THCO MIPS 大作业
设计报告

丁弼原 李浩达 张敬卿

计 25

目录

一. 项目概述

计 25 班丁弼原、李浩达与张敬卿合作的 THCO MIPS 项目实现了如下功能：

1. THCO MIPS 流水线设计方案的基础功能；
2. 解决数据冲突与结构冲突；
3. CPU 运行级、精确异常处理与虚拟内存，即保护模式；
4. 外部时钟中断；
5. VGA 端口访问；
6. PS/2 键盘访问。

本 CPU 运行在 12.5MHz 的时钟频率上，可以运行监控程序，并通过测试程序。我们修改了监控程序，使其支持保护模式。修改部分集中在：修改中断处理程序，使之符合 THCO MIPS 中断处理框架；增加初始化代码，初始化并启用页表；修改 G 命令，使用户程序在用户级运行，不可执行特权指令，不可修改监控程序内存页面。经测试，修改后的保护模式版监控程序可以正常运行，并拦截了用户程序的各种非法访问操作。修改后的保护模式也能够对用户 INT 指令和外部中断即时响应，并完好地恢复被中断指令流。

专门开发的键盘驱动程序与 VGA 驱动程序可以成功运行在 THCO MIPS 上。用户可以使用键盘的上/下/左/右与 R/G/B 键控制 VGA 显示器上色块的位置和颜色。

二. 保护模式

2.1 概述

THCO MIPS 的保护模式由运行级、带运行级切换的精确异常处理和虚拟内存三个功能实现。

CPU 可以运行在两个运行级上（运行级保存在运行级寄存器 LF 中）：

- 0 级：也称为内核态。此时，可以执行一切指令，享有对所有地址空间的访问权，但这个访问权不能超过页属性的规范（不能写一个只读页，也不能执行在不可执行页上的指令）。

- 1 级：也称为用户态。此时，只能够执行普通指令，不能执行特权指令（特权指令在【2.2 对现有架构的扩充】中描述），仅享有对用户页面的访问，而且访问权不能超过页属性的规范。

切换运行级的唯一方式是依靠异常处理机制：当发生异常或者响应中断时，CPU 进入 0 级；当从中断服务例程返回时，CPU 可以降级到 1 级，也可以不降级（例如在 CPU 内核态时发生异常）。详细请参考【2.4 异常处理】中的描述。

通过以上机制，保护模式可以实现如下约束：

1. 保护内核代码数据免遭用户进程的访问和修改；
2. 不同用户进程享有独立的地址空间，不可互相访问；
3. 对只读数据内存实现写保护；
4. 禁止在不可执行的地址空间内执行指令；
5. 防止用户程序执行特权指令，危害其他进程与操作系统。

因此，保护模式可以使一个现代多任务操作系统运行在 THCO MIPS 上，并可以在硬件层面提供安全和保护机制。

2.2 对现有架构的扩充

现有 THCO MIPS 提供的功能有限，不足以运行保护模式，因此，我们对现有的架构进行了扩充。这种扩充在用户使用上体现在两个方面：新增的特殊寄存器和新增指令。

新增的特殊寄存器有：

1. 内核态堆栈的栈指针寄存器 KP；
2. 用户态堆栈的栈指针寄存器 UP；
3. 页表基地址寄存器 PD；
4. 修改了中断寄存器 IH 的使用方法，将 IH 的最低位作为允许分页的控制位，称之为 PE(Page Enable)；
5. 只能被异常处理器 EXCP 用长指令访问的寄存器 ER。该寄存器完全无法用 16 位指令访问到，只能被异常处理器使用，对用户完全不可见。

新增的指令几乎全部为特权指令（例外：MFSP 不是特权指令），这些指令只有当 CPU 在内核态时才可以执行，否则，会触发权限指令异常。新增的特权指令有：

1. 清 IH 最高位的关中断指令 CLI 与置 IH 最高位的开中断指令 STI；
2. 访问寄存器 KP 的指令 MTKP Rx 与 MFKP Rx，使用方法可以参考 MTIH Rx 与 MFIH Rx；
3. 访问寄存器 UP 的指令 MTUP Rx 与 MFUP Rx；
4. 设置寄存器 PD 的指令 MFPD Rx，该指令只是将页表基地址装入寄存器 PD，并不启用分页。若要启用分页，需要将 PE 位设为 1；
5. 将栈指针 SP 快速存入寄存器 UP 的指令 SWUP；
6. 将寄存器 KP 或者 UP 快速装入栈指针 SP 的指令 LDKP 和 LDUP。
7. 从中断服务例程返回的指令 ERET。

现有指令 MFIH 和 MTIH 因为涉及到控制中断和分页，因此被分类为特权指令。

为了和已有指令 MTSP Rx 配对，增加了普通指令 MFSP Rx。

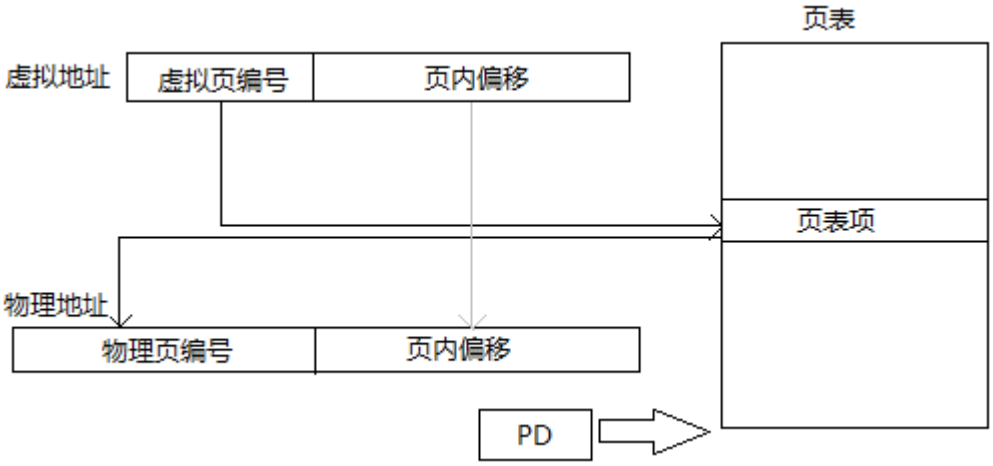
2.3 虚拟内存

虚拟内存是现代 CPU 必须支持的特性之一。虚拟内存不但极大地扩充了进程可用地址空间，而且提供了权限检查，保障了内核代码与用户进程的分离，提高了健壮性。

THCO MIPS 将 64KW 的地址空间按照 1KW 为一页，分成 64 页。每一页的属性用一个字来描述，这个字被称为页表项。64 个页表项构成的数组称为一个页表，页表可以起始于任何地址。页表的基地址被装入 PD 寄存器。一个虚拟地址的高 6 位为页表项的编号，低 10 位为在页中的偏移量。

THCO MIPS 使用两片 RAM 全部 512KW 的物理存储空间，因此，需要将一个 6 位的虚拟页编号翻译成 9 位的物理页编号。

在翻译虚拟地址时，CPU 将虚拟地址的高 6 位作为数组下标，在页表中查找对应的页表项，查找到后，检查访问权限，如果访问被允许，就将页表项中的 9 位物理页编号接在虚拟地址低 10 位前，构成一个 19 位的物理地址。



一个页表项的格式如图：

15 … 7	6 … 4	3	2	1	0
物理页编号	ZEROS	X	W	U	P

对于以上各位的描述如下：

1. P 位表示该页是否存在，P 为 1 时存在。如果该页不存在，一切其他位均没有意义，任何对该页的访问都会出错。
2. U 位表示该页是否为用户页，U 为 1 时为用户页，否则为内核页。CPU 在内核态时可以访问用户页和内核页，在用户态时只能访问用户页，不能访问内核页。
3. W 位表示该页是否可写，W 为 1 时可写。
4. X 位表示该页是否可执行，X 为 1 时可执行。强烈建议将可读写数据页和堆栈页设为不可执行，以防止缓冲区溢出攻击时在数据页和堆栈页上执行恶意代码。

当对页的访问不满足访问权限的时候，就会产生页异常。发生页异常时，CPU 传给中断服务例程两个额外参数（参见【2.4 异常处理】），分别是出错虚拟地址 Addr 与出错原因 Reason。其中，Reason 的格式如下：

位	描述
0	导致页异常的运行级
1	是否尝试写内存页
2	是否尝试执行内存页

2.4 异常处理

异常处理中的异常指的是广义的异常，包括指令在流水线的取指译码执行过程中触发的异常，INT 指令触发的软中断，外部中断到来，也包括执行 ERET 指令时触发的从中断服务程序的返回操作，地址翻译时的 TLB 缺失异常。

