清华大学

计算机组成原理

计算机实验报告

支持 *THCO MIPS* 指令系统的流水线计算机设计与实现

作者：

董胤蓬、钱雨杰、桥本优

指导教师：

刘卫东、李山山

清华大学

计算机科学与技术系

December 10, 2015

Abstract

计算机实验报告

董胤蓬、钱雨杰、桥本优

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Contents

Chapter 1

实验目的

本实验是清华大学计算机科学与技术系开设的《计算机组成原理》课程实验。

实验任务是设计和实现一台 16 位支持指令流水的计算机。实验计算机的 CPU 采用五段流水线结构，支持 THCO MIPS 指令系统，并适当应用数据旁路、分支预测等技术提高流水线效率；使用 SRAM 作为存储器，并对内存进行管理；实验计算机的 I/O 通过串口与运行终端程序的 PC 连接，实现输入输出。实验计算机实现后需要运行监控程序，并可以通过监控程序实现用户写入指令、执行指令、查看寄存器和内存等操作。

实验可以在基础要求上进行一些扩展，包括软硬件中断处理、PS2 键盘输入、VGA 输出、双机通信、多道程序等。

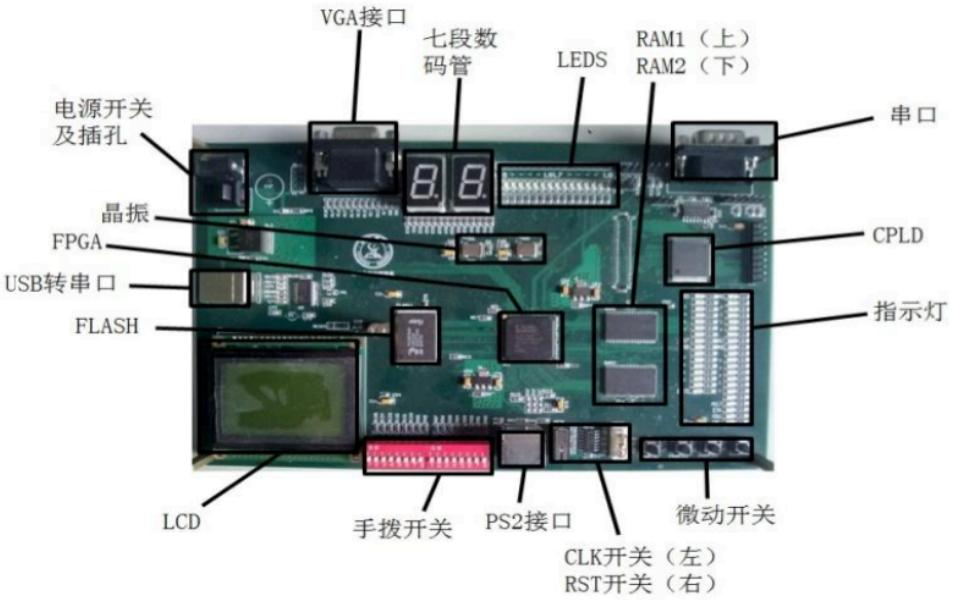
通过实验，可以加深对于计算机系统知识的理解，进一步理解和掌握流水线结构计算机的各部件组成和内部工作原理，掌握计算机外部设备输入输出的设计实现，培养硬件设计和调试的能力。

Chapter 2

实验环境

2.1 硬件环境

实验的硬件环境为 THINPAD 教学计算机硬件平台。整个硬件以大规模可编程逻辑器件为中心，通过总线连接 SRAM 存储器和 Flash 存储器，再配合以外围各种接口。计算机硬件平台如下图所示。



2.1.1 FPGA 芯片

THINPAD 教学计算机上的主实验芯片是一片 FPGA 芯片，是由 Xilinx 公司生产的 Spartan-3E 系列的 XC3S1200EFGG320 芯片。FPGA 芯片的具体技术参数为

器件名称	逻辑单元	系统门数	CLB 阵列	CLB 总数	用户 I/O	BlockRam（Kb）
XC3S1200E	19512	1.2×10^6	60×46	2168	304	504

FPGA 管脚连接 SRAM、FLASH 存储器和拨码开关、LED 灯、七段数码管、PS2、VGA、串口等外部设备。

2.1.2 CPLD 芯片

THINPAD 教学计算机上的扩展芯片是一片 CPLD 芯片，是由 Xilinx 公司生产的 XC9500 系列的 XC95144XL-7TQ100 芯片，它具有 100 个管脚、144 个宏单元，采用 TQFP 封装。

计算机组成原理实验中，扩展 CPLD 充当串口控制器，完成串行数据传输的功能。CPLD 配置成为一个 UART（通用异步收发器），是一种广泛使用的串行数据传输装置。

2.1.3 SRAM 存储器

THINPAD 教学计算机使用了 2 片 SRAM（Static Radom Access Memory）作为主要存储器，采用的是 ISSI 公司生产的异步高速 CMOS SRAM，型号为 IS61LV25616-10TI，每片存储容量为 256K×16b。

两块 SRAM 芯片中 RAM1 为基本内存，与 CPLD 共用基本数据总线和地址总线，RAM2 为扩展内存，拥有独立的地址线和数据线。

2.1.4 Flash 存储器

THINPAD 教学计算机使用 Flash 存储器存储实验系统数据。使用的 Flash 芯片型号为 MT28F640J3，数据线为 16 位，地址线为 23 位，可寻址空间 8MB。Flash 存储器兼具 RAM 和 ROM 的长处，不仅可以快速读取数据，也可以擦除、修改数据，同时数据不会因为断电丢失。

2.1.5 总线

THINPAD 教学计算机上共有三条总线，分别是基本总线、扩展总线和 Flash 总线，它们各自分别有数据、地址和控制线。

基本总线的数据线 16 位、地址线 18 位、控制线 3 位。基本总线连接有多个器件，包括实验 FPGA、基本内存 RAM1、扩展 CPLD 和 LED 指示灯，由 FPGA 控制总线的访问。

扩展总线的数据线 16 位、地址线 18 位、控制线 3 位，连接实验 FPGA 和扩展内存 RAM2。

Flash 总线的数据线 16 位、地址线 23 位、控制线 9 位，连接实验 FPGA 和 Flash 存储器。

2.1.6 外部接口

THINPAD 教学计算机硬件平台提供了一些常用外部设备的接口，包括 2 个七段数码管、16 位 LED 发光二极管、拨码开关、微动开关、复位开关、一个普通串口和一个 USB 转串口电路、一个 PS2 接口用于接收键盘输入，以及一个 VGA 接口用于进行显示器输出。

2.2 软件环境

2.2.1 FPGA 开发工具

实验使用的 FPGA 开发工具软件为 Xilinx ISE 14.7，使用的硬件开发语言为 VHDL。

2.2.2 THINPAD 软件包

实验使用 THINPAD 教学计算机软件包提供的监控程序、终端程序、Flash&RAM 读写程序、汇编程序等，并进行了相关程序的改进。

Chapter 3

实验设计

3.1 CPU 流水结构

3.1.1 整体设计

我们设计并实现了五级流水结构的 CPU，对每条指令的处理分为 IF、ID、EXE、MEM、WB 五个阶段。采用 25M 时钟，每个时钟周期流水线的每一个阶段完成一条指令的一部分，不同阶段并行完成不同指令的不同部分。同时每两个阶段之间均有一个段间锁存器，用与接收上一阶段的信号并在下一个时钟上升沿到来时传递到下一阶段。

流水线五个阶段的功能与所占用的资源如下：

IF：根据输入的 PC 值从内存中取出指令。在执行写入命令时，还需要根据 PC 值向内存中写入用户指令。占用资源：IM、PC、总线

ID：根据 IF 阶段读取的指令进行译码，从寄存器堆中读出所需寄存器的值。占用资源：寄存器组

EXE：根据 ID 阶段生成的控制信号、操作数和操作符进行计算，将结果传递到下一阶段。占用资源：ALU

MEM：根据 ID 阶段生成的控制信号执行写入内存和读取内存的操作，在实现时还需考虑串口的读写与 VGA/Keyboard 的读写访问。占用资源：DM、总线

WB：根据控制信号执行写回寄存器的操作。占用资源：寄存器组

3.1.2 数据通路

我们设计的数据通路见 Figure 3.1.

