，我们也在提交程序中一并给出，希望可以给以后的同学带来方便。

## 4.7 其它贡献

### 4.7.1 管脚自动绑定脚本

在编写代码的过程中，经常需要做单元测试，此时管脚的绑定是一件非常冗长且易错的事情。经常因为绑错管脚而花费很长的时间调试代码。对此，我们写了一个管脚自动绑定的脚本，只需输入顶层vhd文件的路径，即可输出对应的ucf文件。唯一的要求是顶层信号名称与我们约定的相同。这里我们已经纠正了实验指导书上的几处管脚错误。相应的脚本见提交文件夹下的accessory/ucf\_maker.py。

这些约定的信号名如下：

信号名	说明
flash_addr	flash地址
flash_data,flash_addr	flash数据、地址
flash_rp,flash_we,flash_vpen flash_ce,flash_byte,flash_oe	flash控制信号
switch	手拨开关
ram1_addr,ram2_addr	ram地址信号
ram1_data,ram2_data	ram数据信号
ram1_oe,ram1_we,ram1_en	ram1使能信号
ram2_oe,ram2_en,ram2_we	ram2使能信号
oLed	LED灯
red,blue,green	VGA的rgb信号
ps2clock,ps2data	与ps2通信信号
iKey	微动开关
clk_right	50M晶振时钟
wrn,rdn,tbre,tsre,data_ready	与CPLD通信信号
clk	手拨时钟
vs,hs	VGA行场同步信号
reset	硬件复位

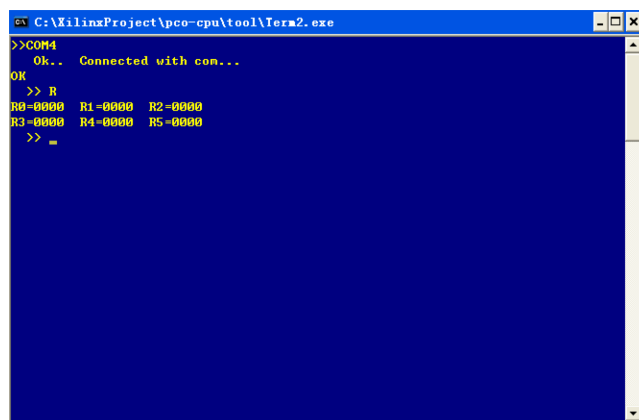


Figure 2: 链接测试

Table 24: 管脚自动绑定对应信号表

#### 4.7.2 实验指导书修改

在从Project1开始的实验中，我们发现实验指导书中有一些错误的地方。因此我们也希望可以协助老师做了修改，为今后做实验的同学提供便利。修改的内容将在之后单独提交给老师，也可以当面再对实验指导书的内容做一些讨论。

## 5 实验成果展示

### 5.1 运行kernel

Figure 2是我们将kernel代码烧入教学机后并连接串口的结果。简单测试计算斐波那契数的程序如Figure3，运行正常。

```

C:\XilinxProject\pcr-cpu\tool\Tera2.exe
>>
>> 0 5000
(5000) LI R1 1
code:16901
(5001) LI R2 1
code:16a01
(5002) LI R3 05
code:16b05
(5003) SLL R3 R3 0
code:3360
(5004) LI R4 19
code:16c19
(5005) SW R3 R1 0
code:4b20
(5006) SW R3 R2 1
code:4b41
(5007) ADDU R1 R2 R1
code:1e45
(5008) ADDU R1 R2 R2
code:1e49
(5009) ADDIU R3 02
code:4b02
(500a) ADDIU R4 FF
code:16c1f
(500b) DNEZ R4 F9
code:2cf9
(500c) NOP
code:0000
(500d) JR R7
code:1ef00
(500e) NOP
code:0000
(500f)
>> 0
int 指令中断
>> 0 5000
>> 0 0500
(0500) 0001
(0501) 0001
(0502) 0002
(0503) 0003
(0504) 0005
(0505) 0008
(0506) 0004
(0507) 0015
(0508) 0022
(0509) 0037
>>