以上异常发生时，异常处理器 EXCP 从流水线接管控制权，根据异常类型的不同采取不同的行动，包括执行一些指令和改变一些 CPU 组件的状态，最终将控制权交还给流水线。

从触发异常的来源来看，可以将异常处理分为三类：

1. 异常准备，包括指令异常、INT 软中断和外部中断；
2. 从异常返回，由 ERET 指令触发；
3. TLB 缺失，地址翻译时由虚拟内存管理器 VM 触发。

这三种异常处理虽然有所不同，但是共用一套处理机制，只是执行的实际操作略有不同。THCO MIPS 的处理方式在【3.6 异常与中断处理数据通路】和【3.7 TLB 缺失处理数据通路】两节中进行了详细的描述。在这里，主要描述的是异常对于使用者（操作系统）的接口。

THCO MIPS 在运行过程中可能发生 4 种狭义上的指令异常，分别是：

1. 非法指令异常，如果从内存中读取的指令不符合任何一个支持的指令格式，就会导致该异常；
2. 特权指令异常，如果 CPU 运行在用户态，而执行了只有内核态可以执行的特权指令，就会导致该异常；
3. 页异常（虚拟内存请参考【2.3 虚拟内存】一节），如果非法地访问了一个内存页面，就会导致该异常。可能的非法操作包括：
    a) 访问不存在的页面（该页面对应的页表项的 P=0）；
    b) 运行在用户态时访问内核页面（U=0）；
    c) 尝试从不可执行的页面（X=0）取指令并执行；
    d) 尝试向不可写的页面（W=0）写入数据。
4. INT 指令触发的异常；

5. TLB 缺失异常，这一类异常在 CPU 内部就会被处理，不会通知操作系统，因此对用户来说是透明的。

如果在 THCO MIPS 的运行过程中，外部中断到来，而且 IH 寄存器的最高位为 1，就会对该外部中断进行响应，响应的流程与发生异常是相同的。

当异常或外部中断发生时，CPU 要中断当前正在执行的指令流，调用中断服务例程。中断服务例程处理完中断后，使用 ERET 指令退出中断服务，恢复被中断的指令流，重新执行被中断指令或者从下一条指令开始执行。

为了处理异常，中断服务程序必须对于所发生的异常有所了解，这些信息必须由 CPU 提供。因此，CPU 会在异常发生后，将必要的信息压入堆栈，再执行中断服务例程。因为涉及到入栈操作，就不得不考虑运行级问题。因为异常可能在任何时候发生，由各种原因引发，甚至本身就可能是因为栈指针指向了不可访问的内存而导致页异常。因此，在异常发生时，必须将异常描述信息压入可靠的堆栈。为了提供可靠性，CPU 在内核态和用户态下使用不同的堆栈，分别称之为内核栈和用户栈。为此，增加了两个特殊寄存器 KP 和 UP。当 CPU 从用户态进入内核态时，就把 SP 保存到用户态栈指针寄存器 UP，而将内核态栈指针 KP 装入 SP，从而完成换栈操作。同理，当 CPU 从中断服务例程返回时，如果需要返回到用户态，就把 UP 装入 SP，恢复用户栈。此时不需要保存内核栈，因为栈指针 SP 一定指向与刚进入内核态时相同的位置。

内核态与用户态使用不同堆栈的另一个好处是内核在处理中断时产生的临时数据不会泄露到用户空间，因此具有一定的保密性与隔离性。

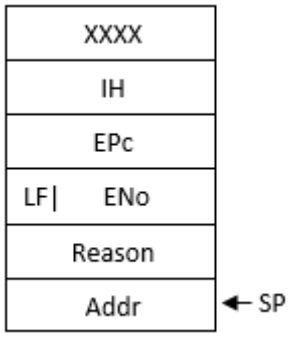
因此，操作系统必须保证内核栈永远是正确的，而用户程序则不需要也没有办法保证用户栈永远是正确的。

另外一个必须注意的问题是，在中断服务例程中，默认情况下是关闭中断的，也就是 IH 最高位为 0。这是为了防止不可控的中断嵌套。这并不是说在处理中断时必须屏蔽中断，中断服务例程可以在必要的时候打开中断，比如在处理阻塞系统调用时，系统调用函数（在中断服务例程中被调用）会将当前进程设为睡眠状态，重新调度进程，此时，必须保证时钟中断可以处理，否则就会死锁。而在处理完中断后，必须恢复 IH 寄存器，因此 IH 寄存器的值也需要保存在栈上。

解决了堆栈可用问题后，就可以考虑调用中断服务例程时栈上的异常信息格式。当刚刚进入中断服务例程时，SP 指向异常信息的最低地址处，异常信息一共由 5 个连续的字构成，从低地址到高地址依次为：

1. Addr (SP+0)，页异常时存放导致页异常的虚拟地址，其他异常不用；

2. Reason (SP+1)，页异常时存放导致页异常的操作，其他异常不用；
3. ENo (SP+2)，低 15 位为异常号，最高位用于表示发生异常时是处在内核态（值为 0）还是用户态（值为 1）；
4. EPc (SP+3)，被中断的指令地址，根据异常与中断类型的不同，可能为出错指令地址，也可能为被中断指令的下一指令地址；
5. IH (SP+4)，发生异常时 IH 寄存器的值，从中断服务例程返回时恢复 IH 寄存器。