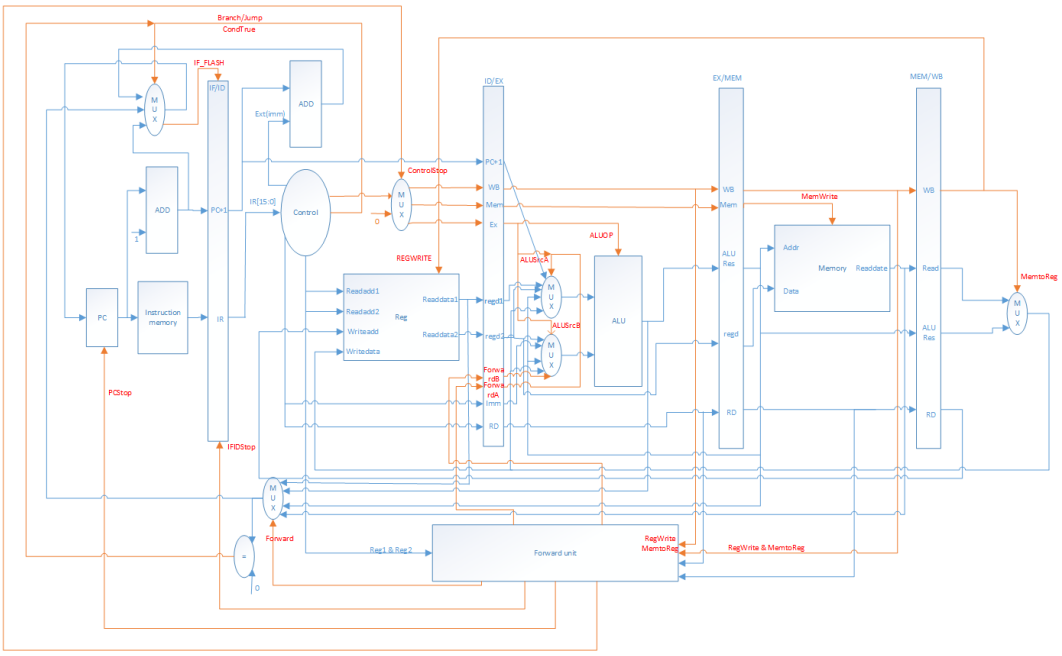


FIGURE 3.1: 数据通路.

在数据通路的设计上，我们基本遵循经典的五级流水线结构，但是在其基础之上有了一些变化和改进。

我们将处理数据冲突的模块全部放在了 Forward Unit 中。既可以将在 EXE、MEM 的计算结果流回到下一条指令的 EXE 阶段，同时也可以对访存操作的数据冲突进行插气泡处理。Forward Unit 模块还可以处理跳转指令的数据冲突，这样整体结构更加简洁。

我们新增 PC 模块，同于计算下一周期取指的 PC 值，根据跳转信号进行计算，可以解决控制冲突。具体实现见冲突处理部分。

3.1.3 控制信号

在 ID 阶段译码的过程中，会产生很多控制信号，针对我们所需实现的 30 条指令的指令集，我们设计的控制信号为：

ALUOP：ALU 运算器的操作符，包括的类型有 ADD（加法）、SUB（减法）、ASSIGNA（赋操作数 A 的值）、ASSIGNB（赋操作数 B 的值）、AND（与）、OR（或）、SLL（逻辑左移）、SRA（算数右移）、EQUAL（判断相等）、LESS（判断小于）、EMPTY（无操作）。

SRCREGA：读取第一个寄存器值的控制使能。1 表示 IR[10:8]，0 表示特殊寄存器。

SRCREGB：读取第二个寄存器值的控制使能。1 表示 IR[10:8]，0 表示 IR[7:5]。

REGDST：写回寄存器编号的控制使能。00 表示特殊寄存器，01 表示 IR[10:8]，10 表示 IR[7:5]，11 表示 IR[4:2]。

ALUSRCA：ALU 运算器第一个操作数的选择信号。00 表示 Reg[IR[10:8]]，01 表示 Reg[IR[7:5]]，10 表示 EPC。

ALUSRCB：ALU 运算器第二个操作数的选择信号。1 表示 reg[IR[7:5]]，0 表示 extend(imm)。

EXTOP：立即数扩展方式。0 表示符号扩展，1 表示零扩展。

MEMTOREG：读取内存并写回寄存器使能。

REGWRITE：写回寄存器使能。

MEMWRITE：内存写使能。

BRANCH：B 指令跳转信号。00 表示无 B 型跳转，10 表示无条件跳转，01 表示不等条件跳转，11 表示相等条件跳转。

JUMP：J 指令跳转使能。

对于每条指令的控制信号，见表 3.1—3.5，其中的'x'表示没有使用。

TABLE 3.1: Control Signals

	ADDIU	ADDIU3	ADDSP	ADDU	AND	B
ALUOP	ADD	ADD	ADD	ADD	AND	EMPTY
SRCREGA	1	1	0	1	1	x
SRCREGB	x	x	x	0	0	x
REGDST	01	10	00	11	01	x
ALUSRCA	00	00	00	00	00	x
ALUSRCB	0	0	0	1	1	x
EXTOP	0	0	0	x	x	0
MEMTOREG	0	0	0	0	0	0
REGWRITE	1	1	1	1	1	0
MEMWRITE	0	0	0	0	0	0
BRANCH	00	00	00	00	00	10
JUMP	0	0	0	0	0	0

TABLE 3.2: Control Signals

	BEQZ	BNEZ	BTEQZ	CMP	CMPI	JALR
ALUOP	EMPTY	EMPTY	EMPTY	EQUAL	EQUAL	ASSIGNA
SRCREGA	1	1	0	1	1	1
SRCREGB	x	x	x	0	x	x
REGDST	x	x	x	00	00	00
ALUSRCA	x	x	x	00	00	10
ALUSRCB	x	x	x	1	0	x
EXTOP	0	0	0	x	0	x
MEMTOREG	0	0	0	0	0	0
REGWRITE	0	0	0	1	1	1
MEMWRITE	0	0	0	0	0	0
BRANCH	11	01	11	00	00	00
JUMP	0	0	0	0	0	1

TABLE 3.3: Control Signals

	JR	JRRA	LI	LW	LW_SP	MFIH
ALUOP	EMPTY	EMPTY	ASSIGNB	ADD	ADD	ASSIGNA
SRCREGA	1	0	x	1	0	0
SRCREGB	x	x	x	x	x	x
REGDST	x	x	01	10	01	01
ALUSRCA	x	x	x	00	00	00
ALUSRCB	x	x	0	0	0	x
EXTOP	x	x	1	0	0	x
MEMTOREG	0	0	0	1	1	0
REGWRITE	0	0	1	1	1	1
MEMWRITE	0	0	0	0	0	0
BRANCH	00	00	00	00	00	00
JUMP	1	1	0	0	0	0

TABLE 3.4: Control Signals

	MFPC	MOVE	MTIH	MTSP	NOP	OR
ALUOP	ASSIGNA	ASSIGNA	ASSIGNA	ASSIGNA	EMPTY	OR
SRCREGA	x	x	1	x	x	1
SRCREGB	x	0	x	0	x	0
REGDST	01	01	00	00	x	01
ALUSRCA	10	01	00	01	x	00
ALUSRCB	x	x	x	x	x	1
EXTOP	x	x	x	x	x	x
MEMTOREG	0	0	0	0	0	0
REGWRITE	1	1	1	1	0	1
MEMWRITE	0	0	0	0	0	0
BRANCH	00	00	00	00	00	00
JUMP	0	0	0	0	0	0

TABLE 3.5: Control Signals

	SLL	SLTI	SRA	SUBU	SW	SW_SP
ALUOP	SLL	LESS	SRA	SUB	ADD	ADD
SRCREGA	x	1	x	1	1	0
SRCREGB	0	x	0	0	0	1
REGDST	01	00	01	11	x	x
ALUSRCA	01	00	01	00	00	00
ALUSRCB	0	0	0	1	0	0
EXTOP	1	0	1	x	0	0
MEMTOREG	0	0	0	0	0	0
REGWRITE	1	1	1	1	0	0
MEMWRITE	0	0	0	0	1	1
BRANCH	00	00	00	00	00	00
JUMP	0	0	0	0	0	0

