



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 9, 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

Abstract	i
1 实验目的	1
2 实验环境	2
2.1 硬件环境	2
2.1.1 FPGA 芯片	2
2.1.2 CPLD 芯片	3
2.1.3 SRAM 存储器	3
2.1.4 Flash 存储器	3
2.1.5 总线	3
2.1.6 外部接口	4
2.2 软件环境	4
2.2.1 FPGA 开发工具	4
2.2.2 THINPAD 软件包	4
3 实验设计	5
3.1 CPU 流水结构	5
3.1.1 整体设计	5
3.1.2 数据通路	5
3.1.3 控制信号	6
3.1.4 冲突处理	10
结构冲突	10
数据冲突	10
控制冲突	10

3.2	寄存器堆	11
3.3	存储器	12
3.3.1	SRAM 存储器	12
3.3.2	Flash 存储器	14
3.4	中断处理	14
3.4.1	ESC 硬件中断	14
3.4.2	Control C 硬件中断	15
3.4.3	软件中断	15
3.4.4	时钟中断	16
3.5	I/O	16
3.6	多道程序	16
Bibliography		17

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。

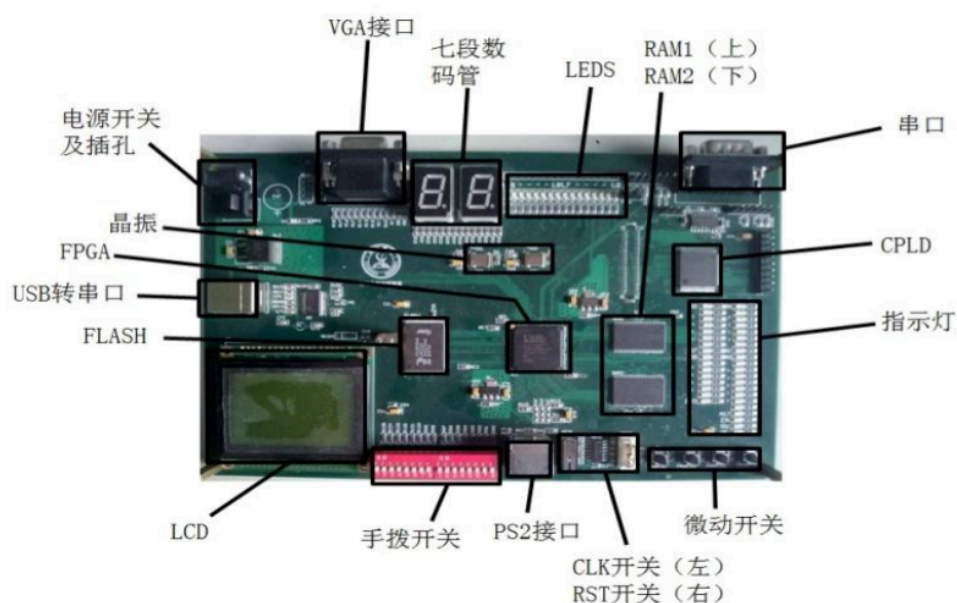
通过实验，可以加深对于计算机系统知识的理解，进一步理解和掌握流水线结构计算机的各部件组成和内部工作原理，掌握计算机外部设备输入输出的设计实现，培养硬件设计和调试的能力。

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 Random 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.

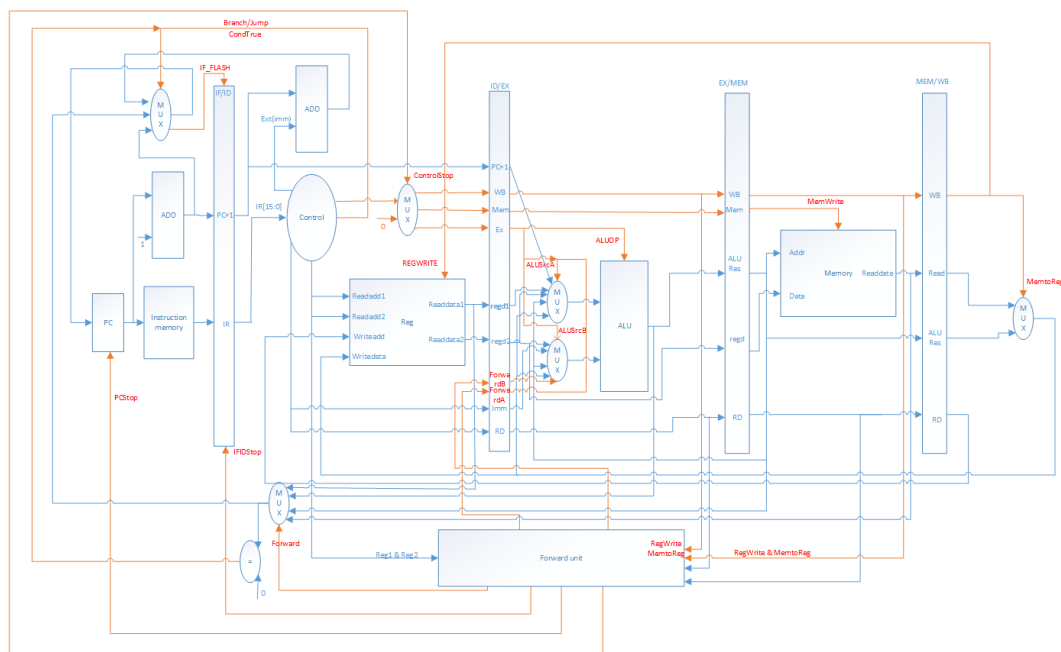


FIGURE 3.1: 数据通路.