```

Figure 3: 测试斐波那契程序

## 5.2 进入中断

Figure 4是我们测试INT指令的结果。软件中断运行正常。

## 5.3 在kernel里使用VGA和键盘响应

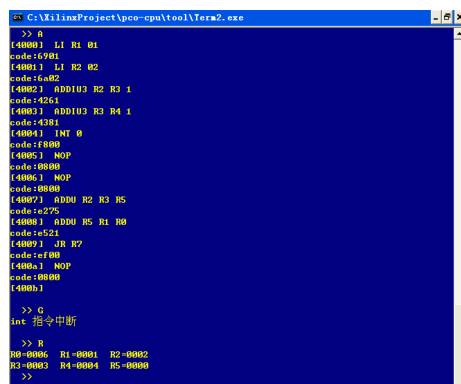
加入了键盘和VGA模块之后，我们用一段汇编代码来测试。该程序将接受键盘输入，将输入字符依次显示在屏幕上，包括回车（Enter）键的换行响应。程序代码见提交文件夹的programs/write2vga.asm。

该程序的代码如下：

```

1  LI R3 0
2  LI R4 1
3  LI R0 F8
4  SLL R0 R0 0
5  LI R1 BF
6  SLL R1 R1 0
7  LW R1 R2 03
8  BEQZ R2 FE

```



```

C:\XilinxProject\pcg-cpu\tool\ferw2.exe
>> 0
[4000] LI R1 01
code:6901
[4001] LI R2 02
code:6a02
[4002] ADDIU3 R2 R3 1
code:4261
[4003] ADDIU3 R3 R4 1
code:4381
[4004] INT 0
code:f800
[4005] NOP
code:0000
[4006] NOP
code:0000
[4007] ADDU R2 R3 R5
code:e275
[4008] ADDU R5 R1 R0
code:e521
[4009] JR R7
code:f000
[400a] NOP
code:0000
[400b]
>> 0
int 指令中断
>> 0
R0=0000 R1=0001 R2=0002
R3=0003 R4=0004 R5=0000
>>

```

Figure 4: 中断测试

```

9   NOP
10  LW R1 R2 02
11  ADDIU3 R0 R5 0
12  ADDU R3 R5 R5
13  ADDU R4 R5 R5
14  SW R5 R2 0
15  LI R5 1E
16  CMP R2 R5
17  BTNEZ 3
18  NOP
19  ADDIU R3 40
20  LI R4 0
21  ADDIU R4 1
22  B 7F0
23  NOP
24  JR R7
25  NOP

```

从程序中可以看出，我们从键盘的端口读入数据，存入VGA的显存部分，并显示出来。Figure 5展示了该程序运行的效果图。

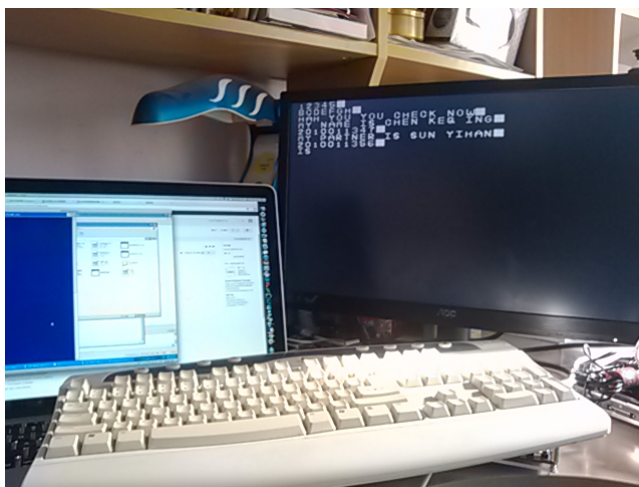


Figure 5: 在kernel中运行VGA和键盘响应程序

## 5.4 即时聊天工具

在有了双机通信的基础上，我们设计了一款即时聊天系统软件。主要的技术介绍在4.5里已经有，不再赘述。这里仅展示一下实践成果。

在往两个THINPAD教学机中烧入我们的计算机系统之后，将它们的串口相连，两台教学机分别接到独立的键盘和显示器，烧入即时聊天工具的代码并运行。此时屏幕将分为两栏，左栏是自己说过的话，右栏是对方说过的话。当一个人在键盘上输入符号，这些字符将同时在两台显示器上出现，实现了即时聊天的功能。Figure 6展示了一幅通信过程中的图片。由于两台电脑离得较远，截图的效果并不是非常好。可以看出，A屏幕的左半边和B屏幕的右边显示的是相同的内容。对于我们双机通信的效果，我们还录了一段DV（见提交文件夹的accessory/双机通信.mp4），附在文件中提交。

## 6 实验心得体会

计原大实验一直是计算机系大三标志性的事件之一。虽然一直听说自己会在大三亲手编写一台计算机，在开始做实验之前还是既期待又担心。“奋战二十天，做台计算机”可以说是这个过程的真实写照。