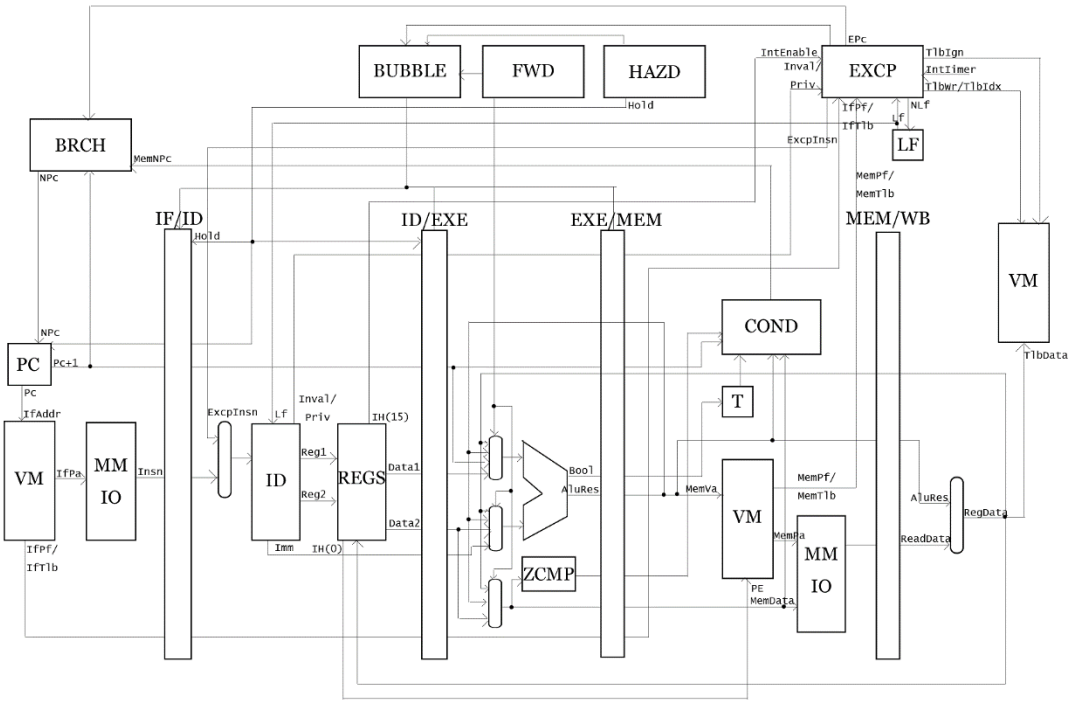


以下表格列举了目前支持的异常与中断：

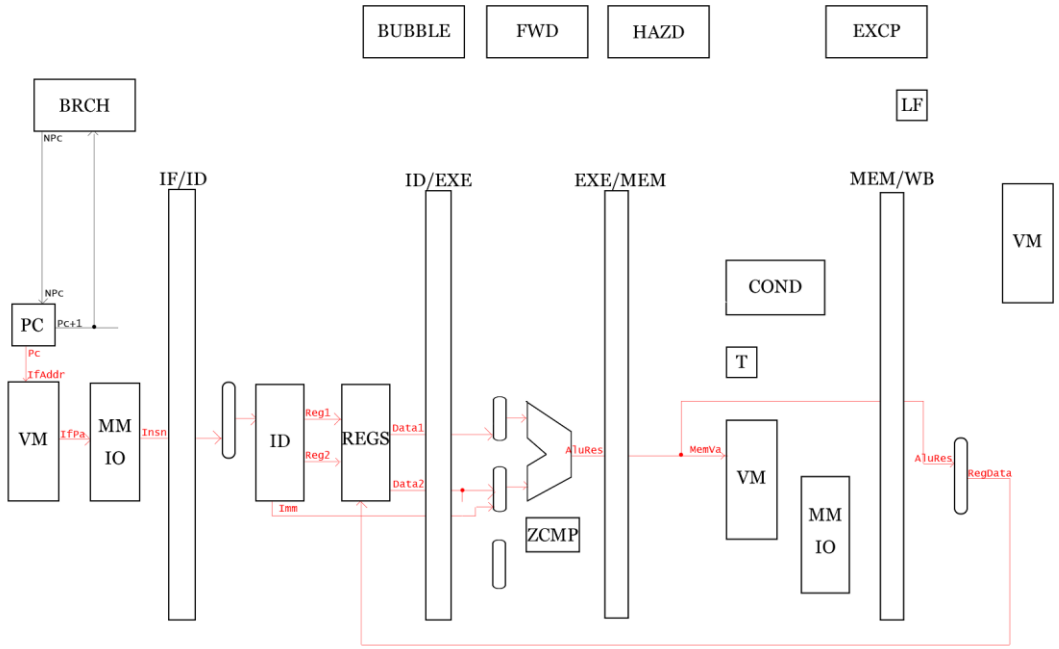
名称	类型	中断号	EPc 的值
非法指令	异常	0x00	导致异常的指令地址
页异常	异常	0x01	导致异常的指令地址
权限指令	异常	0x02	导致异常的指令地址
INT 指令	异常	0x03	下一条指令地址
时钟中断	中断	0x10	下一条指令地址

当中断服务例程处理完中断之后，必须将 SP 指针恢复到刚进入时的位置，而且在执行过程中不可以修改异常信息数据。在这些前提条件都满足下，执行 ERET 指令结束中断处理。

三. CPU 数据通路设计


BUBBLE
FWD
HAZD
Hold
IntEnable
Inval/
Priv
EXCP
EPc
TlbIgn
IntTimer
TlbWr/TlbIdx
IFpf/
IFTlb
NLf
LF
ExcpInsn
MemPf/
MemTlb
BRCH
MemNPc
NPc
IF/ID
ID/EXE
EXE/MEM
MEM/WB
VM
COND
T
TlbData
PC
Pc+1
Pc
IFAddr
VM
IFpa
MM
IO
Insn
ExcpInsn
Lf
Inval/
Priv
IH(15)
ID
Reg1
Reg2
REGS
Data1
Data2
Imm
IH(0)
Bool
AluRes
ZCMP
MemVa
VM
MemPf/
MemTlb
MemPa
MM
IO
PE
MemData
AluRes
RegData
ReadData

3.1 算术逻辑运算数据通路



绝大多数的指令需要从内存中获得（例外情况：在异常处理时，指令可能由异常处理器 EXCP 直接送入流水线。异常处理的具体流程请参考【3.6 异常与中断处理数据通路】）。因此，对于这绝大多数指令而言，指令进入流水线的第一步是从内存中获得指令内容。为了获得一条新的指令，我们需要知道该指令的物理地址，而程序计数器 PC 中存储的为指令的虚拟地址。因此，在取指令时，必须将虚拟地址翻译为物理地址。在翻译的过程中，可能出现 TLB 缺失和页异常（缺页、权限不够）两种异常。对于 TLB 缺失请参考【3.7 TLB 缺失处理数据通路】，对于页异常请参考【3.6 异常与中断处理数据通路】与【2.3 虚拟内存】。在本数据通路中，我们假定不会发生这两个异常。

在以上假设的前提下，程序计数器 PC 输出的指令虚拟地址被虚拟内存管理器 VM 翻译为物理地址。内存 I/O 控制器 MMIO 接受指令的物理地址，通过读取内存得到指令内容，并将该内容存入第一个流水线段寄存器 IF/ID。从而完成取指令操作。

当指令进入流水线第二阶段指令翻译阶段时，该指令首先进入指令翻译器 ID。在进入指令翻译器 ID 之前，需要穿过一个二路选择器。在异常状态下，二路选择器选择从异常处理器 EXCP 输出的指令，而不选择从 IF 阶段获得的指令。在正常状态下，二路选择器选择从 IF

阶段获得的指令，而不选择从异常处理器 EXCP 输出的指令，此时，EXCP 应该输出空指令。

指令翻译器 ID 首先识别该指令，如果不能识别该指令，则将非法指令信号置为 1。如果可以识别该指令，就判断当前 CPU 运行级下是否有权限执行该指令（CPU 运行级可以通过运行级寄存器 LF 连入指令翻译器 ID 的信号来得到，该信号在【3.6 异常与中断处理数据通路】中被画出），如果权限不足，则将权限指令信号置为 1。如果权限也满足，则根据指令的定义，输出相应的控制信号，两个源寄存器（可以为空寄存器，值永远为 0），一个目的寄存器（可以为空寄存器，忽略任何输入）和立即数。两个指令异常信号通过流水线段寄存器传到 MEM 阶段才由异常处理器 EXCP 处理。控制信号通过流水线段寄存器传到使用该控制信号的阶段，两个源寄存器直接输入寄存器堆 REGS，让 REGS 输出两个寄存器值，目的寄存器一直传到 WB 阶段才使用。

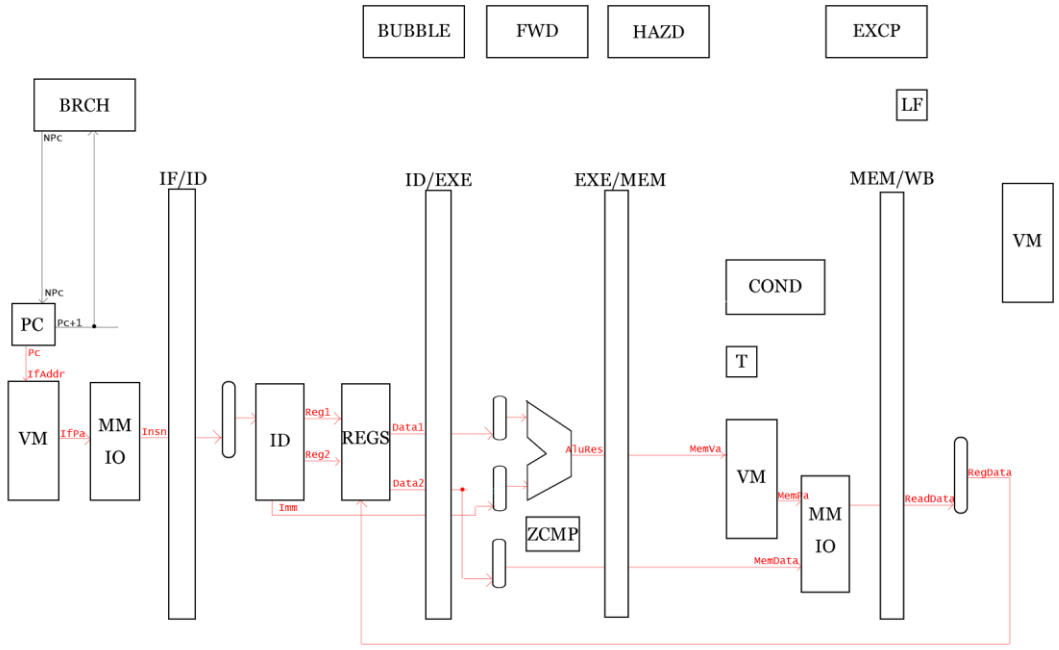
当指令进入执行阶段时，适当的操作数（可能为寄存器值，也可能为立即数）通过两个四路选择器进入 ALU，指令翻译器 ID 产生的 ALU 运算控制信号也传入 ALU，ALU 完成相应的运算并输出。对于一些指令，并不需要对操作数进行运算，通常，此时另一个操作数为 0，由 ALU 完成加法运算。

对于相等比较指令 BTEQZ 和 BTNEZ，ALU 的计算结果为一个布尔值，该信号没有在图中画出。在 MEM 阶段，该信号写入 T 寄存器。

当指令进入访存阶段时，算术逻辑运算指令并不进行访问内存或者跳转操作，因此在该阶段不做任何事情。

当指令进入写回阶段时，指令寄存器 ID 产生的目的寄存器控制信号和待写回的数据传入寄存器堆 REGS，完成数据的写入操作。为了避免结构冲突，寄存器堆在一个时钟周期的前半周期完成写操作（以时钟的上升沿标志一个时钟周期的开始，因此在时钟的下降沿触发写入操作），在一个时钟周期的后半周期完成读操作。