除了以上通过译码器产生的控制信号外，还有 FORWARD UNIT 产生的冲突处理信号，我们将在下一节详细说明。

3.1.4 冲突处理

结构冲突

我们采用指令与数据分离存储的方式来避免结构冲突问题。用 RAM1 和 RAM2 分别存储数据和指令。但在监控程序功能中有一个 A 指令需要向指令存储器中写入用户指令，这时，我们通过在 MEM 阶段判断写入地址是否为指令内存地址区间，并产生控制信号 IFWE。在 IF 阶段如果接收到信号 IFWE，则将流水线暂停一个周期用来写入指令。

数据冲突

我们在这里仅讨论两种不涉及跳转的数据冲突，两种冲突分别为涉及访存和不涉及访存的冲突。比如：

Example1: ADDU R1 R2 R3; SLL R3 R3 0x00

Example2: LW R1 R3 0x00; SLL R3 R3 0x00

数据冲突产生原因是前一条指令或者前两条指令需要写回寄存器，而当前的指令又要访问寄存器中的值。这时我们通过增加旁路的方式将之前的计算结果传输到当前的操作数上。

在 FORWARD UNIT 中，通过判断上一条指令（或上两条指令）的写回寄存器编号与当前目的寄存器编号是否相等来判断是否发生数据冲突。对于涉及访存的数据冲突，我们需要插入气泡等待一个周期，使得上一条指令 MEM 阶段执行完毕取出内存数之后，再参与下一条指令的运算。FORWARD UNIT 产生的控制信号为：

FORWARDA：ALU 第一个操作数选择信号。00 表示无冲突，使用 ID 阶段读取的寄存器的值；01 表示与上一条指令发生冲突，选择 ALU 的结果；10 表示与上两条指令发生冲突，选择 MEM 的结果

FORWARDB：ALU 第二个操作数选择信号。与 FORWARDA 类似。

PCSTOP，IFIDSTOP，CONTROLSTOP：插入气泡的控制信号。

控制冲突

控制冲突是由于 B 指令与 J 指令而产生的。我们将跳转地址的计算放在 ID 阶段执行。这样，在控制器译码结束后，可以直接计算出跳转后的 PC 值，故这种做法不需要分支预测，可以提高速度。另外，我们采用打开延迟槽的策略，对跳转指令的后一条指令继续执行。

在实际的冲突中，控制冲突往往与数据冲突结合在一起，比如：

Example3: ADDU R1 R2 R3; JR R3;

Example4: LW R1 R3 0x00; BEQZ R3 0x10;

在以上的两个例子中，既发生了控制冲突，同时也存在数据冲突。我们同样使用增加旁路的办法，唯一的不同在于旁路的信号需要传送到 ID 阶段进行计算。FORWARD UNIT 模块同样可以产生数据冲突的信号，用于选择跳转所需的寄存器的值。在 PC 模块中，根据跳转信号与冲突选择信号选取下一条指令的 PC 值。

3.2 寄存器堆

寄存器堆用于存放所有寄存器的值，用于在 ID 阶段读取寄存器值与 WB 阶段写回寄存器的值。读寄存器值为组合逻辑，在信号稳定之前读取的值被锁在 ID_EXE 段间的锁存器中，故不会对后面的结果产生影响。写寄存器值为时序逻辑，必须等待信号稳定后才能写回。由于 MEM_WB 段间锁存器在上升沿触发，故在下降沿进行写回，此时信号已经稳定。

我们将 R0-R7 这八个通用寄存器放在寄存器堆中，同时还将 SP、RA、IH、T 四个系统寄存器也放在寄存器堆中，以方便处理。这样，寄存器的编号需要从三位扩展为四位。各寄存器的编号见表 3.6.

TABLE 3.6: Register Cluster

符号	含义	编号
R0	通用寄存器	0000
R1	通用寄存器	0001
R2	通用寄存器	0010
R3	通用寄存器	0011
R4	通用寄存器	0100
R5	通用寄存器	0101
R6	通用寄存器	0110
R7	通用寄存器	0111
SP	栈顶指针寄存器	1001
T	T 标志寄存器	1010
IH	中断寄存器	1011
RA	返回值寄存器	1100

3.3 存储器

3.3.1 SRAM 存储器

我们使用实验平台上的两块 SRAM 芯片作为计算机的内存储器。

由于流水线 CPU 在 IF 阶段和 MEM 阶段都需要访问内存，而实验平台的 SRAM 芯片只提供单端口的访问，因此为了提高流水线效率、简化设计，我们将数据与指令分开存储在两块芯片中，将 RAM1 作为数据存储器，RAM2 作为指令存储器。这样 IF 阶段 CPU 访问指令存储器 RAM2，MEM 阶段 CPU 访问数据存储器 RAM1，两者互相独立，可以同时进行。

RAM1、RAM2 的地址线均为 18 位，而 16 位计算机只需 16 位的地址空间，因此两块 RAM 的地址线高两位始终置零，只使用后 16 位地址线。地址空间的划分按照监控程序的要求，如下表所示：

TABLE 3.7: 地址空间划分

功能区	地址段	说明
系统程序区	0x0000 ~ 0x1FFF	存放监控程序 1
	0x2000 ~ 0x3FFF	存放监控程序 2
用户程序区	0x4000 ~ 0x7FFF	存放用户程序
系统数据区	0x8000 ~ 0x8EFF	监控程序使用的数据区
数据端口/命令端口 1	0xBF00 ~ 0xBF01	PS2、VGA 的端口
数据端口/命令端口 2	0xBF02 ~ 0xBF03	串口的端口
保留端口	0xBF04 ~ 0xBF0F	保留
系统堆栈区	0xBF10 ~ 0xBFFF	用于系统堆栈
用户数据区	0xC000 ~ 0xFFFF	用户程序使用的数据区

以上地址空间中，0x0000 ~ 0x7FFF 均为指令，存放在 RAM2 中；0x8000 ~ 0xFFFF 均为数据，存放在 RAM1 中。

因为实现了多道程序，我们在计算机中同时运行两套监控程序，监控程序分别存放在 0x0000 ~ 0x1FFF 和 0x2000 ~ 0x3FFF 空间，第一个监控程序使用第一组数据/命令端口，通过键盘输入、VGA 输出，第二个监控程序使用第二组数据/命令端口，在 PC 上通过终端程序输入输出。详见本章第 6 节。