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

FORWARDDB: 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，对跳转指令的下一条指令继续执行。

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:

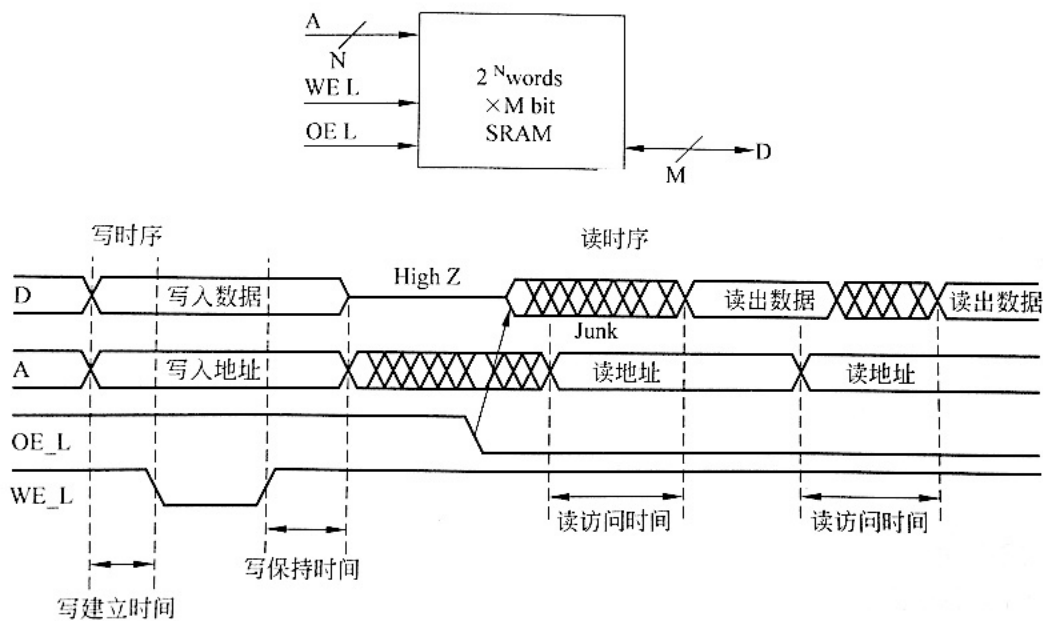


FIGURE 3.2: SRAM 访问时序

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。下面对这四种中断分别介绍。

3.4.1 ESC 硬件中断

ESC 硬件中断是通过在用户程序执行时，键盘摁下 ESC 键产生的中断，在监控程序执行时无效。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 3.2.

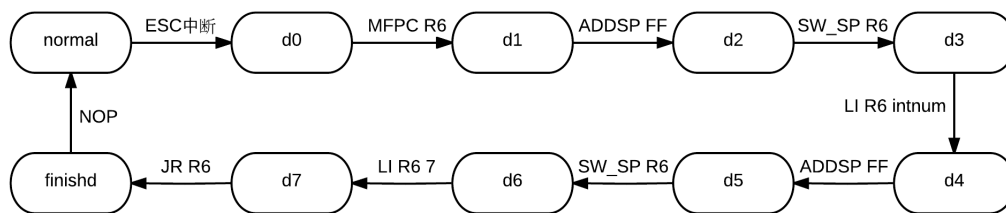


FIGURE 3.3: ESC 硬件中断状态机.

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 3.3.

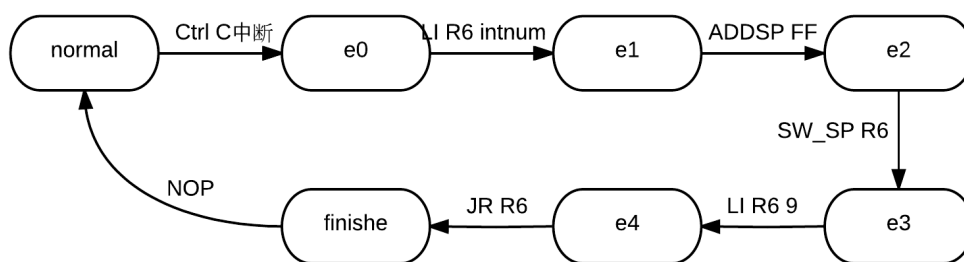


FIGURE 3.4: Control C 硬件中断状态机.

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 3.4.

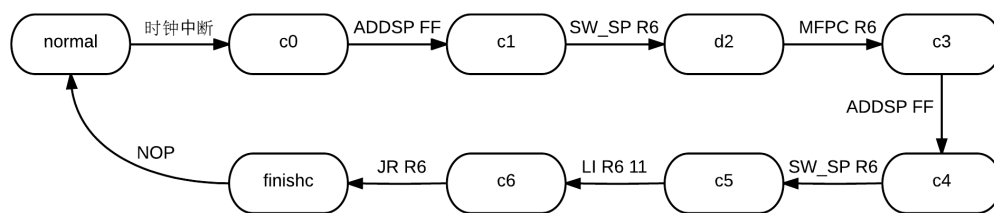


FIGURE 3.5: 时钟中断状态机.

3.5 I/O

3.6 多道程序

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>.