3.2 内存访问与 I/O 数据通路



内存控制与 I/O 数据通路与算术逻辑运算数据通路的差别主要集中在 EXE 与 MEM 阶段。

该类型的数据通路针对 LW/LW_SP/SW/SW_SP 等涉及到访问内存或者以访问内存的形式进行内存映射 I/O 操作的指令。对于这一类指令，被访问的内存地址为第一寄存器加立即数，而待写入的数据来自第二寄存器，或者取出的数据存在第二寄存器。因此，在 EXE 阶段的三个选择器中有两个与【3.1 算术逻辑运算数据通路】的选择线路不同。

在 MEM 阶段，由 EXE 阶段计算得出的内存地址首先传入虚拟内存管理器 VM，由 VM 查找 TLB 翻译成物理地址。如果在翻译的过程中发现 TLB 缺失，则触发 TLB 缺失异常，取消该次内存访问（尤其注意可能为写操作），并将控制权交给异常处理器 EXCP。如果在翻译的过程中发现当前 CPU 运行级没有权限进行该操作（例如 1 级 CPU 读写内核内存，任意运行级下写入不可写内存），则触发页异常。

如果 VM 成功完成地址翻译操作，则由内存管理器 MMIO 完成真正的内存读写或者 I/O 操作。

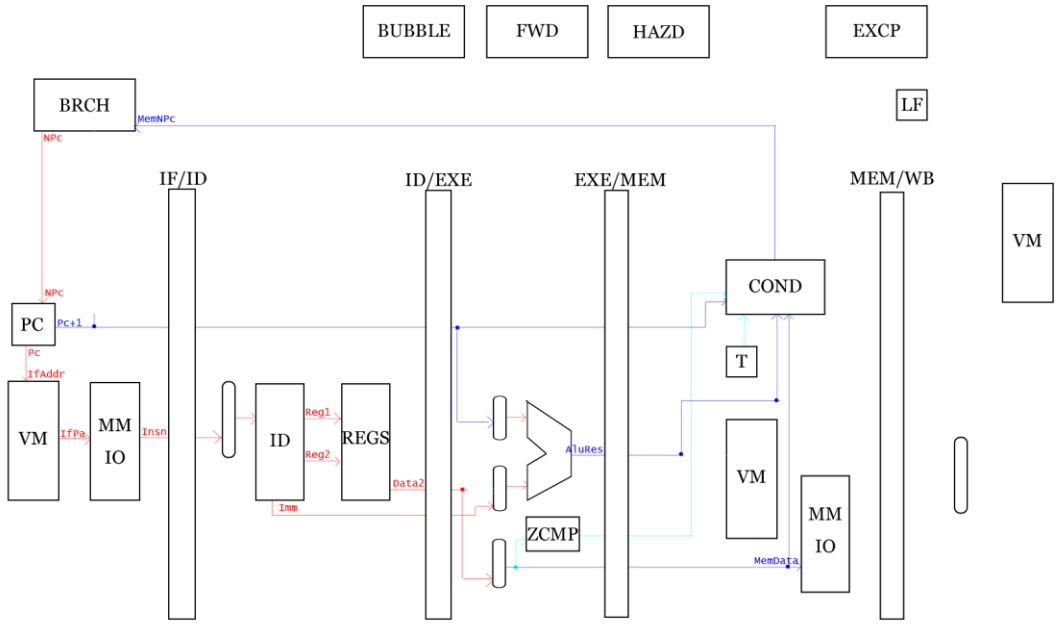
对于 I/O 端口，目前采用内存映射的方式进行访问。也就是将 I/O 端口映射到特定的物理地址上，由 MMIO 根据传入的物理地址决定是访问内存还是访问端口。特别地，如果读取

一个只能写入的端口或者写入一个只能读取的端口，这样的操作会被忽略，而不会触发异常。但是，以不符合硬件操作流程的方式读写端口可能导致不可预测的结果。

目前已经映射到物理地址的 I/O 端口有：

I/O 端口	物理地址	访问方式
串口状态	0xBF00	只读
串口数据	0xBF01	读写
键盘数据	0xBF02	只读
键盘状态	0xBF03	只读
VGA 数据	0xBF04	只写

3.3 跳转控制数据通路



目前指令集中的跳转指令指定的新指令地址只有两种情况：PC 加立即数或某个寄存器的值。对于跳转地址存放在某个寄存器中的指令（JR 与 JRRA），指令译码器 ID 将该寄存器的编号作为第二寄存器控制信号输出。

使用 ALU 计算 PC 加立即数，而将第二寄存器的值直接传送给跳转条件判断器 COND。对于 BEQZ 和 BNEZ 来说，跳转与否取决于第二寄存器的值是否为 0，因此，将第二寄存器的值传入零比较器 ZCMP。当输入为零时，该比较器输出 1，否则输出 0。

条件判断器 COND 接受三个待选择地址（顺序执行时的下一指令地址，ALU 的计算结果与第二寄存器的值）和跳转方式控制信号（由指令译码器 ID 生成，决定了跳转方式），生成下一指令的地址，并将该地址和跳转发生信号（如果该地址不是顺序执行的下一指令地址就置为 1）传给分支控制器 BRCH。

目前支持的跳转方式有：

1. 不跳转（所有非跳转指令）
2. 无条件跳转到 PC+立即数
3. 无条件跳转到第二寄存器的值
4. 当 T 为 0 时跳转到 PC+立即数
5. 当 T 为 1 时跳转到 PC+立即数
6. 当第二操作数为 0 时跳转到 PC+立即数
7. 当第二操作数为 1 时跳转到 PC+立即数

BRCH 根据 COND 传入的参数执行对应的操作。BRCH 实现了分支预测技术，可以预测时间上最近的一个分支。如果分支预测成功，就不需要重置流水线。

BRCH 的分支预测记录由一个 Key-Value 对组成，Key 代表跳转指令所在的指令地址+1（这里+1 是为了方便与程序计数器 PC 输出的值直接进行比较），Value 代表预测的下一条指令地址。

当 MEM 阶段指令不是跳转指令时，就按照分支预测记录输出 IF 的下一条指令地址（如果 Hit，就输出 Value，否则就输出原 PC+1）。

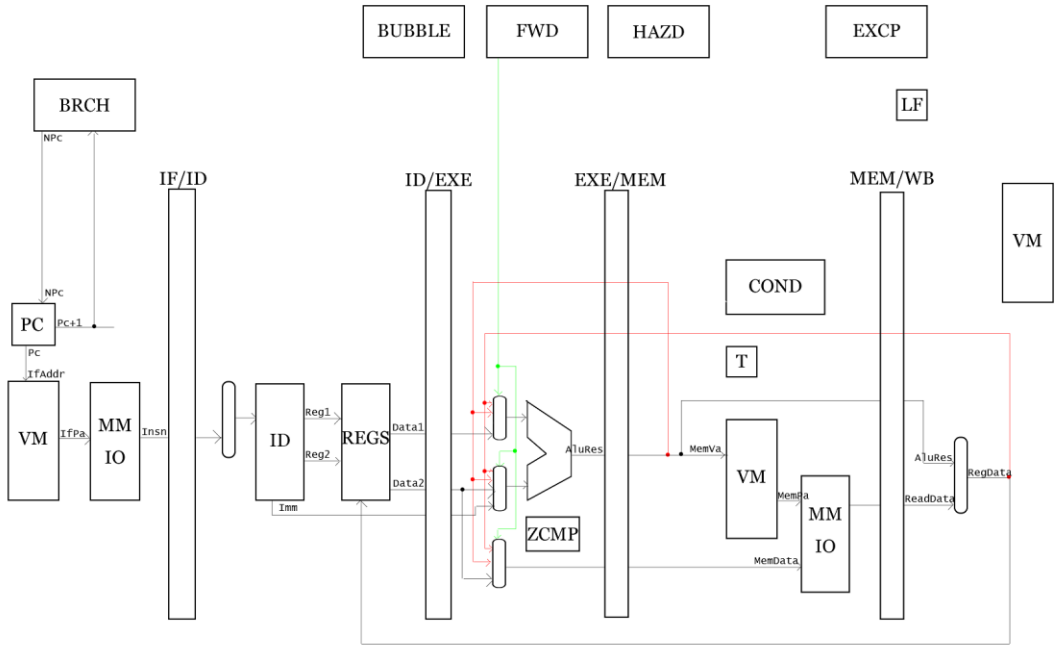
当 MEM 阶段指令为跳转指令并且顺序执行（跳转条件不满足）时，首先检查该分支指令是否在分支预测记录中，如果在，则需要检查 MEM 阶段所指定的下一指令地址与分支预测记录是否一致，如果一致，则说明分支预测正确，继续执行。如果不一致，则说明分支预测错误，清空流水线，修改分支预测记录，并将 MEM 阶段指定的下一指令地址写入程序计数器 PC。如果该指令不在分支预测记录中，则继续执行（因为是顺序执行）。

