

# 微系统设计方案综述

## 总体设计概述

本微系统为 Verilog 实现的 MIPS 微系统，包括流水线 CPU、Bridge、计时器，CPU 支持的指令集包括 lb, lbu, lh, lhu, lw, sb, sh, sw, add, addu, sub, subu, mult, multu, div, divu, sll, srl, sra, sllv, srlv, srav, and, or, xor, nor, addi, addiu, andi, ori, xori, lui, slt, slti, sltiu, sltu, beq, bne, blez, bgtz, bltz, bgez, j, jal, jalr, jr, mfhi, mflo, mthi, mtlo, mfc0, mtc0, eret。为了实现这些功能，CPU 主要包含了 IFU, GRF, CP0, DM, ALU, MDU, IF\_ID, ID\_EX, EX\_MEM, MEM\_WB, SFU, DEC, CTRL 这些模块。

## 关键模块定义

### 1.IFU

#### 介绍

取指令单元，内部包括 PC(程序计数器)、IM(指令存储器)及相关逻辑，其中 PC 具有同步复位功能，IM 的容量为 容量为 32bit \* 4096，起始地址为 0x00003000。

#### 端口定义

端口	输入/输出	位宽	描述
nPC	I	32	设置下一个 PC 值。
WE	I	1	使能端。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
I	O	32	当前指令。
PC	O	32	当前 PC 值。

#### 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将 PC 设置为 0x00003000。
2	取指令	当时钟上升沿到来时，I 输出当前 PC 对应的指令。
3	取PC值	当时钟上升沿到来时，PC 输出当前 PC 值。
4	设置PC值	当时钟上升沿到来且使能端有效时，将 nPC 设置为当前 PC 值。

## 2.GRF

### 介绍

通用寄存器组，也称为寄存器文件、寄存器堆，可以存取 32 位数据，具有同步复位功能。寄存器标号为 0 到 31，其中 0 号寄存器读取的结果恒为 0。

### 端口定义

端口	输入/输出	位宽	描述
PC	I	32	当前指令的 PC 值。
A1	I	5	指定 32 个寄存器中的一个，将其存储的数据读出到 RD1。
A2	I	5	指定 32 个寄存器中的一个，将其存储的数据读出到 RD2。
A3	I	5	指定 32 个寄存器中的一个，作为写入的目标寄存器。
WD	I	32	写入寄存器的数据信号。
WE	I	1	写使能信号，高电平有效。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
RD1	O	32	输出 A1 指定的寄存器中的数据。
RD2	O	32	输出 A2 指定的寄存器中的数据。

### 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有寄存器的值设置为 0x00000000。
2	读数据	读出 A1 和 A2 地址对应寄存器中存储的数据到 RD1 和 RD2；当 WE 有效时会把 WD 的值会实时反馈到对应的 RD1 或 RD2，即内部转发。
3	写数据	当 WE 有效且时钟上升沿到来时，将 WD 的数据写入 A3 对应的寄存器中。

### 3.CP0

#### 介绍

协处理器 0，包含 4 个 32 位寄存器，用于支持中断和异常。

#### 端口定义

端口	输入/输出	位宽	描述
A1	I	5	指定 4 个寄存器中的一个，将其存储的数据读出到 RD。
A2	I	5	指定 4 个寄存器中的一个，作为写入的目标寄存器。
WD	I	32	写入寄存器的数据信号。
nEPC	I	32	目前传入的下一个 EPC 值。
nExc	I	5	目前传入的下一个 ExcCode 值。
nBD	I	32	目前传入的下一个 BD 值。
HWInt	I	6	外部硬件中断信号。
WE	I	1	写使能信号，高电平有效。
ERET	I	1	eret 指令信号，高电平有效。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
IntReq	O	1	输出当前的中断请求。
RD	O	32	输出 A 指定的寄存器中的数据。

#### 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有寄存器的值设置为 0x00000000。
2	读数据	读出 A1 地址对应寄存器中存储的数据到 RD；当 WE 有效时会将 WD 的值会实时反馈到对应的 RD，当 ERET 有效时会将 EXL 置 0，即内部转发。
3	写数据	当 WE 有效且时钟上升沿到来时，将 WD 的数据写入 A2 对应的寄存器中。
4	中断处理	根据各种传入信号和寄存器的值判断当前是否要进行中断，将结果输出到 IntReq。

