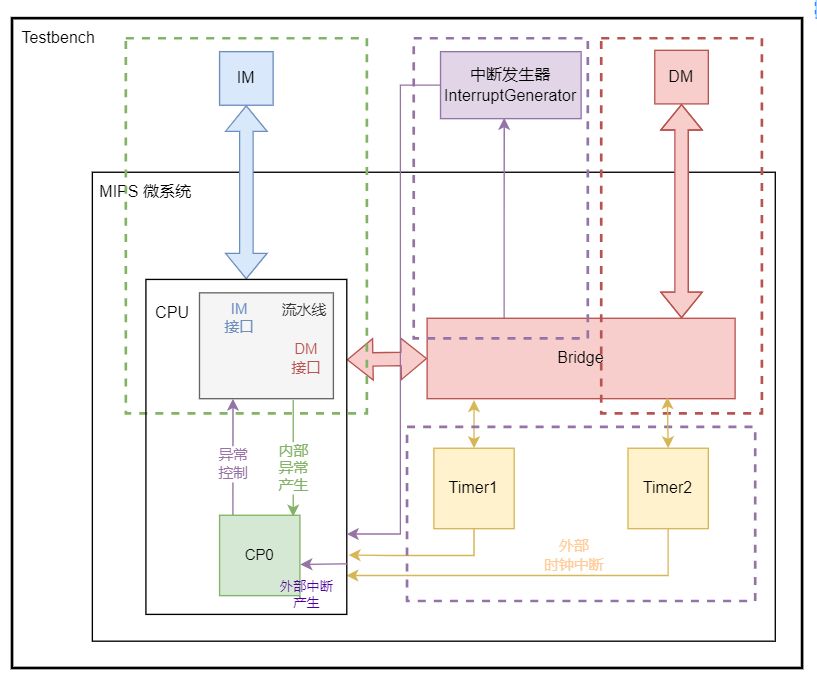
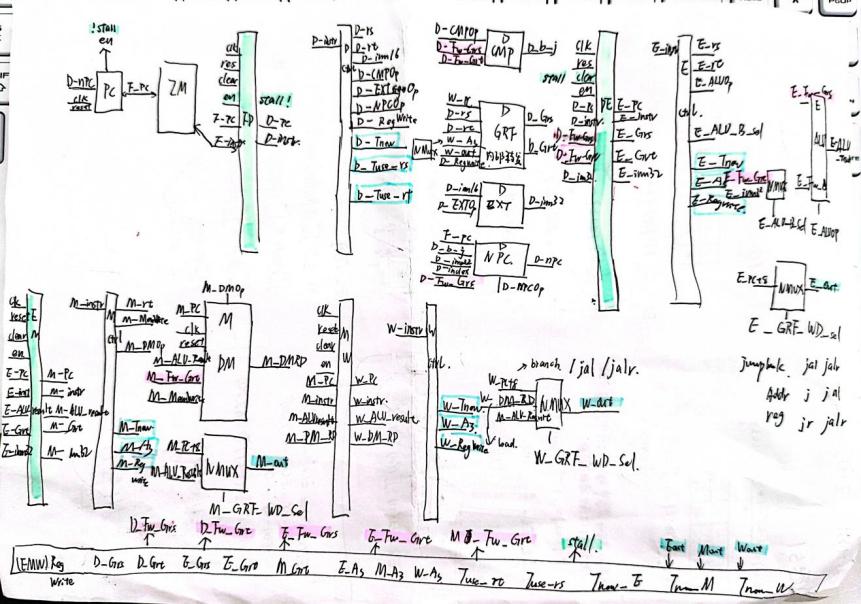
# MIPS微系统设计文档

## 设计方案综述





### Mips

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号，将PC值置为**0x00003000**  高电平有效 |
| interrupt | I | 外部中断信号 |
| macroscopic\_pc | O | 宏观pc |
| i\_inst\_addr | O | IM 读取地址（取指 PC） |
| i\_inst\_rdata | I | IM 读取数据 |
| m\_data\_addr | O | DM 读写地址 |
| m\_data\_rdata | I | DM 读取数据 |
| m\_data\_wdata | O | DM 待写入数据 |
| m\_data\_byteen | O | DM 字节使能信号 |
| m\_int\_addr | O | 中断发生器待写入地址 |
| m\_int\_byteen | O | 中断发生器字节使能信号 |
| m\_inst\_addr | O | M 级 PC |
| w\_grf\_we | O | GRF 写使能信号 |
| w\_grf\_addr | O | GRF 待写入寄存器编号 |
| w\_grf\_wdata | O | GRF 待写入数据 |
| w\_inst\_addr | O | W 级 PC |