当 MEM 阶段指令为跳转指令并且跳转时，首先检查该分支指令是否在分支预测记录中。如果在，则需要检查 MEM 阶段所制定的下一指令地址与分支预测记录是否一致。如果一致，则说明分支预测正确，继续执行。否则，则说明分支预测错误，清空流水线，修改分支预测记录，并将 MEM 阶段指定的下一指令地址写入程序计数器 PC。以上操作与 MEM 阶段为跳转指令并且顺序执行时的处理流程相同。不同的是当该指令不在分支预测记录中

时，也清空流水线，修改分支预测记录，并将 MEM 阶段指定的下一指令地址写入程序计数器 PC。

以上三种情况均发生在没有异常处理的时候。如果在异常处理状态中（该状态包括发生中断或异常时、从中断返回时、处理 TLB 缺失时），则 BRCH 将中断处理器 EXCP 输出的 EPc 直接写入程序计数器 PC，以保证从异常处理状态退出时，直接从指定的指令地址处开始取指令并执行。

3.4 数据冲突与旁路数据通路



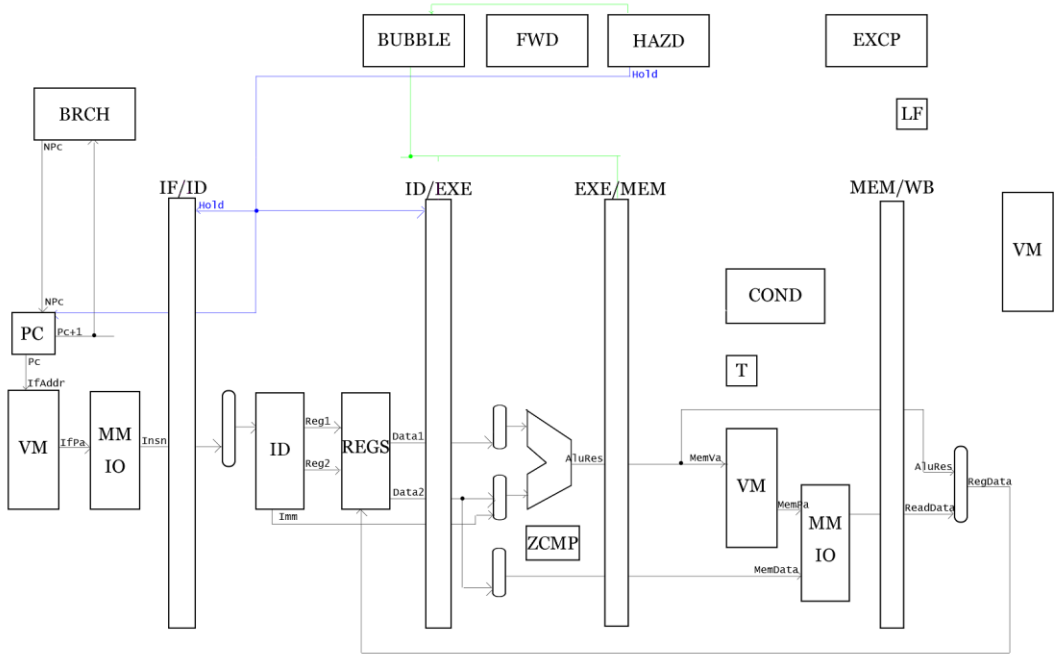
对于以下两种数据冲突，可以用旁路技术加以解决：

1. 相邻两条指令，前一条指令为算术逻辑运算指令，并且该指令的运算结果是后一条指令的输入。

2. 相隔一条指令的两条指令，前一条指令为算术逻辑运算指令，并且该指令的运算结果是后一条指令的输入。

为了检测以上两种数据冲突，转发控制器 FWD 监控 EXE 阶段、MEM 阶段、WB 阶段的信号。冲突检测条件参照《计算机组成与设计》。当冲突发生时，FWD 控制 EXE 阶段的选择器，将 MEM 或者 WB 阶段的数据直接传送到对应 ALU 操作数。

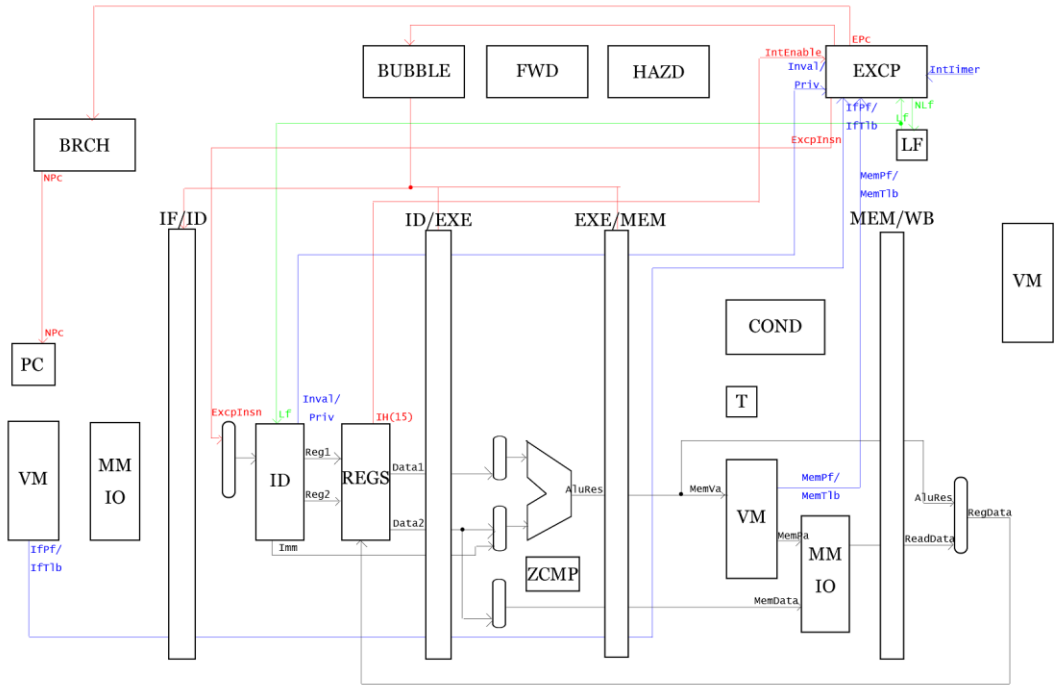
3.5 数据冲突/结构冲突与暂停流水线数据通路



通过旁路技术不能够解决读内存指令后紧接着就使用该值的情况，这种情况可以精确地描述为：相邻两条指令，前一条指令为 LD/LD_SP，后一条指令的操作数寄存器为前一条指令的目的寄存器。在这种情况下，因为旁路技术失效，只能暂停流水线。在暂停流水线时，冲突控制器 HAZD 一边通过给气泡生成器 BUBBLE 传入信号，使 EXE/MEM 寄存器的值无效化（变成 NOP 指令），另一边将保持信号 Hold 置为 1，使 PC、IF/ID、ID/EXE 三个寄存器的值不在下一个时钟上升沿到来时发生改变。这样，就相当于将 LD/LD_SP 指令后的其他指令全部延迟了一个周期，或者认为在流水线中插入了一个气泡。此时，不可解决的冲突变成可以通过旁路技术解决的冲突。

以上冲突处理方式不能够解决一个极端情况：如果后一条指令（设为指令 N）的另一个操作数寄存器为读内存指令的前一条指令（设为指令 PP）的目的寄存器，那么，暂停一个周期后，后一条指令仍然在 EXE 阶段，而指令 PP 已经离开 WB 阶段，因此，该值无法通过旁路传给处在 EXE 阶段的指令 N。对于这种特殊情况，一旦发现，就给分支控制器 BRCH 传递清空流水线信号 Reload，将读内存指令后的其余指令全部放弃（读内存指令照常完成），重新取指令执行。

3.6 异常与中断处理数据通路



异常与中断处理数据通路为全部 CPU 数据通路中最复杂的部分。异常处理器 EXCP 的任务为解决以下三个问题：

1. 异常、软中断者外部中断发生时，切换运行级与堆栈（如果 CPU 在 1 级），屏蔽外部中断，并准备好中断服务例程 ISR 所需要的数据，最后跳转到中断服务例程 ISR 的入口处。该任务被称为“异常准备”。