SRAM 的访问时序如下图所示：

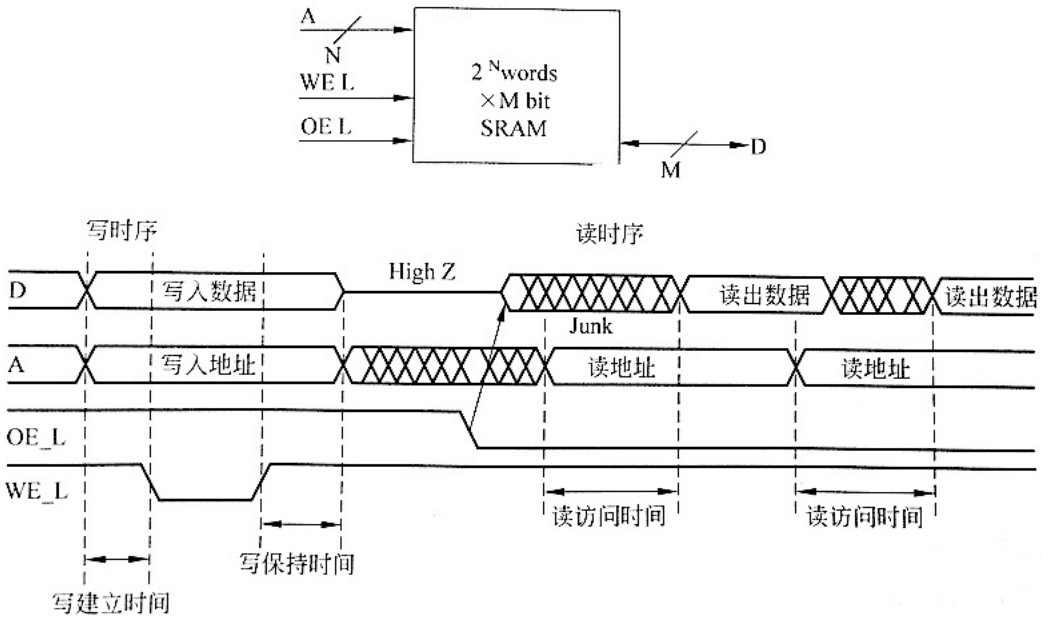


FIGURE 3.2: SRAM 访问时序

对于写操作，事先对地址线、数据线赋好值，然后将 WE 使能拉低，保持一段时间之后拉高，就完成了写入操作。

对于读操作，保持 OE 使能为低，WE 使能为高，对数据线赋高阻，地址线赋为要读取的地址，经过一段延迟时间之后数据线稳定，可以读出数据。

实验使用的 SRAM 芯片的建立时间、保持时间、读数据延迟等主要参数都在 10ns 左右。我们在实验中，每个流水段由上升沿驱动，在上升沿根据当前指令的读写情况进行对数据线、地址线赋好值，在下降沿给使能信号赋值，读操作拉低 OE 使能，写操作拉低 WE 使能，然后在下一个上升沿到来时拉高。由于信号不能同时被时钟上升沿、下降沿驱动，实际上是根据状态直接把时钟信号赋给使能信号，而不再使用时序逻辑。读操作时，后面一个阶段在下一时钟周期直接从 RAM1 或 RAM2 的数据线中获取读出的值。

指令寄存器 RAM2 在 IF 阶段访问，在 CPU 运行时一般只需要进行读操作。但是有两个特殊情况，一是监控程序的 A 命令，用户输入程序需要写入指令寄存器 RAM2，二是 boot 阶段，系统从 Flash 中读取数据，写入 RAM2 中。IF 阶段默认执行读操作，对于第一种情况，执行的指令还是 SW 指令，在 MEM 阶段发现需要写入 RAM2 控制的地址空间，则给 IF 段发出一个写信号，IF 阶段接收到信号后改为进行写操作，当前周期输出 NOP，即插入一个气泡，所有阶段暂停一个周期，下一个周期再重新执行原来要执行的指令。对于第二种情况，在 boot 阶段 Flash 控制器给 IF 段一个写信号，IF 同样改为写操作，输出 NOP，暂停一个周期。

一个小技巧用于解决监控程序的 U 命令，即反汇编，需要在 MEM 阶段读指令寄存器 RAM2。正常情况下 IF 阶段访问指令寄存器 RAM2，MEM 阶段只能访问数

据寄存器 RAM1，无法读取指令。为了解决这个问题，我们发现写指令寄存器的操作都是执行到了 MEM 阶段，然后返回 IF 阶段暂停当前周期，写入指令。这时 MEM 阶段可以同时把写入 RAM2 的数据也写入一份到 RAM1，这一操作在电路中完全并行，互相毫不影响。这样在 U 命令读取指令时，就无需真的再访问 RAM2 读指令，直接从 RAM1 中读取我们备份的指令就可以了。

3.3.2 Flash 存储器

我们将监控程序写入 Flash 存储器中，由于 Flash 断电数据不丢失，以避免每次写入监控程序的麻烦。

每次开机之后进行 boot 阶段，将 Flash 存储器中的数据依次读出，写入 RAM2 的对应位置。由于此时 RAM2 执行写操作，流水线暂停直到写入完毕。

Flash 的写入使用 Flash& RAM 软件写入即可，读操作需要进行一些时序的操作。具体为：首先写入操作码 0xFF，地址任意，即设置写使能 WE 为 0，将数据线置为 00FF，下一个周期将 WE 置为 1，数据已经写入，Flash 进入读模式。读操作需要置读使能 OE 为 0，地址线赋为要读取的地址，数据线赋为高阻，经过一段时间延迟可以读出数据。由于 Flash 的延迟相对比较大，我们使用的是 25MHz 时钟，一个时钟周期内 Flash 读出数据还不稳定，因此两个周期读一个数据。另外要注意的是，Flash 的编址方式和 RAM1、RAM2 不同，使用按字节编址。

3.4 中断处理

我们实现了四种类型的中断，其中包括两种硬件中断 (ESC 中断：返回到中断 PC；Control C 中断：返回到监控程序)，软件中断，时钟中断。对这四种中断分别介绍如下。

3.4.1 ESC 硬件中断

ESC 硬件中断是在用户程序运行时，键盘摁下 ESC 键产生的中断，在监控程序运行时无效。ESC 中断发生时，调用中断处理程序输出中断号，并返回到发生中断的指令继续执行。

这部分硬件中断会复用监控程序的 delint 中断处理代码，在中断处理程序中需要用到中断时的 PC 以及中断号，所以需要在发生中断时通过硬件来保存 PC 与中断号，并跳到中断处理程序。在 IF_ID 的段间锁存器中加入状态机，来处理 ESC 中断。状态机见 Figure 3.3.

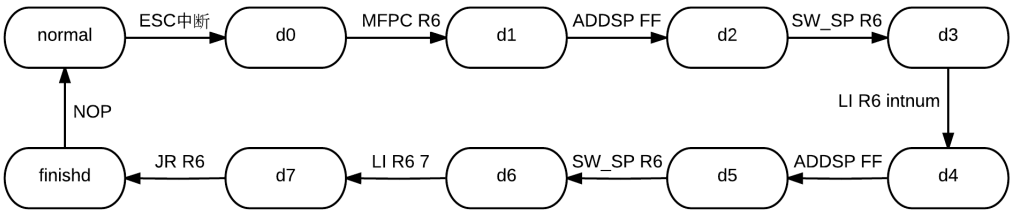


FIGURE 3.3: ESC 硬件中断状态机.

其中箭头上的指令为每个状态机下输出的指令，用来向栈中存储 PC 值和中断号。保存完毕后，跳到中断处理程序执行，同时状态回到 normal。

3.4.2 Control C 硬件中断

ESC 硬件中断的不足在于对于死循环的用户程序，中断发生后无法跳出死循环，而是继续回到中断发生的位置执行。我们希望仿照真正计算机上的 Control C 功能，可以实现跳出用户程序的功能。

Control C 中断的处理与 ESC 硬件中断类似，但是不需要再次返回中断时的 PC，而是直接跳到监控程序的 BEGIN 部分即可。所以处理过程比 ESC 更加简单，只需要在中断发生后将中断号入栈即可，不需要保存 PC 值。同时需要在监控程序中加入 Control C 中断的处理。状态机见 Figure 3.4.

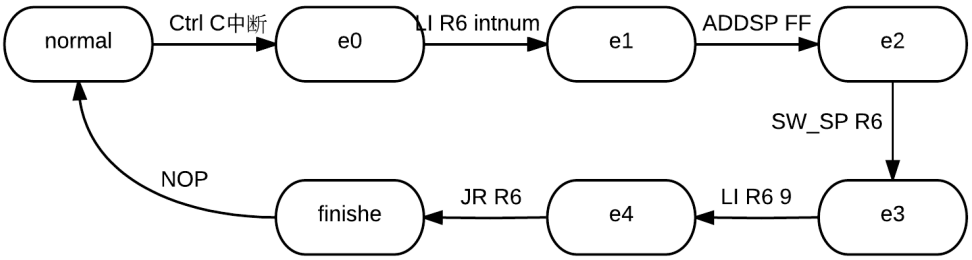


FIGURE 3.4: Control C 硬件中断状态机.