## 4.ALU

### 介绍

算术逻辑单元，提供 32 位的多种运算功能。

### 端口定义

端口	输入/输出	位宽	描述
A	I	32	参与 ALU 计算的第一个值。
B	I	32	参与 ALU 计算的第二个值。
Op	I	4	ALU 功能的选择信号，具体见功能定义。
AO	O	32	ALU 的计算结果。
Exc	O	1	计算过程中是否产生溢出。

### 功能定义

序号	功能名称	功能描述
1	按位与	当 Op = 0 时, $AO = A \& B$ 。
2	按位或	当 Op = 1 时, $AO = A   B$ 。
3	加法, 忽略溢出	当 Op = 2 时, $AO = A + B$ 。
4	减法, 忽略溢出	当 Op = 3 时, $AO = A - B$ 。
5	左移 16 位	当 Op = 4 时, $AO = A \ll 16$ 。
6	有符号比大小	当 Op = 5 时, $AO = A < B$ 。
7	无符号比大小	当 Op = 6 时, $AO = A < B$ 。(有符号)
8	逻辑左移	当 Op = 7 时, $AO = A \ll B[4:0]$ 。
9	逻辑右移	当 Op = 8 时, $AO = A \gg B[4:0]$ 。
10	算术右移	当 Op = 9 时, $AO = A \ggg B[4:0]$ 。
11	异或	当 Op = 10 时, $AO = A \wedge B$ 。
12	或非	当 Op = 11 时, $AO = \sim(A   B)$ 。
13	加法, 不忽略溢出	当 Op = 12 时, $AO = A + B$ , 如果产生溢出 Exc = 1。
14	减法, 不忽略溢出	当 Op = 13 时, $AO = A - B$ , 如果产生溢出 Exc = 1。

## 5.MDU

### 介绍

乘除块, 内有 hi 和 lo 两个寄存器, 可以进行乘除运算, 完成相关乘除指令, 并模拟乘除运算的延时。

### 端口定义

端口	输入/输出	位宽	描述
A	I	32	参与乘除运算的第一个值。
B	I	32	参与乘除运算的第二个值。
Op	I	4	MDU 功能的选择信号, 具体见功能定义。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
Busy	O	1	当前是否正在进行乘除运算。
hi	O	32	hi 寄存器的值。
lo	O	32	lo 寄存器的值。

功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将 MDU 中所有值设为 0x00000000。
2	有符号乘	当 Op = 1 时，将 A * B 的高 32 位和低 32 位分别写入 hi 和 lo 寄存器，延迟为 5 个时钟周期。（有符号）
3	无符号乘	当 Op = 2 时，将 A * B 的高 32 位和低 32 位分别写入 hi 和 lo 寄存器，延迟为 5 个时钟周期。（无符号）
4	有符号除	当 Op = 3 时，将 A / B 的余数和商分别写入 hi 和 lo 寄存器，延迟为 10 个时钟周期。（有符号）
5	无符号除	当 Op = 4 时，将 A / B 的余数和商分别写入 hi 和 lo 寄存器，延迟为 10 个时钟周期。（无符号）
6	写 hi 寄存器	当 Op = 7 时，将 A 写入 hi 寄存器。
7	写 lo 寄存器	当 Op = 8 时，将 A 写入 lo 寄存器。

6.DM

介绍

数据存储器，可以存取 32 位数据，容量为 32bit \* 4096，具有同步复位功能，起始地址为 0x00000000。

端口定义

端口	输入/输出	位宽	描述
PC	I	32	当前指令的 PC 值。
A	I	5	读取或写入数据的地址。
WD	I	32	写入 DM 中的数据。
Op	I	2	DM 功能的选择信号，具体见功能定义。
PrRD	I	32	来自计时器的输入信号。
IntReq	I	1	中断请求信号。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
RD	O	32	根据 A 和 Op 输出对应的数据。
PrWE	O	1	输入到计时器的写使能信号。
Exc	O	2	无异常时为 0，取数异常时为 1，存数异常时为 2。