2. 中断服务例程 ISR 执行 ERET 指令或者某些特殊情况下执行 ERET（内核初始化完成后手动降级并运行用户态程序）时，清理堆栈，如果必要则恢复运行级并切换堆栈，最后跳转回被中断处继续执行（从下一条指令开始执行或者重新执行异常指令）。该任务被称为“从异常返回”。

3. IF 阶段或者 MEM 阶段地址翻译过程中发生 TLB 缺失异常时，将缺失的页表项加载进 TLB 并重新执行异常指令。

第三个问题在下一节【3.7 TLB 缺失处理数据通路】中被详细介绍。本节仅仅介绍第一和第二问题。

3.6.1 异常准备

异常准备包括从“发生异常、软中断、响应外部中断”到“跳转到中断服务例程入口”这一过程中需要做的一系列准备工作。针对（广义的，下同）异常发生时的 CPU 运行级的不同，这些准备工作也有差异。

对于从 0 级发生的异常，需要的准备工作为：

1. 将栈指针 SP 减 5，空出五个字，用于存放异常描述数据。
2. SP+4 处保存 IH 寄存器的值。该寄存器的值在 ERET 时从栈上恢复。
3. 将 IH 的最高位设为 0，使得进入中断服务例程时屏蔽外部中断，防止破坏性的中断嵌套。在中断服务例程中可以在安全的时候开启中断，但这由操作系统决定。
4. SP+3 处保存出错指令地址 EPc。对于页异常、非法指令、越权执行特权指令这三种异常，EPc 为异常指令的地址。对于软中断 INT、外部中断，EPc 为被中断指令的下一条指令地址。ERET 时该地址被加载入程序计数器 PC，使正常的指令流恢复。
5. SP+2 处保存异常中断号 ENo。ENo 的最高位为 0，表示从 0 运行级中断指令流。
6. SP+1 和 SP+0 处保存页异常描述信息 Reason 和 Addr，以便页异常处理程序可以妥善地处理该异常。
7. 跳转到中断服务例程的入口开始执行。

对于从 1 级发生的异常，需要的准备工作为：

1. 将当前运行级从 1 级上升到 0 级。
2. 将当前栈指针 SP 保存到用户态栈指针寄存器 UP。
3. 将内核态栈指针寄存器 KP 的值装入 SP，完成从用户栈换到内核栈的操作。
4. 以下的操作与处理 0 级异常相同，唯一的区别是 ENo 的最高位为 1，表示从 1 运行级中断指令流。

为了减少硬件复杂度，尽可能复用现有流水线，以上的操作由指令来实现。与正常的流水线不同的是，在异常准备状态中，需要执行的指令通过 EXCP 直接送到 ID 阶段，而越过 IF 阶段。为了能够在一条指令内存入整个 16 位立即数（如 EPc、ENo、Reason、Addr 等），我们引入长指令。EXCP 向 ID 传入的指令为长指令。长指令有 20 位长，高 4 位为指令操作

码，低 16 位为立即数。为了支持长指令，引入一个额外的寄存器：中断临时寄存器 ER，用于在异常准备中临时存放立即数的值。

目前支持的长指令有：

1. LI_ER <Imm>　　将一个立即数加载入 ER 寄存器
2. LW_ER <Imm>　　将一个内存单元中的值加载入 ER 寄存器
3. SW_ER <Imm>　　将 ER 寄存器的值存入 SP+<Imm>地址对应的内存
4. IH2ER　　将 IH 寄存器的值存入 ER 寄存器
5. ER2IH　　将 ER 寄存器的值存入 IH 寄存器
6. LTLB <Addr>　　将页表基地址寄存器 PD+<Addr>所指向的页表项加载入 TLB

因为长指令为 20 字节，最高位为 1，以便与普通指令区分，所以，在内存中不可能读取到长指令。也就是说，只有异常处理器 EXCP 能够将长指令送入流水线并执行。

长指令解决了 16 位立即数问题，因此，接下来就需要解决如何按顺序将这些指令送入流水线，并完成异常准备的任务。

为此，我们引入一个有限状态机，称之为异常状态机。异常状态机的状态 0 为正常状态（没有异常或者 MEM 阶段指令刚刚触发异常处理或者中断响应时处在该状态），其余状态均为异常状态。在异常状态下，不响应一切外部中断，程序计数器 PC 的值保持不变，IF 阶段不取指令，BRCH 接受 EXCP 传来的 EPc 值并直接写入 PC。因此，从异常状态回到正常状态时，流水线将从 EPc 处开始执行。

异常状态机在时钟上升沿时改变状态。因为这样的限制，当 MEM 阶段发生异常（或者需要响应外部中断）的时候，只有当下一个时钟上升沿到来时，EXCP 才能开始响应这一输入。因此，在下一个时钟周期，异常处理机仍然在状态 0。只有再来一个时钟上升沿，才能进入异常状态。

为了保证在异常发生时，流水线中的指令不会继续执行，EXCP 使用组合电路监控所有输入的异常信号和中断信号，一旦异常触发条件达成（有异常信号或中断信号，但是排除中断发生时 IH 最高位为 0、正在异常准备、从异常返回或者处理 TLB 异常的情况），就将 IF/ID、ID/EXE 和 EXE/MEM 流水线段寄存器全部无效化（全部变成 NOP 指令），以此清空流水线。而在异常状态中，仅仅将 IF/ID 段寄存器无效化，用来防止中断服务例程的第一个指令被执行两次。

状态 0 检查异常信号和中断信号，如果有异常或者中断发生（且没有被屏蔽），就设置好 EXCP 内部寄存器 EPc/ENo/Reason/Addr，并根据触发异常时的运行级将下一状态设为状态 1 或状态 3。

随着时钟上升沿一个接一个的到来，异常状态逐步切换，在每个状态中向 ID 阶段送入一条指令，最终返回正常状态，完成异常准备。

异常处理状态机的详细描述请参照附录 A。

3.6.2 从异常返回

从异常返回任务复用了异常准备的硬件结构。复用方法是，把从异常返回这一请求也当作一种异常或者软中断。当指令译码器 ID 遇到 ERET 指令时，它生成 Ret 信号，当该信号到达 MEM 阶段并被 EXCP 收到时，就触发异常。不同的是，EXCP 从状态 0 进入异常状态 12，完成从异常返回的操作。

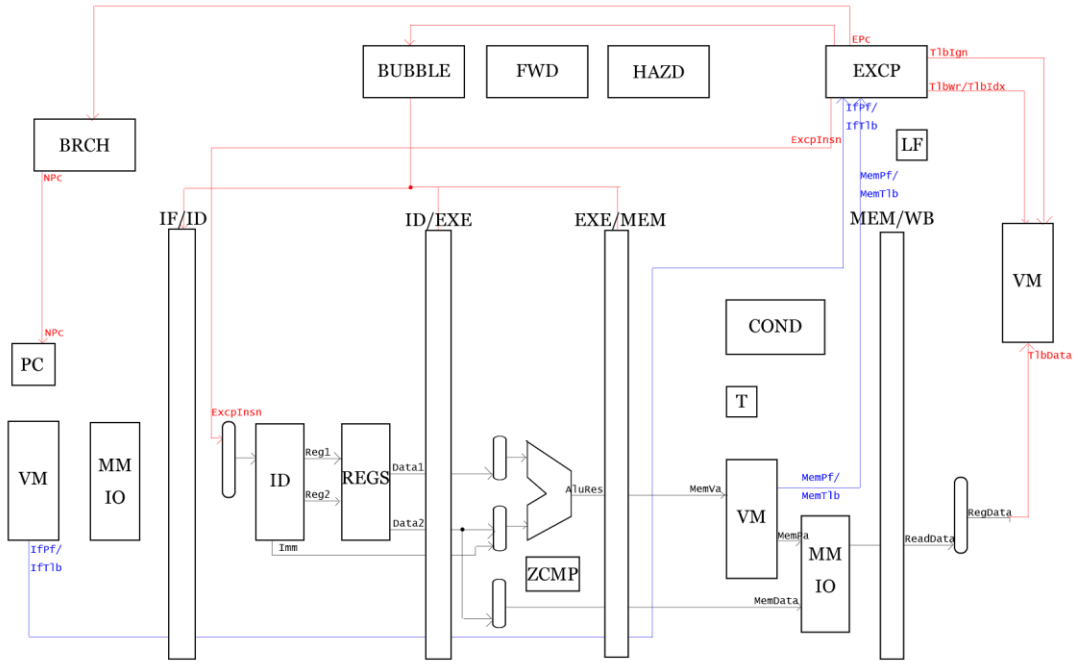
从异常返回时，操作系统必须保证内核栈的形式与刚刚执行中断服务例程时相同。而异常处理器 EXCP 需要完成如下工作：