Figure 6: 双机通信视频截图

我们组只有两个人，并且在讨论和权衡下，选择了流水线的CPU的实现。开始的几次小实验我们基本上也是班里较早开始做的一组，虽然遇到了很多问题，但是在解决问题的过程中加深了我们对计算机系统的理解，也加深了我们对硬件编程语言的理解。由于较早开始调试，我们也给班里的其他同学提供了帮助，还是非常有成就感的。

编写一台计算机，过程虽然漫长，但是涉及到的都是课堂上讲过的知识，实践的过程加深了我们的理解。

开始做大实验之前，我们花了很长时间研究了数据通路，并提出了实验的目标。在初版的数据通路中就充分考虑了数据旁路、流水线暂停等问题。虽然在最后的实现中，数据通路和之前的图还是有较大出入，但是在开始编写代码之前充分地考虑这些问题有效地提高了效率，同时，我们仔细阅读了实验指导书和kernel代码，在充分理解了kernel的工作流程之后，才开始写代码。虽然看起来比别人晚开始，我们的初版代码写成之后，几乎没有经过大幅度的修改。调试的过程也轻松了很多。

同时，我们充分进行了单元测试。对于多数重要的模块，我们都编写了test程序进行单元测试，并将程序保留，方便之后的调试。事实证明，这些测试程序给调试工作提供了很多便利，它能帮助我们尽快定位错误的位置和原因。

值得一提的是，在写小实验的过程中，我们就萌发了写一个自动绑定管脚的脚本的念头。在对顶层信号名做了约定之后，我们写出了自动绑定管脚脚

本。这样每一次编写测试代码之后的测试工作中，不需要额外绑定管脚，单元测试的代码编写的效率高了很多。

另外，我们在程序初步代码写出之后，首先调通了串口。虽然调通串口经历了一个很漫长的过程，但是之后的调试就可以利用串口来进行。串口调试通过是我们第一次在板子上运行出的一个程序。也是我们得到的第一个正反馈，增强了我们的信心。之后，我们又测试了指令集里的所有指令，利用串口调试通过。

在正式跑kernel代码之前，我们自己手动测试了很多程序，并做了细致的单元测试。在跑起kernel代码之后，我们将自动时钟和手动时钟结合，有效地提高了效率。在整个实验过程中，我们没有用手按时钟定位调试。从第一次烧入kernel代码到kernel代码调试通过，前后只用了不超过4个小时的时间，基础部分宣告完成。这和之前细致的准备工作是分不开的。

后面我们又加上了各种扩展功能，也写了一个超过300行的软件运行即时通信系统，因为宿舍没有两个PS/2键盘，我们在检查的当天晚上在实验室进行了第一次测试，并立刻通过。看到自己编写出的计算机真的可以运行程序，并和VGA和键盘通信，真的是非常有成就感的。我们把整个过程录了一段DV作为留念。相信计原这一门课程，这一个大实验，也是每一个计算机系人永远抹不掉的记忆吧。

我们组只有两位同学，一位在公司实习，另一位在实验室有科研工作，但是，在完成大实验的过程中，两个人都付出了努力，也经常在宿舍一起熬夜到凌晨。没有抱怨也没有退缩，多次的讨论和尽力的合作使我们顺利完成了实验。虽然那一周几乎没有睡好过，但是感觉充实又有成就感。在此要向每一位成员致敬。

此外，整个过程我们使用GitHub实现合作和同步，也算是学习使用了一个新的版本控制系统。很有收获。

最后，感谢刘卫东老师课堂上的讲解。正是这些计算机系统结构的知识构成了这个大实验的基石。感谢李山山老师和分管计05班的助教的耐心指导，在我们对实验过程有疑问的时候，与他们的及时交流帮助我们尽快理解了实验内容，当我们有了想法的时候，他们也对我们的idea做出了评价并给出了建议。感谢计05班的每一位同学，我们班的多数同学都选择了流水线，虽然充满挑战性，但是交流和讨论使我们收获颇丰。感谢许欣然·邹林希组，在最辛苦的一

周，他们与我们同时熬夜，我们的相互讨论提高了彼此的效率，也让完成实验的过程充满了乐趣。感谢李百恩同学， 在我们遇到问题的时候给出了很多中肯的建议。相信这一次宝贵的经历将永远留在我们的记忆之后，也是我们本科生活中的一个标志性事件。