功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将 DM 中所有值设为 0x00000000。
2	有符号读字节	当 Op = 0 时，输出对应字节的 32 位有符号扩展。
3	无符号读字节	当 Op = 1 时，输出对应字节的 32 位无符号扩展。
4	有符号读半字	当 Op = 2 时，输出对应半字的 32 位有符号扩展。
5	无符号读半字	当 Op = 3 时，输出对应半字的 32 位无符号扩展。
6	读字	当 Op = 4 时，输出对应字。
7	写字节	当 Op = 5 且时钟上升沿来临时，将 WD[7:0] 写入 A 对应的地址。
8	写半字	当 Op = 6 且时钟上升沿来临时，将 WD[15:0] 写入 A 对应的地址。
9	写字	当 Op = 7 且时钟上升沿来临时，将 WD 写入 A 对应的地址。

7.IF\_ID

介绍

I 级和 D 级间的流水线寄存器，同时产生 D 级的控制信号。

端口定义

端口	输入/输出	位宽	描述
nl	I	32	下一个指令。
nPC	I	32	下一个 PC 值。
WE	I	1	使能端
nBD	I	1	下一个 BD 值。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
I	O	32	当前指令。
PC	O	32	当前 PC 值。
BD	O	1	当前 BD 值。
PCSrc	O	4	0: $PC = PC + 4$ 。 1: 跳转到 beq 指令对应的跳转地址。 2: 跳转到 jal 指令和 j 指令对应的跳转地址。 3: 跳转到 jalr 指令和 jr 指令对应的跳转地址。 4: 跳转到 bne 指令对应的跳转地址。 5: 跳转到 blez 指令对应的跳转地址。 6: 跳转到 bgtz 指令对应的跳转地址。 7: 跳转到 bltz 指令对应的跳转地址。 8: 跳转到 bgez 指令对应的跳转地址。

### 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有值设为 0x00000000。
2	读数据	读出各个当前寄存器对应的值。
3	写数据	当 WE 有效且时钟上升沿到来时，将各个对应的值写入寄存器中。

## 8.ID\_EX

### 介绍

D 级和 E 级间的流水线寄存器，同时产生 E 级的控制信号。

### 端口定义



端口	输入/ 输出	位 宽	描述
nl	I	32	下一个指令。
nRD1	I	32	下一个 RD1 值。
nRD2	I	32	下一个 RD2 值。
nPC	I	32	下一个 PC 值。
nBD	I	1	下一个 BD 值。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
I	O	32	当前指令。
RD1	O	32	当前 RD1 值。
RD2	O	32	当前 RD2 值。
PC	O	32	当前 PC 值。
BD	O	1	当前 BD 值。
ALUOp	O	4	ALU 功能的选择信号，具体见 ALU 模块的功能定义。
ALUSrc	O	3	0：ALU 的 A 输入端选择 RD1 输出端，ALU 的 B 输入端选择 RD2 输出端。 1：ALU 的 A 输入端选择 RD1 输出端，ALU 的 B 输入端选择 I[15:0] 的 32 位无符号扩展。 2：ALU 的 A 输入端选择 RD1 输出端，ALU 的 B 输入端选择 I[15:0] 的 32 位有符号扩展。 3：ALU 的 A 输入端选择 RD2 输出端，ALU 的 B 输入端选择 I[10:6]。 4：ALU 的 A 输入端选择 RD2 输出端，ALU 的 B 输入端选择 RD1 输出端。
MDUOp	O	3	MDU 功能的选择信号，具体见 MDU 模块的功能定义。

## 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有值设为 0x00000000。
2	读数据	读出各个当前寄存器对应的值。
3	写数据	当时钟上升沿到来时，将各个对应的值写入寄存器中。

## 9.EX\_MEM

### 介绍

E 级和 M 级间的流水线寄存器，同时产生 M 级的控制信号。

### 端口定义

端口	输入/输出	位宽	描述
nI	I	32	下一个指令。
nAO	I	32	下一个 ALU 计算结果。
nRD2	I	32	下一个 RD2 值。
nPC	I	32	下一个 PC 值。
nBD	I	1	下一个 BD 值。
nExc	I	5	下一个 ExcCode。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
I	O	32	当前指令。
AO	O	32	当前 ALU 计算结果。
RD2	O	32	当前 RD2 值。
PC	O	32	当前 PC 值。
BD	O	1	当前 BD 值。
Exc	O	5	当前 ExcCode。
MemOp	O	3	DM 功能的选择信号，具体见 DM 模块的功能定义。
MemtoReg	O	2	0：准备写回 GRF 的 WD 输入端选择 AO 输出端。 1：准备写回 GRF 的 WD 输入端选择 DM 的 RD 输出端。 2：准备写回 GRF 的 WD 输入端选择 PC + 8。