1. 恢复 IH 寄存器。如果异常发生前允许中断，那么，从异常处理程序返回后也应该允许中断。反之亦然。
2. 恢复运行级。如果异常发生前为 1 级，那么，从异常处理程序返回后也应该是 1 级。但若是这样，就必须从内核栈切换到用户栈。
3. 从被中断处恢复执行。这一步也就是把栈上 SP+3（EPc）的值装入程序计数器 PC 即可。

考虑到恢复到运行级 0 或者运行级 1，EXCP 所执行的操作也略有不同：

1. 从内核栈保存的 IH 寄存器值恢复到 IH 寄存器。
2. 把内核栈保存的 EPc 加载到程序计数器 PC。
3. 如果 ENo 的高位为 0，则调到第 5 步。
4. 将 CPU 运行级降低。
5. 一系列 NOP 指令送入流水线，等待流水线中的指令执行完毕。
6. 回到正常状态，恢复正常指令流。

3.7 TLB 缺失处理数据通路



与从异常返回类似，为了利用现有硬件电路，对于 TLB 缺失的处理也使用已经存在的异常状态机，但是所经过的状态是 TLB 缺失所独有的状态。TLB 缺失异常发生时，异常状态机进入状态 0E。处理完 TLB 缺失异常后，异常状态机回归正常状态。

对于 TLB 缺失异常，异常处理器 EXCP 需要完成将缺失的页表项加载进 TLB 的操作，然后重新执行导致异常的指令。将页表项加载进 TLB 需要使用虚拟内存管理器 VM。虚拟内存管理器 VM 接受三个信号：写 TLB 信号 TlbWr，TLB 的虚拟地址信号 TlbVa，TLB 的页表项内容信号 TlbData。EXCP 在 TLB 异常触发时将导致异常的虚拟地址存入内部寄存器 Va。当页表项的内容从内存中读出并到达 WB 阶段时，EXCP 将 TlbWr 设为 1，将 TlbVa 设为 Va 的值，而 TlbData 连接到 WB 阶段写回数据的信号线，时钟上升沿触发 VM 修改 TLB，完成 TLB 缺失项的填充。

需要注意的是，在处理 TLB 缺失异常时，EXCP 访问页表项的地址为物理地址。即使 CPU 已经开启分页（IH 寄存器最低位为 1），在这个过程中，也必须禁止地址翻译。

四．MMIO——内存及其它 IO 控制模块设计

MMIO 模块是 CPU 的 IO 控制模块，主要负责将一个内存地址映射到两个 Ram、串口、PS2 接口、VGA 等外部设备上。控制完成读写操作。

4.1 相关模块及端口分配

使用到 MMIO 的模块有：

1、 取指阶段

2、 访存阶段

IO 分组及地址映射

所有 IO 端口被分为两组：

1、 Ram2。

2、 Ram1 及其它端口。

地址映射方式（19 位）：

地址第 16 位表示访问的 IO 端口组，0 表示第一组 IO，1 表示第二组 IO。

访问 Ram 时，将 19 位地址的高 3 位和低 15 位拼接为物理 Ram 的 18 地址输入。

其余 IO 端口映射：

I/O 端口	物理地址	访问方式
串口状态	0x0BF00	只读
串口数据	0x0BF01	读写
键盘数据	0x0BF02	只读
键盘状态	0x0BF03	只读
VGA 数据	0x0BF04	只写

若试图读取只写端口，取指操作将返回 NOP，访存操作将返回 0x0。

4.2 冲突及处理方案

可能出现的冲突：

取指、访存均要访问同一组外部设备。

冲突处理方案：

对于取指，直接返回 NOP 指令，正常执行访存操作，并置冲突标记为 1。

4.3 输入输出描述

输入：

1、 取指地址，19 位

2、 访存地址，19 位

3、 访存数据输入，16 位

4、 访存读、写信号，各 1 位，要求不能同时为 1。

5、 时钟，1 位。

输出：

1、 取指结果，16 位

2、 读出的内存数据，16 位

3、 冲突标记，1 位。

外部管脚：

1、 Ram1、Ram2 地址管脚、数据管脚、使能信号。

2、 串口读、写标记，串口读写使能。

3、 PS2 特殊时钟，PS2 数据信号。

4、 VGA 特殊时钟信号，输出数据管脚。

5、 Flash 数据、地址、使能等信号。

4.4 读写时序设计

读 Ram 数据时相对简单，根据输入条件判断要读取的 IO 端口，调整使能信号即可。

写 Ram 数据时，要求写使能开启过程中数据、地址都要保持稳定，并在数据写入后及时关闭写使能。因此选择在周期后半段进行写入操作。

为了保证 IO 使能开启过程中输入数据和地址均保持稳定，实现时利用了输入数据的延迟。具体方式是将写入使能和时钟作或后再接到 IO 使能上，同时要求访存数据的输入要比时钟信号延迟 5ns 以上方可保证 IO 写使能关闭前的数据稳定。由于 MMIO 是在 PC、VM 等模块的后面，因此上述要求的信号延迟很容易达到。

读写其它 IO 与读写 Ram 类似，并且由于其它 IO 模块对读写时序的要求更低，因此实现上更加简单一些。

五、外设模块设计

5.1 VGA 端口访问

实现 VGA 主要分为两个部分：

1. 创建足够使用又不至于多大的显存空间（FPGA 内有 RAM 模块）；
2. 根据 VGA 行场同步时序关系，发送像素的扫描信号、行同步信号和场同步信号。

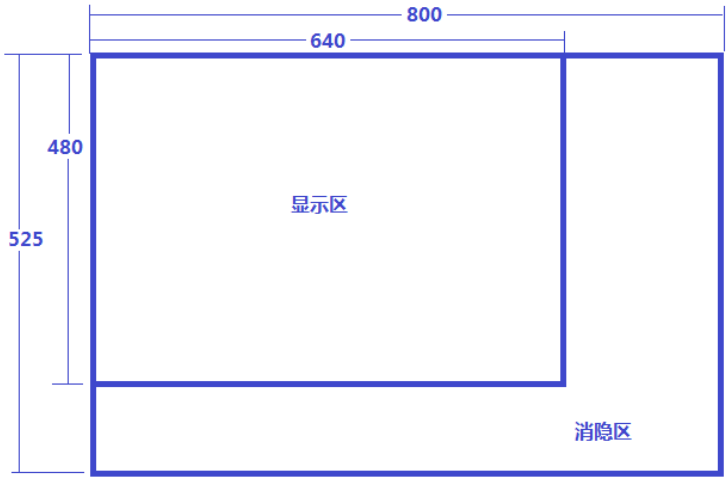
如果仅仅是画彩色正方形的显示效果，显存只用简单存储正方形左上角坐标、大小、颜色 RGB 三色值即可。通过输入信号调整这些信息，外接 50MHz 的时钟，每个上升沿从指定的端口获取信号，如果正方形相关的信息发生了调整（标志位），则更新相关信息，否则不做任何调整。

如果需要显示较为复杂的图像或者文字效果，需要将每个像素块（如 2*2 个像素点为单位组成像素块）的 RGB 三色信息（或者黑白两色信息）存储到 FPGA 内 RAM 对应的内存地址中。因为 FPGA 内 RAM 的存在，让我们可以通过硬件 VHDL 语言直接访问支配一定空间的内存，而不需要通过端口发送信息。因为端口的时钟是以 CPU 运行速度为基准的，也就是 12.5MHz，如果依次发送每个像素块的颜色信息，速度会非常慢，无法跟上 VGA 50MHz 的速度。所以通过 FPGA 内置的 RAM，将极大的增加了访问和存储像素点颜色信息的效率。

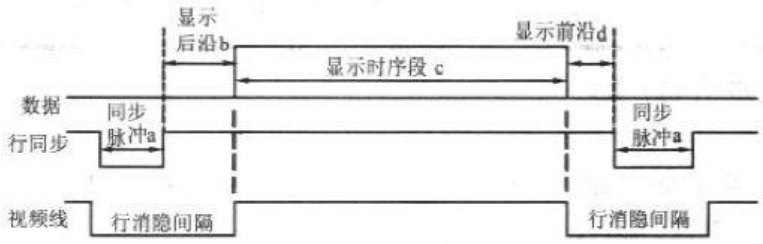
本次大实验中，我们组成功实现了较为简单的正方形显示效果，部分实现了较为复杂的显示效果，但是没有能够完成较为复杂的显示效果的软件驱动。

VGA 行场同步时序关系，可以简单理解为 VGA 在屏幕上由左到右、由上到下，一行一行的根据当前像素点的 RGB 三色信息绘制像素点，这是扫描信号；同时设置行同步信号 Hs 和