### CPU

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号，将PC值置为**0x00003000**  高电平有效 |
| HWInt | I | 外部中断信号 |
| CPU\_inst\_rdata | I | IM 读取数据 |
| CPU\_DM\_rdata | I | DM 读取数据 |
| CPU\_inst\_addr | O | IM 读取地址 |
| CPU\_DM\_addr | O | DM 读写地址 |
| CPU\_DM\_wdata | O | DM 读取数据 |
| CPU\_DM\_byteen | O | DM 字节使能信号 |
| PC\_M | O | M 级 PC |
| CPU\_w\_grf\_we | O | GRF 写使能信号 |
| CPU\_w\_grf\_addr | O | GRF 待写入寄存器编号 |
| CPU\_w\_grf\_wdata | O | GRF 待写入数据 |
| PC\_W | O | W 级 PC |
| PC\_macro | O | 宏观pc |

### CP0

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号，将PC值置为**0x00003000**  高电平有效 |
| A1 | I | mfc0 读 rd |
| A2 | I | mtc0 写 rd |
| BD | I | 是不是延时槽指令 |
| Din | I | CP0寄存器的写入数据 |
| PC | I | 中断/异常时的PC |
| ExcCodeIn | I | 中断/异常的类型 |
| HWInt | I | 6个设备中断 |
| WE | I | CP0寄存器写使能 |
| EXLClr | I | 用于清除SR的EXL(EXL为0) |
| Req | O | 中断请求，输出至CPU控制器 |
| EPCOut | O | EPC寄存器输出至NPC |
| Dout | O | CP0寄存器的输出数据 |

### Bridge

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| CPU\_DM\_addr | I |  |
| CPU\_DM\_byteen | I |  |
| DM\_rdata | I | m\_data\_rdata |
| TC0\_Dout | I |  |
| TC1\_Dout | I |  |
| CPU\_DM\_rdata | O |  |
| DM\_Addr | O | m\_data\_addr |
| DM\_byteen | O | m\_data\_byteen |
| TC0\_Addr | O |  |
| TC1\_Addr | O |  |
| TC0\_WE | O |  |
| TC1\_WE | O |  |
| Int\_Addr | O | m\_int\_addr |
| Int\_byteen | O | m\_int\_byteen |

### TC

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| clk | I |  |
| reset | I |  |
| Addr | I | CPU\_DM\_addr[31:2] |
| WE | I | 写使能 |
| Din | I | CPU\_DM\_wdata |
| Dout | O | 32 位数据输出 |
| IRQ | O | 中断请求 |

|  |  |  |  |
| --- | --- | --- | --- |
| 异常与中断码 | 助记符与名称 | 指令与指令类型 | 描述 |
| 0 | Int （外部中断） | 所有指令 | 中断请求，来源于计时器与外部中断。 |
| 4 | AdEL （取指异常） | 所有指令 | PC 地址未字对齐。 |
| PC 地址超过 0x3000 ~ 0x6ffc。 |
| AdEL （取数异常） | lw | 取数地址未与 4 字节对齐。 |
| lh | 取数地址未与 2 字节对齐。 |
| lh, lb | 取 Timer 寄存器的值。 |
| load 型指令 | 计算地址时加法溢出。 |
| load 型指令 | 取数地址超出 DM、Timer0、Timer1、中断发生器的范围。 |
| 5 | AdES （存数异常） | sw | 存数地址未 4 字节对齐。 |
| sh | 存数地址未 2 字节对齐。 |
| sh, sb | 存 Timer 寄存器的值。 |
| store 型指令 | 计算地址加法溢出。 |
| store 型指令 | 向计时器的 Count 寄存器存值。 |
| store 型指令 | 存数地址超出 DM、Timer0、Timer1、中断发生器的范围。 |
| 8 | Syscall （系统调用） | syscall | 系统调用。 |
| 10 | RI（未知指令） | - | 未知的指令码。 |
| 12 | Ov（溢出异常） | add, addi, sub | 算术溢出。 |