### 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有值设为 0x00000000。
2	读数据	读出各个当前寄存器对应的值。
3	写数据	当时钟上升沿到来时，将各个对应的值写入寄存器中。

## 10.MEM\_WB

### 介绍

M 级和 W 级间的流水线寄存器，同时产生 W 级的控制信号。

### 端口定义

nI	I	32	下一个指令。
nPC	I	32	下一个 PC 值。
nWD	I	32	下一个写入 GRF 的值。
clk	I	1	时钟信号。
reset	I	1	同步复位信号。
I	O	32	当前指令。
PC	O	32	当前 PC 值。
WD	O	32	当前写入 GRF 的值。
RegWrite	O	1	0：GRF 的写使能端 WE 无效。 1：GRF 的写使能端 WE 有效。
RegDst	O	2	0：准备写回 GRF 的 A3 输入端选择 I[20:16]。 1：准备写回 GRF 的 A3 输入端选择 I[15:11]。 2：准备写回 GRF 的 A3 输入端选择 31。
RegWrite	O	1	0：GRF 的写使能端 WE 无效。 1：GRF 的写使能端 WE 有效。

### 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有值设为 0x00000000。
2	读数据	读出各个当前寄存器对应的值。
3	写数据	当时钟上升沿到来时，将各个对应的值写入寄存器中。

## 11.SFU

### 介绍

暂停 (Stall) 和 转发 (Forward) 的控制单元，产生两者的控制信号。

### 端口与功能定义

端口	输入/输出	位宽	描述
if_id_l	I	32	IF/ID 流水线寄存器当前的指令。
id_ex_l	I	32	ID/EX 流水线寄存器当前的指令。
ex_mem_l	I	32	EX/MEM 流水线寄存器当前的指令。
mem_wb_l	I	32	MEM/WB 流水线寄存器当前的指令。
Busy	I	1	乘除块的 Busy 信号。
Start	I	1	乘除块的 Start 信号。
Stall	O	1	暂停信号。
F_if_id_rs	O	3	0: D 级 Rs 选择 GRF 的 RD1 输出端。 1: D 级 Rs 选择 ID/EX 流水线寄存器的 PC + 8 输出端。 2: D 级 Rs 选择 EX/MEM 流水线寄存器的 PC + 8 输出端。 3: D 级 Rs 选择 EX/MEM 流水线寄存器的 AO 输出端。 5: D 级 Rs 选择 EX/MEM 流水线寄存器的 RD2 输出端。
F_if_id_rt	O	3	0: D 级 Rt 选择 GRF 的 RD2 输出端。 1: D 级 Rt 选择 ID/EX 流水线寄存器的 PC + 8 输出端。 2: D 级 Rt 选择 EX/MEM 流水线寄存器的 PC + 8 输出端。 3: D 级 Rt 选择 EX/MEM 流水线寄存器的 AO 输出端。 5: D 级 Rt 选择 EX/MEM 流水线寄存器的 RD2 输出端。
F_if_id_cp0rd	O	3	0: D 级 Rd 选择 CP0 的 RD 输出端。 5: D 级 Rd 选择 EX/MEM 流水线寄存器的 RD2 输出端。
F_id_ex_rs	O	3	0: E 级 Rs 选择 ID/EX 流水线寄存器的 RD1 输出端。 2: E 级 Rs 选择 EX/MEM 流水线寄存器的 PC + 8 输出端。 3: E 级 Rs 选择 EX/MEM 流水线寄存器的 AO 输出端。 4: E 级 Rs 选择 MEM/WB 流水线寄存器的 WD 输出端。 5: E 级 Rs 选择 EX/MEM 流水线寄存器的 RD2 输出端。
F_id_ex_rt	O	3	0: E 级 Rt 选择 ID/EX 流水线寄存器的 RD2 输出端。 2: E 级 Rt 选择 EX/MEM 流水线寄存器的 PC + 8 输出端。 3: E 级 Rt 选择 EX/MEM 流水线寄存器的 AO 输出端。 4: E 级 Rt 选择 MEM/WB 流水线寄存器的 WD 输出端。 5: E 级 Rt 选择 EX/MEM 流水线寄存器的 RD2 输出端。