3.4.3 软件中断

软件中断的原理与 ESC 硬件中断一样，通过在控制器译码时产生软件中断的信号，传到 IF_ID 段间锁存器，同时将软件中断号同时传回，按照 ESC 的状态机处理软件中断。

3.4.4 时钟中断

时钟中断指计时时钟到一定的时间后，产生中断信号。我们做时钟中断的原因是为了之后的多道程序，多道程序需要分时执行两套监控程序，故需要有分时机制。多道程序的细节在后面讨论，这里仅讨论一下时钟中断的处理。

时钟中断的处理过程与 ESC 硬件中断基本一致，但是需要注意的一个地方是 ESC 硬件中断只会发生在用户程序中，这时我们就可以随意使用 R6、R7 的值 (因为用户程序不允许使用)。但是时钟中断可以发生在任何地方，在执行监控程序时也会有时钟中断，所以要首先保存 R6 的值，才可以执行后续的保存现场的指令。状态机见 Figure 3.5.

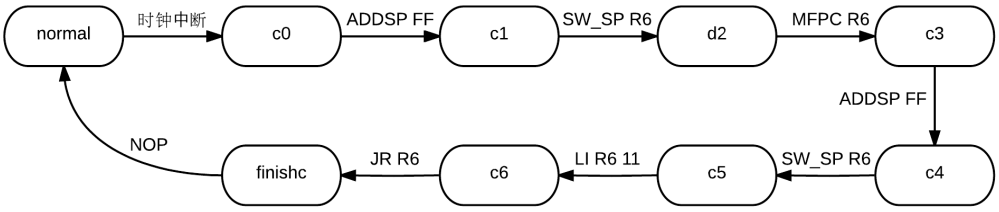


FIGURE 3.5: 时钟中断状态机.

3.5 I/O

我们实现了基础版本和扩展键盘 VGA 的两种 CPU。对于基础版本，通过串口与主机 term 通信。对于键盘 VGA 版本的 CPU，输入输出变成了键盘和 VGA，实现了一个完全独立的 VGA 版的 term 软件。此版本的 term 软件实现了主机的 term 的全部功能，包括 A 指令输入汇编指令、G 指令执行、U 指令查看反汇编的指令、D 指令查看内存、R 指令查看寄存器的值。

3.5.1 串口输入输出

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。串口通信功能已经在 CPLD 中实现，所以我们仅需要根据串口的访问时序读写总线即可。

我们通过阅读监控程序，了解到读写串口是通过 LW/SW 指令访问 BF00 及 BF01 地址实现的 (访问 BF01 以判断串口是否可读或可写)。所以我们在 MEM 阶段加入了判断，根据读写地址是否为 BF00 及 BF01 判断是否为串口访问。对于 BF00，我们只需在数据总线上赋予或读取相应的值，然后将串口通信的信号 wrn 或 rdn

拉低即可。对于 BF01，我们只需在数据总线的第 0 位赋予 tsre&tbre，第一位赋予 data_ready，从而判断串口状态。

和普通的串口芯片 8251 不同，教学机上的 UART 没有设计缓冲，也不具备可编程的性质，所以 CPLD 实现的串口功能有所欠缺。在基础功能的实现中，串口的访问没有遇到很大的问题，但是在后面的多道程序中，如果设置时钟中断的切换时间较长，那么串口则不能保存数据，所以如果在读写过程中发生时钟中断，很容易造成丢失数据，使得 term 无法正常工作。后面一节会详细讨论多道程序的实现。

3.5.2 键盘输入

键盘主要状态机有 13 个状态，分别为 delay, start, d0, d1, d2, d3, d4, d5, d6, d7, parity, stop, finish，称为 R 状态机整个状态机使用键盘时钟驱动，根据实验书上内容，当数据线上有数据读入时，首先会将数据线拉低，然后时钟每跳变一次，数据线上将会传入下一个值，将数据按位依次存入到 code 中，由于键盘在串行传入数据时候可能存在数据出错，因此在接受完数据后需要判断数据校验位，如果数据存在问题，则将数据丢弃并回到起始状态，重新串行读入数据。根据这个状态机，我们可以在状态为 stop 时接受到一个完整数据，在下一部分将对接受到的数据进行处理，使用一个数据通道和一个使能通道传入给键盘状态机模块。具体的状态机见 Figure 3.6.

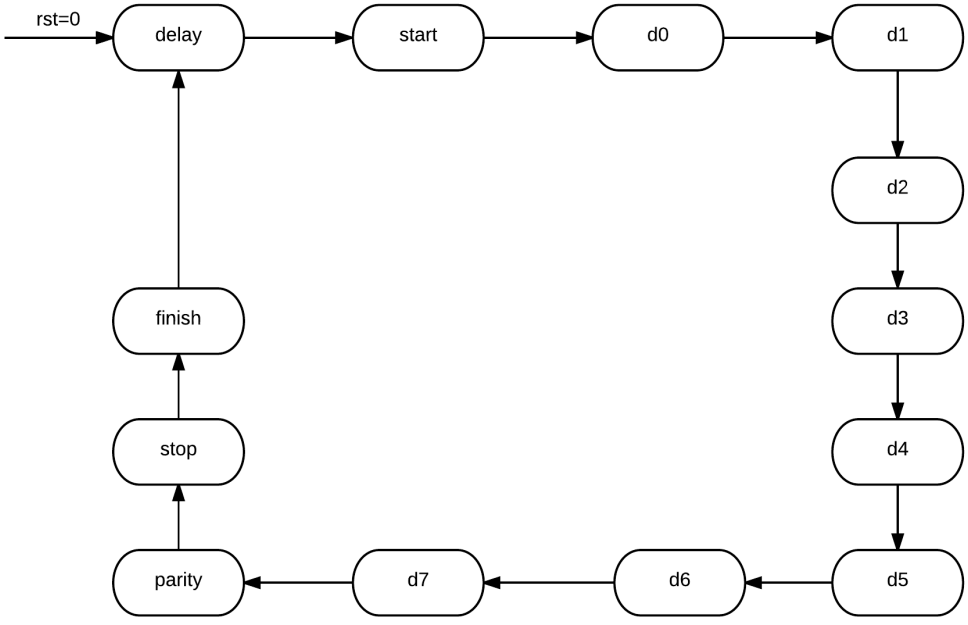


FIGURE 3.6: 键盘

对于键盘输入的处理，有两个状态 move_state 和 wait_state，称为 S 状态机在上一部分中，已经能够获得串行读入的数据，键盘数据在正常按下后，会发送一个

编码，在放开键盘后，会依次发送一个断码 F0 和该键的编码，需要注意的是，如果按下一个键不放，在经过一个比较短的时间后，键盘将会连续地发送同样的编码。

根据这样的特点，设计了两个状态的状态机，根据读入的编码来选择状态转变，如果读入的编码为 F0，会进入 wait_state 而在进入 wait_state 时需要判断读入的编码，如果不是先前读入的键的编码，则回到 move_state。键盘模块传入给键盘上层模块的数据线使用当前存入的按下的键，即当按下一个键不放时，数据线上并行传入当前按下的键，放开键后，数据线全部赋为低电位。在本键盘中，为了能够尽量真实地模拟键盘，当在按下 A 键时再按下 B 键时，数据线上将赋值为 B 键。

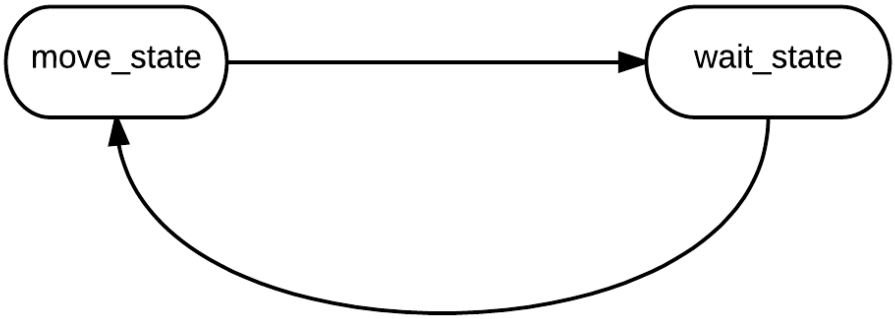


FIGURE 3.7: 键盘输入

键盘上层模块将根据使能位对数据进行读取处理。在前一章状态机中，在 start 状态机内，会将使能位拉低，而在 stop 状态机内，根据 S 状态机状态和当前键盘读入的数据判断是否将使能位拉高，在读入到一个普通编码（非断码，非 shift，ctrl）后使能位被拉高。这样做的好处在于可以真实地模拟键盘在长按一个键的情况，在这样的情况下，键盘上层可以根据使能位不断读入一串字符。