## 测试方案

.text

ori $t0, $t0, 0xfc01

mtc0 $t0, $12

#系统调用

syscall

ori $t9, $t9, 1

#RI

#Ov

lui $t1, 0x8000

lui $t2, 0x8000

add $t2, $t1, $t2

ori $t9, $t9, 1

#AdES

lui $t1, 0xffff

ori $t2,$0,0x7f24

ori $t3,$0,0x7f00

ori $t4,$0,0x7f10

sw $t1,1($0)

sh $t1,1($0)

#存 Timer 寄存器的值

sh $t1,0($t3)

sb $t1,0($t3)

sw $t1,0($t3)

sh $t1,0($t4)

sb $t1,0($t4)

sw $t1,0($t4)

sw $t1,0x1000($t1) #计算地址加法溢出

#向计时器的 Count 寄存器存值

sw $t1,0x7f08($0)

sw $t1,0x7f18($0)

#存数地址超出 DM、Timer0、Timer1、中断发生器的范围

sw $t1,0($t2)

sb $t1,0($t2)

sh $t1,0($t2)

ori $t9, $t9, 1

#AdEL

lui $t0, 0xffff

ori $t2,$0,0x7f24

ori $t3,$0,0x7f00

ori $t4,$0,0x7f10

lw $t1,1($0)

lh $t1,1($0)

#取 Timer 寄存器的值

lh $t1,0($t3)

lb $t1,0($t3)

lw $t1,0($t3)

lh $t1,0($t4)

lb $t1,0($t4)

lw $t1,0($t4)

lw $t1,0x1000($t0) #计算地址加法溢出

#存数地址超出 DM、Timer0、Timer1、中断发生器的范围

lw $t1,0($t2)

lb $t1,0($t2)

lh $t1,0($t2)

ori $t9, $t9, 1

end: beq $0, $0, end

nop

.ktext 0x4180

mfc0 $k0, $12

mfc0 $k0, $13

mfc0 $k0, $14

mfc0 $k0, $13

ori $k1, $0, 0x7c

and $k0, $k1, $k0

beq $0, $k0, ERET

addi $k0, $k0, 0xffff

mfc0 $k0, $14

lui $1, 0xffff

ori $1, $1, 0xfffc

and $k1, $k1, $1

addu $k0, $k0, 4

mtc0 $k0, $14

j ERET

ERET:

eret

## **思考题汇总**

1. **请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？**

鼠标键盘发出不同中断信号，中断信号经过中断控制器传到CPU，然后CPU根据不同的中断信号号执行不同的中断响应程序，然后进行相应的IO操作

1. **请思考为什么我们的 CPU 处理中断异常必须是已经指定好的地址？如果你的 CPU 支持用户自定义入口地址，即处理中断异常的程序由用户提供，其还能提供我们所希望的功能吗？如果可以，请说明这样可能会出现什么问题？否则举例说明。（假设用户提供的中断处理程序合法）**