端口	输入/输出	位宽	描述
F_id_ex_cp0rd	O	3	0：E 级 Rt 选择 ID/EX 流水线寄存器的 RD2 输出端。 4：E 级 Rd 选择 MEM/WB 流水线寄存器的 WD 输出端。 5：E 级 Rd 选择 EX/MEM 流水线寄存器的 RD2 输出端。
F_ex_mem_rt	O	3	0：M 级 Rt 选择 EX/MEM 流水线寄存器的 WD 输出端。 4：M 级 Rt 选择 MEM/WB 流水线寄存器的 WD 输出端。

## 数据冒险分析表

IF/ID当前指令			ID/EX(Tnew)					EX/MEM(Tnew)					MEM/WB(Tnew)				
指令类型	源寄存器	Tuse	jal 0/rd	cp0 mfc0.1/r mtc0.1/rd	cal_i 1/rt	cal_r 1/rd	load 2/rt	jal 0/rd	cp0 mfc0.0/rt mtc0.0/rd	cal_r 0/rd	cal_i 0/rt	load 1/rt	jal 0/rd	cp0 mfc0.0/rt mtc0.0/rd	cal_r 0/rd	cal_i 0/rt	load 0/rt
j	rs/rt	0		暂停	暂停	暂停	暂停					暂停					
cal_r	rs/rt	1					暂停										
cal_i	rs	1					暂停										
ld	rs	1					暂停										
st	rs	1					暂停										
st	rt	2															
mtc0	rt	1					暂停										
mfc0	cp0.rd	1															
eret	cp0.rd	0		暂停													
流水级	源寄存器	涉及指令	控制信号	PC + 8				PC + 8	RD2	AO	AO		WD	WD	WD	WD	WD
IF/ID	rs	j	F_if_id_rs														
	rt	j	F_if_id_rt														
	cp0.rd	eret	F_if_id_cp0rd														
ID/EX	rs	cal_r, cal_i, ld, st	F_id_ex_rs														
	rt	cal_r, st, mtc0	F_id_ex_rt														
	cp0.rd	mfc0	F_id_ex_cp0rd														
EX/MEM	rt	st	F_ex_mem_rt														

## 12.DEC

### 介绍

指令分类译码器，输入一个指令，返回该指令在数据冒险中对应的类型。

### 端口与功能定义

端口	输入/输出	位宽	描述
I	I	32	需要分类的指令。
j	O	1	需要读寄存器的跳转指令。
r	O	1	除 jalr, jr 外的 R 型指令。
i	O	1	I 型指令。
ld	O	1	load 型指令。
st	O	1	store 型指令。
jal	O	1	jal, jalr 指令。
rs	O	5	I[25:21]。
rt	O	5	I[20:16]。
rd	O	5	如果指令为 jal 输出 31，否则输出 I[15:11]。

# 13.CTRL

## 介绍

指令译码器，输入 Op 和 Func，输出该指令是哪个指令。本 CPU 采用分布式译码，对应控制信号的功能已在上述四个流水线寄存器中详细描述。

## 端口与功能定义

端口	输入/输出	位宽	描述
Op	I	6	所有指令的操作码，对应 I[31:26]。
Func	I	6	R 指令中辅助识别的操作码，对应 I[5:0]。
每种指令	O	1	判断是否是每种指令。

## 重要机制实现方法

### M 级处理中断

将异常信号流水到 M 级，同时结合外部中断信号，判断此时是否要进行中断，如果进行中断就清空所有流水寄存器，并将 nPC 改为 0x00004180。

这样做 MEM/WB 流水线寄存器的指令如果需要写入 GRF，可以恰好写入而不用再特殊处理。

## 测试方案

### 典型测试样例

- 取指异常：

```
.text

li $28, 0
li $29, 0

# jr PC mod 4 not 0
la $1, label1
la $2, label1
addiu $1, $1, 1
jr $1
nop
label1:

# jr PC < 0x3000
li $1, 0x2996
la $2, label2
jr $1
nop
label2:

# jr PC > 0x4ffc
li $1, 0x4fff
la $2, label3
jr $1
```

```

nop
label3:

end:j end

.ktext 0x4180
mfc0 $12, $12
mfc0 $13, $13
mfc0 $14, $14
mtc0 $2, $14
eret
ori $1, $0, 0