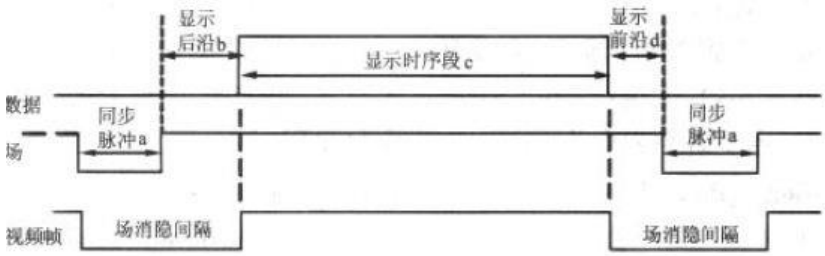
场同步信号 Vs，向 VGA 接收方同步当前所在的行列位置。对于 640*480 的显示屏幕，VGA 实际屏幕大小为 800*525，其中左上 640*480 为显示区，其余位置消隐区，消隐区的 RGB 三色信息需要置为 0。VGA 显示区和消隐区的示意如下图：



VGA 行/场扫描信号与同步信号的示意如下图：



VGA 的行时序



VGA 的场时序

5.2 PS/2 键盘访问

PS/2 单方向从键盘获取按键信息，主要是根据键盘通讯协议解析数据，并将按键信息存储到指定的端口。按照 PS/2 按键接口协议，每次按下和松开一个键会产生三组串行的数据。按下按成一组串行的数据 MAKE，松开产生两组串行的数据 BREAK，其中前一组为 F0，后一组和按下时产生的一样。PS/2 键盘扫描码节选如下表格：

KEY	MAKE	BREAK
R	2D	F0 2D
G	34	F0 34
B	32	F0 32
U ARROW	E0 75	E0 F0 75
L ARROW	E0 6B	E0 F0 6B
D ARROW	E0 72	E0 F0 72
R ARROW	E0 74	E0 F0 74

每组串行数据有 11 位，每组数据的包括：

1 位起始位，低电平；

8 位数据位（即扫描码），从低位开始传播；

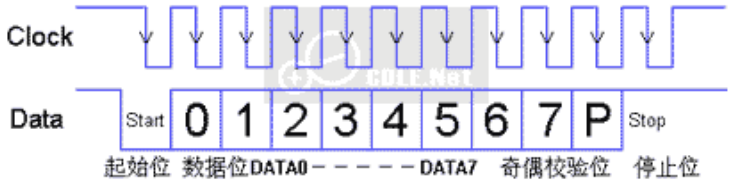
1 位奇偶校验位（奇校验）；

1 位结束位，高电平；

对于键盘到主机的通信，当时钟为高，数据线改变状态，在时钟信号的下降沿数据被封存。键盘数据传输过程大致如下：

1. 键盘不传输数据的时候，时钟线、数据线均保持高电平；
2. 当开始传播数据时，数据线首先变成低电平，等待第一个时钟下降沿；
3. 第一个时钟下降沿到来时，该电平将被作为起始位，由主机读入；
4. 下一个数据将在下一个时钟下降沿到来时准备好；
5. 当下一个下降沿到来时，主机将再次从数据线上读入下一个数据；

6. 如此重复，直到 11 个时钟周期结束，此时键盘时钟线、数据线再次恢复高电平。



VHDL 代码只需要将键盘发送过来的数据提取出数据位（即扫描码），经过奇校验确认无误后发送给指定串口即可，后续工作应该由软件执行。

5.3 VGA 与 PS/2 结合的软件驱动

这一个软件驱动中，PS/2 键盘将作为输入，VGA 将作为输出。VGA 的输出只需将显存需要更新的信息通过指定串口发送给硬件即可，对于简单的正方形方块，只需要根据时钟将最新的坐标信息、大小信息、颜色信息发给串口即可。

而对于 PS/2 键盘输入则需要维护一个状态机，根据键盘信号和当前的状态确定当前的 KEY。状态机状态转换表如下：

	2D	34	32	75	6B	72	74	E0	F0
START	S1	S2	S3	START	START	START	START	S4	START
S1	S1								S9
S2		S2							S9
S3			S3						S9
S4				S5	S6	S7	S8		S9
S5				S5				S4	
S6					S6			S4	
S7						S7		S4	
S8							S8	S4	
S9	START	START	START	START	START	START	START	START	START

其中浅绿色的单元格表示，在当前状态下，如果键盘输入数据为该信息的话，此时获取这个 KEY，根据这个 KEY 更新 VGA 正方形信息。比如在 S1 状态下键盘获取到 2D 这个数据，则将 VGA 正方形颜色更新为红色；在 S4 状态下获取到 75 这个数据，则将 VGA 正方形的位置向上移动一定的距离。

参考文献

[1]刘卫东 李山山 宋佳兴编著，《计算机硬件系统实验教程》，清华大学出版社，2013.10

附录 A 异常处理器 EXCP 的状态机

注释：对异常状态机中出现的信号进行说明。

InTlb 表示是否在处理 TLB 缺失异常。如果为 1，则除了屏蔽外部中断外，还禁用 VM，即是说，EXCP 传送的指令中涉及的所有地址均直接为物理地址。此外，InTlb=1 导致 EXCP 输出给 BRCH 的地址 EPc 为异常指令的地址。

InRet 表示是否在从异常中返回。如果为 1，则 EXCP 输出给 BRCH 的地址 EPc 为 ERET 时栈上的 EPc 值。

如果以上两个信号均为 0，则 EXCP 输出给 BRCH 的地址 EPc 为 0x2（中断服务例程入口地址）。

TlbWr 和 TlbVa 为写入 TLB 的控制信号。TlbWr 为 1 时，VM 将 TlbVa 和 WB 阶段的数据拼接成一个完整的 TLB 项，存入 TLB 中。其中 TlbVa 为缺失页表项对应的虚拟地址高位，在 TLB 异常发生时，该值被存入 EXCP 内部寄存器 Va。WB 阶段的数据为从内存中读出的页表项（页表项中只有物理地址高位和页属性，没有虚拟地址信息）。

00: 无异常状态。如果有异常信号，就无效化前三个阶段，并进入异常状态 1 或 3。否则，维持无异常状态。

01: SVUP

02: LDKP

03: ADDSP -5

04: IH2ER

05: 将 CPU 运行级设为 0, SW_ER 4

06: LI_ER <EPc>

07: SW_ER 3

08: LI_ER <ENo>

09: SW_ER 2

0A: LI_ER <Reason>

0B: SW_ER 1

0C: LI_ER <Addr>

OD: SW_ER 0 ===> 状态 0

0E: InTlb = 1, LTLB <TlbIdx>

OF: InTlb = 1, NOP

10: InTlb = 1, NOP

11: InTlb = 1, TlbWr = 1, TlbVa = <Va>,(将从内存中读出的页表项写入 TLB) NOP ===> 0

12: InRet = 1, LW_ER 4

13: InRet = 1, NOP

14: InRet = 1, ER2IH

15: InRet = 1, LW_ER 2

16: InRet = 1, NOP

17: InRet = 1, LW_ER 3

18: InRet = 1, ADDSP 5

19: InRet = 1, NOP, ===> 状态 1A/1C

1A: InRet = 1, DoDegrade <= 1 (使 CPU 运行级降级为 1), LDUP

1B: InRet = 1, DoDegrade <= 0 (降级信号清零), NOP ===> 00

1C: InRet = 1, NOP

1D: InRet = 1, NOP ===> 00

附录 B 新增指令的格式

新增特权指令的前五位均为 10000，后 4 位为指令代码，若指令需要访问寄存器，则寄存器编号放在 Rx 处（10-8 位）。以下只写出各特权指令与其后四位。

指令名称	指令代码	是否访问寄存器
MTUP	0000	Rx
MTKP	0001	Rx
MFUP	0010	Rx
MFKP	0011	Rx
SVUP	0100	无
LDKP	0101	无
MTPD	0111	Rx
CLI	1000	无
STI	1001	无
ERET	1010	无
LDUP	1011	无

另外，新增与 MTSP 配对的指令 MFSP，其格式与 MTSP 相同，差别是 MTSP 的 10-8 位为 100，MFSP 的 10-8 位为 101。

附录 C 长指令的格式

长指令的高 4 位为指令代码，低 16 位为 0 或者 16 位立即数。普通指令自动被扩展为高 4 位为 0 的长指令。下表列出了异常处理器 EXCP 使用的长指令：

指令名称	指令代码	是否使用立即数
LI_ER	1000	是
SW_ER	1001	是
LTLB	1010	是
LW_ER	1011	是
ER2IH	1100	否
IH2ER	1101	否