如果让用户自定义入口地址，是可以的，但是将入口地址写入寄存器可能会出现很多问题，比方说入口地址还没写入下一条指令在M级异常还有增多的阻塞情况。

1. **为何与外设通信需要 Bridge？**

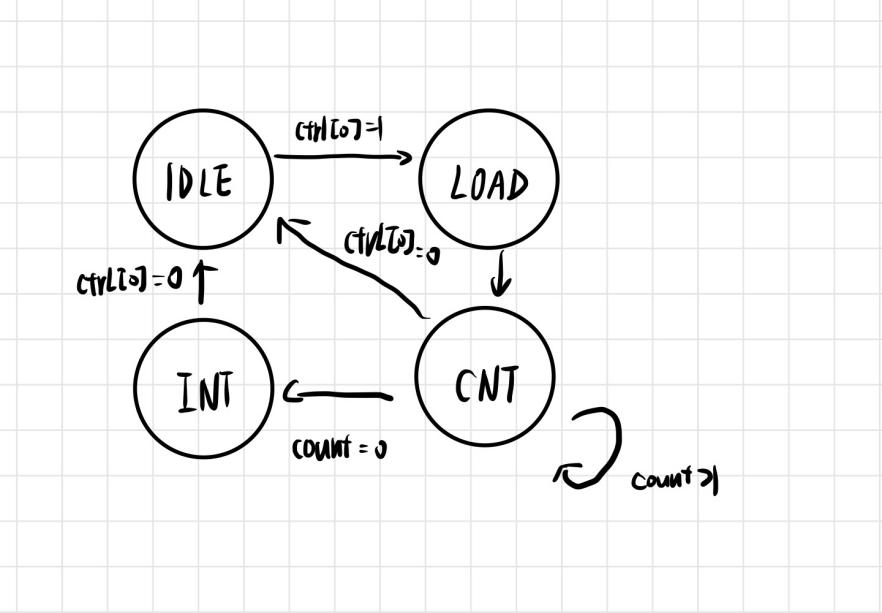
便于 CPU 可以用统一的方法访问不同外设

Bridge是连接 CPU 和外设的功能设备，它会给 CPU 提供一种接口，使得 CPU 可以像读写普通存储器一样（即按地址读写）来读写复杂多变的外设

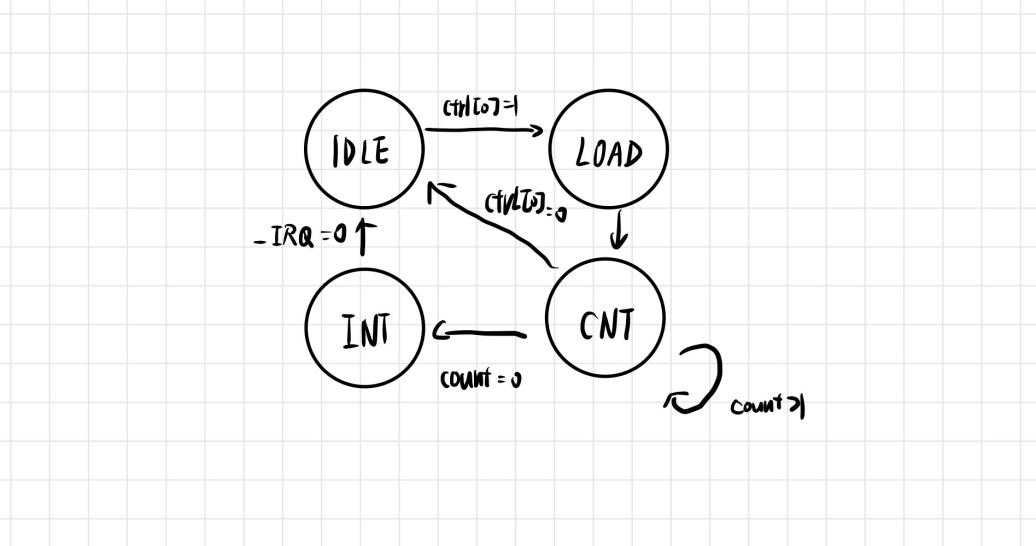
Bridge统一且简化了 CPU 的对外接口，CPU 不必为每种外设单独提供接口，符合高内聚，低耦合的设计思想。

1. **请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态移图。**

模式0：当计数器倒计数为 0 后，计数器停止计数，此时控制寄存器中的使能Enable自动变为 0。当使能 Enable 被设置为 1 后，初值寄存器值再次被加载至计数器，计数器重新启动倒计数。模式 0 下的中断信号将持续有效，直至控制寄存器中的中断屏蔽位被设置为0。



模式1：当计数器倒计数为 0 后，初值寄存器值被自动加载至计数器，计数器继续倒计数。模式 1 下计数器每次计数循环中只产生一周期的中断信号。

****

1. **倘若中断信号流入的时候，在检测宏观 PC 的一级如果是一条空泡（你的 CPU 该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在 P7 中，清空流水线产生的空泡指令应该保留原指令的哪些信息？**

这时宏观pc是0，判断是否延迟槽信号错误

所以应该保留PC和判断是否延迟槽信号

**6、为什么 jalr 指令为什么不能写成 jalr $31, $31？**

因为如果jalr的延时槽里中断/异常，$31已经被改变了，等异常处理介绍再次运行jalr无法恢复执行