```

- 其它异常:

```

.text

li $28, 0
li $29, 0

lw $1, 1($0)
lh $1, 1($0)
lhu $1, 1($0)

lh $1, 0x7f00($0)
lhu $1, 0x7f04($0)
lb $1, 0x7f08($0)
lbu $1, 0x7f10($0)
lb $1, 0x7f14($0)
lb $1, 0x7f18($0)

li $2, 0x7fffffff
lw $1, 1($2)
lh $1, 1($2)
lhu $1, 1($2)
lb $1, 1($2)
lbu $1, 1($2)

lw $1, 0x3000($0)
lh $1, 0x4000($0)
lhu $1, 0x6000($0)
lb $1, 0x7f0c($0)
lbu $1, 0x7f1c($0)

sw $1, 1($0)
sh $1, 1($0)

sh $1, 0x7f00($0)
sb $1, 0x7f04($0)
sh $1, 0x7f08($0)
sb $1, 0x7f10($0)
sh $1, 0x7f14($0)
sb $1, 0x7f18($0)

li $2, 0x7fffffff
sw $1, 1($2)

```

```

sh $1, 1($2)
sb $1, 1($2)

sw $1, 0x7f08($0)
sh $1, 0x7f08($0)
sb $1, 0x7f08($0)
sw $1, 0x7f18($0)
sh $1, 0x7f18($0)
sb $1, 0x7f18($0)

sw $1, 0x3000($0)
sh $1, 0x4000($0)
sh $1, 0x6000($0)
sb $1, 0x7f0c($0)
sb $1, 0x7f1c($0)

msub $1, $2

li $1, 0x7fffffff
add $1, $1, $1
addi $1, $1, 1
li $1, 0x80000000
add $1, $1, $1
addi $1, $1, -1
sub $1, $1, $2
sub $1, $2, $1

end:j end

.ktext 0x4180
mfc0 $12, $12
mfc0 $13, $13
mfc0 $14, $14
addi $14, $14, 4
mtc0 $14, $14
eret
ori $1, $0, 0

```

- 计时器功能测试:

```

.text
li $12, 0x0c01
mtc0 $12, $12

li $1, 500
li $2, 9

sw $1, 0x7f04($0)
sw $2, 0x7f00($0)
li $1, 1000
sw $1, 0x7f14($0)
sw $2, 0x7f10($0)

lw $1, 0x7f00($0)
lw $1, 0x7f04($0)
lw $1, 0x7f10($0)

```



```

lw $1, 0x7f14($0)

li $1, 0
li $2, 0

for:
ori $3, $3, 0
beq $1, $0, for
nop
beq $2, $0, for
nop

lw $1, 0x7f00($0)
lw $1, 0x7f04($0)
lw $1, 0x7f10($0)
lw $1, 0x7f14($0)

end:j end

.ktext 0x4180
mfc0 $13, $13
li $15, 0x7fffffff
and $13, $13, $15
li $14, 1024
beq $13, $14, timer0
nop
li $14, 2048
beq $13, $14, timer1
nop
eret

timer0:
li $1, 1
sw $0, 0x7f00($0)
eret

timer1:
li $2, 2
sw $0, 0x7f10($0)
eret

```

- 延迟槽异常测试:

```

.text

li $28, 0
li $29, 0

li $1, 1
bne $0, $0, end
lw $1, 1($0)
li $1, 1
bne $0, $0, end
lh $1, 1($0)
li $1, 1
bne $0, $0, end
lhu $1, 1($0)