键盘模块中还会发出三种特殊信号。信号为是否按下 shfit，是否按下 ctrl，是否按下中断键（ESC 键），shift 键和 ctrl 键的处理方式相同，在按下 shift/ctrl 键时，对应的数据位为高，放开 shift/ctrl 键后，对应的数据位为低，上层模块根据 shift/ctrl 键来判断是否按下 shift 或 ctrl 键。中断按键处理方式类似于普通按键，在此不再赘述。

对于我们实现的 term 程序的 A、G、U、D、R 五条指令，通过判断缓冲区中的字符来进行发送。为了尽量不改变监控程序以及 CPU 的代码，我们让键盘仿照串口，通过 BF00 和 BF01 端口输入，同时增加使能信号 rdn、data_ready 来进行数据发送。同时将数据放到数据总线上，完成键盘输入功能。

3.5.3 汇编码转为机器码

我们在实现键盘 VGA 时，为了尽量模拟主机 term 程序的功能，加入了 A 指令输入汇编码的功能。由于监控程序的输入指令为机器码表示，所以我们需要在键盘给 CPU 发送数据之前将用户输入的汇编码转为机器码。通过 compiler 模块实现。

compiler 模块的输入为指令缓冲区，用断码表示汇编码的每个字符，输出为 16 位的机器码指令。我们加入了一部分的代码检错功能，对于不合法的汇编指令，翻译成的机器码全为 1，然后指示用户重新输入此条指令。基本做到了与 term 程序一样的汇编码输入功能。

3.5.4 VGA 输出

我们实现了使用 VGA 接口输出，在显示屏上显示计算机运行的信息。

仿照终端程序 Term，我们输出的界面是一个命令行的界面，即一个字符的阵列。我们 VGA 的输出使用分辨率为 640 × 480 像素的显示模式，故规定屏幕上显示 30 行、80 列字符，每个字符的显示大小为 16 × 8 像素。

为了实现命令行的界面，我们使用实验 FPGA 芯片提供的片内 BlockRam 作为显存。具体地，我们例化了两块显存，一块双端口 Ram，用于存取字符阵列信息，另一块是单端口 Rom，用于存取所有可见 ASCII 码的字体图片。Ram 的地址线 12 位（5 位表示行、7 位表示列）、数据线 16 位，存放字符。Rom 的地址线 14 位（7 位表示 ASCII、4 位表示行、3 位表示列）。

有了显存的帮助，尤其是有双端口 Ram 的情况下，我们只需要在每次有输入数据的时候，将数据写入 Ram，然后 VGA 在行、场信号扫描时，根据坐标从 Ram 中读出字符信息，再从 Rom 中读出该字符在当前位置的颜色，用 VGA 输出出来。这些过程都可以独立同时进行，而不会造成访存的冲突。

在硬件程序中还需要做一些命令行界面的控制，包括存储每一行有多少个字符，处理删除、回车、换行、换页等情况，以及显示光标。具体地，在硬件语言中需要维护一个数组记录当前每行的长度（即已有字符的个数，左对齐显示），以及维护当前在哪一行。这样有输入数据时，除了要往 Ram 里写入数据，还要增加当前行的长度。如果是删除操作，就减少当前行的长度。在显示时，不完全按照 Ram 中存放的数据，超过当前行长度后面的字符均不显示，所以删除操作也不需要抹去 Ram 中的存储。如果是换行操作，则需要更新当前行，即进入下一行。这里会有一个换页的情况，如果写到最后一行（第 30 行）换行，那么第一行就没有用了，返回到第一行继续写。为了显示的一致性，需要记录一个 offset，即目前存储中实际的第一行是哪一行。光标的显示，在当前行行末的一个字符交替显示黑白即可实现。

3.5.5 屏幕保护界面

我们将上一学期《数字逻辑设计》的课程项目经过简化后加入我们此次的计算机实验，作为屏幕保护界面呈现出来。具体就是在一段时间没有用户键盘的输入、CPU 也没有输出的情况下，显示屏进入另外一套 VGA 的控制，显示屏幕保护界面。

我们的屏幕保护界面内容是一系列小球围绕中央转动的动画。实现上，根据时间变化计算小球的坐标，然后绘制圆、直线，通过 VGA 显示。由于 640×480 像素的分辨率比较低，我们在实现上还应用了计算机图形学的知识，对画圆、画直线算法进行了消锯齿处理。

3.5.6 机器码转为汇编码

同样，为了模拟 term 程序的 U 指令反汇编的功能，我们加入了机器码转为汇编码输出的模块。由于硬件显示为时序逻辑，且延迟较难处理，我们把反汇编功能放到软件上实现。通过改写监控程序，将 16 位的机器码转为汇编码，然后逐字发送到 VGA 显示。

3.6 多道程序

我们实现了扩展功能中的多道程序。在同一台教学计算机上可以连接两个终端，同时运行各自的监控程序，并可以互不干扰的运行不相同的用户程序。

考虑到我们已经较好地实现了键盘 VGA 模拟的 term 程序，这样我们就可以将键盘 VGA 的 term 程序看做第二个终端。这样，教学计算机同时与电脑端的 term 通信，也与键盘 VGA 的 term 通信。多道程序的实现原理为分时机制，我们之前介绍过的时钟中断在这里派上用场。通过时钟中断，CPU 在两套监控程序中切换，这样就可以同时运行两套监控程序。

我们将地址单元进一步划分。

TABLE 3.8: Memory Allocate

	电脑端 term	键盘 VGA
监控程序	0x2000-0x3FFF	0x0000-0x1FFF
用户寄存器	0xBF10-0xBF15	0xBF16-0xBF1B
栈顶指针	0xBF50	0xBF60
保存现场缓存区	0xBF20-0xBF2F	0xBF30-0xBF3F

在实现多道程序时，我们遇到了很多问题，除了时钟中断外，在设计上也遇到了很多挑战，我们详细说明如下。

由于时钟中断会发生在监控程序中，所以我们在处理中断使用寄存器之前要首先将寄存器的值保存。所以在处理时钟中断时，我们不能使用没有保存过的寄存器。同时，在跳转到另一套监控程序之前，也需要将各个寄存器的值恢复，才能跳转。除了保存寄存器之外，还要保存当前的 PC、栈顶指针 SP 等。

由于两套监控程序的栈顶指针寄存器地址不一样，所以中断发生时要保存 SP，再恢复另一套监控程序的 SP。但是在指令集中我们没有发现读取当前 SP 到某个寄存器或地址的指令。这样就不能保存栈顶指针的值。所以我们需要扩展指令获取当前 SP 的值，然后进行保存。考虑到新创建指令 Assembler 不能编译的问题，我们复用 SLTUI 指令，并将其解析为将 SP 值保存到寄存器中。

在恢复现场时遇到的另一个问题为在中断处理的最后需要跳到另一套监控程序的 PC 值，这时候我们需要通过 JR 指令进行跳转，但是需要一个寄存器保存跳转的 PC。问题是所有寄存器需要保存之前的值，没有多余的寄存器来保存跳转的地址。为了解决这个问题，我们采用了一种简单的办法，就是打开跳转的延迟槽。首先我们将 R6 的值放在栈中，然后让 R6 保存跳转的 PC，通过 JR R6 实现跳转，然后在跳转的延迟槽中将栈中的 R6 值恢复。

Example5: SW_SP R6 0xFF; LI R6 PC; JR R6; LW_SP R6 0xFF

Chapter 4

模块划分与接口设计

本章介绍实验代码的模块划分，以及各个模块的接口设计。

4.1 IFetch

读写指令模块，控制指令寄存器 RAM2 的读写操作，时序逻辑。同时根据输入 PC 和暂停流水线信号输出下一条顺序执行指令的 PC 值。

TABLE 4.1: IFetch Interface

接口名称	类型	功能
clk	in	输入时钟
Ram2Addr	out	Ram2 地址线
Ram2Data	inout	Ram2 数据线
PC	in	输入 PC
PCInc	out	输出 PC
IR	out	输出指令
PCStop	in	插入气泡信号
IFWE	in	Ram2 写信号
IFData	in	写入 Ram2 的数据
IFAddr	in	写入 Ram2 的地址
Ram2EN	out	Ram2 的 EN 使能
Ram2WE	out	Ram2 的 WE 使能
Ram2OE	out	Ram2 的 OE 使能