```

```
li $1, 1
bne $0, $0, end
lh $1, 0x7f00($0)
li $1, 1
bne $0, $0, end
lhu $1, 0x7f04($0)
li $1, 1
bne $0, $0, end
lb $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
lbu $1, 0x7f10($0)
li $1, 1
bne $0, $0, end
lb $1, 0x7f14($0)
li $1, 1
bne $0, $0, end
lb $1, 0x7f18($0)
```

```
li $2, 0x7fffffff
li $1, 1
bne $0, $0, end
lw $1, 1($2)
li $1, 1
bne $0, $0, end
lh $1, 1($2)
li $1, 1
bne $0, $0, end
lhu $1, 1($2)
li $1, 1
bne $0, $0, end
lb $1, 1($2)
li $1, 1
bne $0, $0, end
lbu $1, 1($2)
```

```
li $1, 1
bne $0, $0, end
lw $1, 0x3000($0)
li $1, 1
bne $0, $0, end
lh $1, 0x4000($0)
li $1, 1
bne $0, $0, end
lhu $1, 0x6000($0)
li $1, 1
bne $0, $0, end
lb $1, 0x7f0c($0)
li $1, 1
bne $0, $0, end
lbu $1, 0x7f1c($0)
```

```
li $1, 1
bne $0, $0, end
sw $1, 1($0)
li $1, 1
bne $0, $0, end
```

```
sh $1, 1($0)

li $1, 1
bne $0, $0, end
sh $1, 0x7f00($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f04($0)
li $1, 1
bne $0, $0, end
sh $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f10($0)
li $1, 1
bne $0, $0, end
sh $1, 0x7f14($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f18($0)

li $2, 0x7fffffff
li $1, 1
bne $0, $0, end
sw $1, 1($2)
li $1, 1
bne $0, $0, end
sh $1, 1($2)
li $1, 1
bne $0, $0, end
sb $1, 1($2)

li $1, 1
bne $0, $0, end
sw $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
sh $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
sw $1, 0x7f18($0)
li $1, 1
bne $0, $0, end
sh $1, 0x7f18($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f18($0)

li $1, 1
bne $0, $0, end
sw $1, 0x3000($0)
li $1, 1
bne $0, $0, end
sh $1, 0x4000($0)
li $1, 1
```

```

bne $0, $0, end
sh $1, 0x6000($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f0c($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f1c($0)

li $1, 1
bne $0, $0, end
msub $1, $2

li $1, 0x7fffffff
li $11, 1
bne $0, $0, end
add $1, $1, $1
li $11, 1
bne $0, $0, end
addi $1, $1, 1
li $1, 0x80000000
li $11, 1
bne $0, $0, end
add $1, $1, $1
li $11, 1
bne $0, $0, end
addi $1, $1, -1
li $11, 1
bne $0, $0, end
sub $1, $1, $2
li $11, 1
bne $0, $0, end
sub $1, $2, $1

end:j end

.ktext 0x4180
mfc0 $12, $12
mfc0 $13, $13
mfc0 $14, $14
addi $14, $14, 8
mtc0 $14, $14
eret
ori $1, $0, 0

```

## 自动测试工具

### 全自动化对拍器（仅测试 P6 + mfc0 + mtc0 + IO）

- 重复下述过程无数次：
  - 使用 C++ 随机化生成一段 MIPS 汇编代码。
  - 使用魔改版的 Mars，执行汇编指令时会按照格式输出评测机需要的信息（即需要 `$display` 的信息）。