4.2 IF_ID_REGISTER

IF_ID 段间锁存器，时序逻辑。除了传递信号外，还需处理中断，通过硬件方式保存现场并跳到中断处理程序。另外，还需要根据 Forward Unit 模块产生的插气泡信号和 MEM 产生的写 RAM2 信号进行停止流水线操作。

TABLE 4.2: IF_ID_REGISTER Interface

接口名称	类型	说明
clk	in	时钟信号
INPC	in	输入 PC 值
INIR	inout	输入指令值
INTERRUPT	in	中断信号，包括四种中断
IFIDSTOP	in	插入气泡信号
IF_FLUSH	in	跳转指令清除延迟槽信号
IF_WRITE_RAM2	in	Ram2 写指令，暂停流水线
OUTPC	out	输出 PC 值
OUTIR	out	输出指令值

4.3 Controller

CPU 控制器，组合逻辑。译码指令来产生控制信号。

TABLE 4.3: Controller Interface

接口名称	类型	说明
INSTRUCTION	in	需要译码的指令
SRCREG1	out	第一个源寄存器编号
SRCREG2	out	第二个源寄存器编号
TARGETREG	out	目的寄存器编号
EXTENDIMM	out	扩展后的立即数
ALUOP	out	ALU 操作符
ALUSRCA	out	ALU 第一个操作数选择信号
ALUSRCB	out	ALU 第二个操作数选择信号

MEMTOREG	out	读内存信号
REGWRITE	out	写寄存器信号
MEMWRITE	out	写内存信号
BRANCH	out	B 型跳转信号
JUMP	out	J 型跳转信号

4.4 RegisterCluster

寄存器堆，存放各个寄存器的值，用于读写寄存器。写寄存器为时序逻辑，读寄存器为组合逻辑。

TABLE 4.4: RegisterCluster Interface

接口名称	类型	说明
clk	in	时钟信号
rst	in	复位信号
SRCREG1	in	输入第一个源寄存器编号
SRCREG2	in	输入第二个源寄存器编号
TARGETREG	in	输入目的寄存器编号
REGWRITE	in	寄存器写使能
WRITEDATA	in	写回数据
REGDATA1	out	输出第一个源寄存器值
REGDATA2	out	输出第二个源寄存器值

4.5 PC

计算下一周期 PC 值的模块，组合逻辑。根据跳转信号判断和输出下一周期的跳转地址。为了兼顾 B 指令、J 指令的各种情况，输入多个可能的跳转地址，包括旁路信号。

TABLE 4.5: PC Interface

接口名称	类型	说明
CURRENTPC	in	当前 PC
LASTPC	in	上一周期的 PC
EXTENDIMM	in	B 指令的偏移值
BRANCH	in	B 指令跳转类型
JUMP	in	J 指令跳转
REG1data	in	寄存器值，对 B 指令可用于判断是否跳转，对 J 指令指定跳转地址
ALURES1	in	ALU 阶段旁路值
ALURES2	in	MEM 阶段保存的 ALU 结果
MEMDATA	in	MEM 阶段读取的内存值
FORWARD	in	数据冲突信号
NEXTPC	out	下一指令 PC 值
IF_FLUSH	out	清除延迟槽信号

4.6 ID_EX_REGISTER

ID_EXE 段间锁存器，时序逻辑。传递信号，还需根据 FORWARD UNIT 模块的插气泡信号暂停流水线。

TABLE 4.6: ID_EX_REGISTER Interface

接口名称	类型	说明
clk	in	时钟信号
CONTROLSTOP	in	暂停流水线信号
IN_MEMTOREG	in	输入读内存信号
IN_REGWRITE	in	输入写寄存器信号
IN_MEMWRITE	in	输入写内存信号
IN_ALUSRCA	in	输入 ALU 第一个操作数选择信号
IN_ALUSRCB	in	输入 ALU 第二个操作数选择信号
IN_ALUOP	in	输入 ALU 操作符
IN_FORWARDA	in	输入数据旁路选择信号 A

IN_FORWARDB	in	输入数据旁路选择信号 B
IN_EXTENDIMM	in	输入扩展的立即数
IN_TARGETREG	in	输入写回寄存器编号
IN_REGDATA1	in	输入第一个寄存器值
IN_REGDATA2	in	输入第二个寄存器值
IN_PC	in	输入 PC 值
OUT_MEMTOREG	out	输出读内存信号
OUT_REGWRITE	out	输出写寄存器信号
OUT_MEMWRITE	out	输出写内存信号
OUT_ALUSRCA	out	输出 ALU 第一个操作数选择信号
OUT_ALUSRCB	out	输出 ALU 第二个操作数选择信号
OUT_ALUOP	out	输出 ALU 运算符
OUT_FORWARDA	out	输出数据旁路选择信号 A
OUT_FORWARDB	out	输出数据旁路选择信号 B
OUT_EXTENDIMM	out	输出扩展的立即数
OUT_TARGETREG	out	输出写回寄存器编号
OUT_REGDATA1	out	输出第一个寄存器值
OUT_REGDATA2	out	输出第二个寄存器值
OUT_PC	out	输出 PC 值

4.7 ALU

ALU 运算器，组合逻辑。用于 EXE 阶段进行运算。

TABLE 4.7: ALU Interface

接口名称	类型	说明
REGDATA1	in	第一个寄存器的值
REGDATA2	in	第二个寄存器的值
EXTENDIMM	in	扩展的立即数
LASTALU	in	上一条指令 ALU 结果
LASTMEM	in	上两条指令 MEM 结果
forwardA	in	第一个运算数的数据冲突选择信号

forwardB	in	第二个运算数的数据冲突选择信号
ALUSRCA	in	第一个运算数的选择信号
ALUSRCB	in	第二个运算数的选择信号
ALUOP	in	ALU 运算符
PC	in	PC 值
ALURES	out	ALU 结果
REGDATA	out	选择的寄存器值，用于写到内存

4.8 EX_MEM_REGISTER

EXE_MEM 段间锁存器，时序逻辑。

TABLE 4.8: EX_MEM_REGISTER Interface

接口名称	类型	说明
clk	in	时钟信号
IN_MEMTOREG	in	输入读内存信号
IN_REGWRITE	in	输入写寄存器信号
IN_MEMWRITE	in	输入写内存信号
IN_ALURES	in	输入 ALU 计算结果
IN_REGDATA	in	输入 MEM 写入数据
IN_TARGETREG	in	输入目的寄存器编号
OUT_MEMTOREG	out	输出读内存信号
OUT_REGWRITE	out	输出写寄存器信号
OUT_MEMWRITE	out	输出写内存信号
OUT_ALURES	out	输出 ALU 计算结果
OUT_REGDATA	out	输出 MEM 写入数据
OUT_TARGETREG	out	输出目的寄存器编号

4.9 MEM

访存阶段模块，控制数据寄存器 Ram1 的读写，时序逻辑。此模块中还需根据读写地址判断是否为写入指令、读写串口、键盘 vga 读写等等。

TABLE 4.9: MEM Interface

接口名称	类型	说明
clk	in	输入时钟
Ram1Data	inout	Ram1 数据线
Ram1Addr	out	Ram1 地址线
Ram1OE	out	Ram1 的 OE 使能
Ram1WE	out	Ram1 的 WE 使能
Ram1EN	out	Ram1 的 EN 使能
data_ready	in	键盘的 data_ready 信号
tbre	in	VGA 模块的 tbre 信号
tsre	in	VGA 模块的 tsre 信号
rdn	out	给键盘的 rdn 控制信号
wrn	out	给 VGA 模块的 wrn 控制信号
uart_data_ready	in	串口的 data_ready 信号
uart_tbre	in	串口的 tbre 信号
uart_tsre	in	串口的 tsre 信号
uart_rdn	out	串口的 rdn 控制信号
uart_wrn	out	串口的 wrn 控制信号
MemData	in	输入的数据线
MemAddr	in	输入的地址线
MemWE	in	写信号
MemRE	in	读信号
IFWE	out	控制是否写 Ram2 的信号
IFData	out	给 Ram2 的数据
IFAddr	out	给 Ram2 的地址

4.10 MEM_WB_REGISTER

MEM_WE 段间锁存器，时序逻辑。

TABLE 4.10: MEM_WB_REGISTER Interface

接口名称	类型	说明
clk	in	时钟信号
IN_MEMTOREG	in	输入内存数据写入寄存器信号
IN_REGWRITE	in	输入写寄存器信号
ALURES	in	输入 ALU 计算结果
IN_TARGETREG	in	输入目的寄存器编号
READDATA	in	内存读出的数据
OUT_REGWRITE	out	输出写寄存器信号
OUT_TARGETREG	out	输出目的寄存器编号
WRITEDATA	out	写回数据

4.11 FORFARD_UNIT

数据冲突检测模块，组合逻辑。根据当前 ID 阶段指令的源寄存器编号与 EXE、MEM 阶段的目的寄存器编号是否相等及写回寄存器、读写内存等信号判断是否发生数据冲突及是否需要插气泡。

TABLE 4.11: FORFARD_UNIT Interface

接口名称	类型	说明
SRCREG1	in	ID 阶段源寄存器 1 编号
SRCREG2	in	ID 阶段源寄存器 2 编号
TARGETREGALU	in	ALU 阶段目的寄存器编号
TARGETREGMEM	in	MEM 阶段目的寄存器编号
REGWRITEALU	in	ALU 阶段写回寄存器信号
MEMTOREGALU	in	ALU 阶段读内存信号
REGWRITEMEM	in	MEM 阶段写回寄存器信号
MEMTOREGMEM	in	MEM 阶段读内存信号

FORWARDA	out	当前指令 ALU 操作数 1 数据冲突选择信号
FORWARDB	out	当前指令 ALU 操作数 2 数据冲突选择信号
FORWARD	out	当前指令跳转时数据冲突选择信号
PCSTOP	out	插入气泡
IFIDSTOP	out	插入气泡
CONTROLSTOP	out	插入气泡

4.12 Keyboard

用于键盘读入，通过状态机来接收通码、断码，并做奇偶校验。将得到的键盘输入输出到 keyboardCpuRam 模块进行处理。

TABLE 4.12: Keyboard Interface

接口名称	类型	说明
datain	in	键盘输入数据
clkin	in	键盘时钟
fclk	in	50M 时钟
interrupt	out	ESC 中断信号
interruptC	out	Control C 中断信号
keyboard_out	out	键盘输入结束信号
shiftstate	out	shift 组合键
ctrlstate	out	ctrl 组合键
word	out	键盘数据

4.13 KeyboardCpuRam

模拟 term 程序的输入部分，通过键盘的摁键设置状态机，实现 A、G、U、D、R 五条 term 程序指令，并发送给 CPU 模块。

TABLE 4.13: KeyboardCpuRam Interface

接口名称	类型	说明
clk50	in	50M 时钟
KeyBoardData	in	键盘输入到状态机的数据
DataReady	in	键盘输入至状态机的通知
CpuDataReady	out	状态机输入到主机
DataPath	inout	数据总线
CpuRdn	in	主机给状态机读信号
CpuEnable	in	主机给状态机是否能写
RamData	out	将 acsii 字符写入 ram，用于 VGA 显示
RamDataReady	out	dataready 给 ram
RamRdn	in	ram 给状态机的通知
shiftstat	in	shift 组合键信号
ctrlstate	in	ctrl 组合键信号

4.14 KeyboardStateMachine

键盘输入部分的 top 模块。用于与 CPU 和 VGA 进行通信，同时访问底层 keyboard 模块数据，保证数据的正确性。

TABLE 4.14: KeyboardStateMachine Interface

接口名称	类型	说明
datain	in	键盘输入数据
clkin	in	键盘时钟
fclk	in	50M 时钟
interrupt	out	ESC 中断信号
interruptC	out	Control C 中断信号
rdn	in	主机给状态机的读信号
CpuDataReady	out	键盘给主机可读信号
showData	inout	键盘给 CPU 的数据
KBData	out	键盘给 VGA 的数据
KBdata_ready	out	键盘给 VGA 可读信号

KBrdn	in	VGA 给键盘读使能

4.15 Compiler

汇编代码转为机器码的模块。当 term 执行 A 指令时，用户可以输入汇编码，键盘模块将整句的汇编码保存在缓冲区中，等待回车到来后将输入的汇编码转为机器码，并发送到 CPU 以写入 RAM2.

TABLE 4.15: Compiler Interface

接口名称	类型	说明
sendbuffer	in	输入指令缓冲区
instruc	out	16 位指令机器码表示

4.16 VGA_Controller

VGA 控制模块，控制计算机的 VGA 显示功能，主要任务是将需要显示的数据存入显存（一块 FPGA 片内 RAM），并控制 VGA 的显示。

TABLE 4.16: VGA_Controller Interface

接口名称	类型	说明
clk50	in	50M 时钟
rst	in	复位信号
DataIn	in	输入数据
data_ready	in	VGA 的 data_ready 信号
rdn	out	VGA 的 rdn 控制信号
hs	out	VGA 行同步信号
vs	out	VGA 场同步信号
r, g, b	out	VGA 输出的 RGB

4.17 VGA

VGA 模块，控制 VGA 输出，包括从一块 FPGA 片内 ROM 中读取字符图片并显示。

TABLE 4.17: VGA_Controller Interface

接口名称	类型	说明
clk50	in	50M 时钟
rst	in	复位信号
DataIn	in	输入数据
data_ready	in	VGA 的 data_ready 信号
hs	out	VGA 行同步信号
vs	out	VGA 场同步信号
r, g, b	out	VGA 输出的 RGB
row	out	字符行
column	out	字符列
data	in	一个像素的色彩值

4.18 VGA_Core_Ball

屏幕保护显示模块，控制在屏幕保护阶段的 VGA 输出。这一部分代码由作者上一学期《数字逻辑设计》课程的课程项目经过精简后得到。

TABLE 4.18: VGA_Core_Ball Interface

接口名称	类型	说明
clk	in	25M 时钟输入
reset	in	复位信号
r, g, b	out	输出的 RGB
vector_x	in	当前 VGA 输出的行
vector_y	in	当前 VGA 输出的列

num_c	in	小球个数
theta_c	in	小球角度
mtime	in	时间

4.19 Output_Adaptor

输出转换接口模块。VGA 的显示有来自 CPU 和来自键盘两个不同的来源，经过这个模块的处理整合后提供给 VGA 控制模块。

TABLE 4.19: Output_Adaptor Interface

接口名称	类型	说明
clk50	in	50M 时钟
CPUData	in	CPU 的数据
CPUwrn	in	CPU 的 wrn 信号
CPUtsre	out	CPU 的 tsre 信号
KBData	in	键盘的数据
KBdata_ready	in	键盘的 data_ready
KBrdn	out	键盘的 rdn 信号
DataOut	out	输出的数据
data_ready	out	输出的 data_ready 信号
rdn	in	rdn 控制信号

Bibliography

Arnold, A. S. et al. (1998). "A Simple Extended-Cavity Diode Laser". In: *Review of Scientific Instruments* 69.3, pp. 1236–1239. URL: http://link.aip.org/link/?RSI/69/1236/1.

Hawthorn, C. J., K. P. Weber, and R. E. Scholten (2001). "Littrow Configuration Tunable External Cavity Diode Laser with Fixed Direction Output Beam". In: *Review of Scientific Instruments* 72.12, pp. 4477–4479. URL: http://link.aip.org/link/?RSI/72/4477/1.

Wieman, Carl E. and Leo Hollberg (1991). "Using Diode Lasers for Atomic Physics". In: *Review of Scientific Instruments* 62.1, pp. 1–20. URL: http://link.aip.org/link/?RSI/62/1/1.