- 使用 Mars 的命令行代码编译并执行，将输出导入到 `m.out`。
- 使用 Mars 的命令行代码编译并将机器码导出到 `code.txt`。
- 使用 IVerilog 命令行代码将 testbench 文件转换为可进行仿真的 `tb.out` 文件。
- 使用 IVerilog 命令行代码执行仿真，并将输出导入到 `v.out`。
- 使用 C++ 删掉 `v.out` 中多余的信息（例如一些警告和时间），将 `m.out` 和 `v.out` 分别排序。
- 使用 `fc` 将 `v.out` 和 `m.out` 进行比对，在第一组 WA 的数据点停下。
- 代码如下：
  - 数据生成器：

```
#include <bits/stdc++.h>

using namespace std;

vector<int> r;
mt19937 mt(time(0));
uniform_int_distribution<int>
    u16(0, (1 << 16) - 1),
    s16(-(1 << 15), (1 << 15) - 1),
    siz(0, 15),
    reg(0, 2),
    grf(1, 30),
    shift(0, 31),
    I(1, 42),
    J(43, 51),
    IJ(1, 51),
    one(11, 42),
    cp0(12, 14),
    b(0, 1);

int cnt, tot;

int getR(){
    return r[reg(mt)];
}

void solve(int i){
    int x, X;
    switch(i){
        case 1:
            x = getR();
            printf("ori %d, $0, 0\n", x);
            printf("lb %d, %d($d)\n", getR(), siz(mt), x);
            tot += 2;
            break;
        case 2:
            x = getR();
            printf("ori %d, $0, 0\n", x);
            printf("lbu %d, %d($d)\n", getR(), siz(mt), x);
            tot += 2;
            break;
        case 3:
            x = getR();
            printf("ori %d, $0, 0\n", x);
            printf("lh %d, %d($d)\n", getR(), siz(mt) >> 1 << 1, x);
            tot += 2;
    }
}
```

```

        break;
case 4:
    x = getR();
    printf("ori %d, $0, 0\n", x);
    printf("lhu %d, %d($d)\n", getR(), siz(mt) >> 1 << 1, x);
    tot += 2;
    break;
case 5:
    x = getR();
    printf("ori %d, $0, 0\n", x);
    x = b(mt) ? siz(mt) >> 2 << 2 : b(mt) ? 0x7f04 : 0x7f14;
    printf("lw %d, %d($d)\n", getR(), X, x);
    tot += 2;
    break;
case 6:
    x = getR();
    printf("ori %d, $0, 0\n", x);
    printf("sb %d, %d($d)\n", getR(), siz(mt), x);
    tot += 2;
    break;
case 7:
    x = getR();
    printf("ori %d, $0, 0\n", x);
    printf("sh %d, %d($d)\n", getR(), siz(mt) >> 1 << 1, x);
    tot += 2;
    break;
case 8:
    x = getR();
    printf("ori %d, $0, 0\n", x);
    x = b(mt) ? siz(mt) >> 2 << 2 : b(mt) ? 0x7f04 : 0x7f14;
    printf("sw %d, %d($d)\n", getR(), X, x);
    tot += 2;
    break;
case 9:
    x = getR();
    printf("ori %d, %d, 1\n", x, x);
    printf("div %d, %d\n", getR(), x);
    tot += 2;
    break;
case 10:
    x = getR();
    printf("ori %d, %d, 1\n", x, x);
    printf("divu %d, %d\n", getR(), x);
    tot += 2;
    break;
case 11:
    printf("add %d, $0, %d\n", getR(), getR());
    tot++;
    break;
case 12:
    printf("addu %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 13:
    x = getR();
    printf("sub %d, %d, %d\n", getR(), x, x);
    tot++;
    break;

```

```
case 14:
    printf("subu %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 15:
    printf("mult %d, %d\n", getR(), getR());
    tot++;
    break;
case 16:
    printf("multu %d, %d\n", getR(), getR());
    tot++;
    break;
case 17:
    printf("slt %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 18:
    printf("sltu %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 19:
    printf("sll %d, %d, %d\n", getR(), getR(), shift(mt));
    tot++;
    break;
case 20:
    printf("srl %d, %d, %d\n", getR(), getR(), shift(mt));
    tot++;
    break;
case 21:
    printf("sra %d, %d, %d\n", getR(), getR(), shift(mt));
    tot++;
    break;
case 22:
    printf("sllv %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 23:
    printf("srlv %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 24:
    printf("srav %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 25:
    printf("and %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 26:
    printf("or %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 27:
    printf("xor %d, %d, %d\n", getR(), getR(), getR());
    tot++;
    break;
case 28:
    printf("nor %d, %d, %d\n", getR(), getR(), getR());
```

```

        tot++;
        break;
case 29:
    printf("addi %d, %d, %d\n", getR(), getR(), 0);
    tot++;
    break;
case 30:
    printf("addiu %d, %d, %d\n", getR(), getR(), s16(mt));
    tot++;
    break;
case 31:
    printf("andi %d, %d, %d\n", getR(), getR(), u16(mt));
    tot++;
    break;
case 32:
    printf("ori %d, %d, %d\n", getR(), getR(), u16(mt));
    tot++;
    break;
case 33:
    printf("xori %d, %d, %d\n", getR(), getR(), u16(mt));
    tot++;
    break;
case 34:
    printf("lui %d, %d\n", getR(), u16(mt));
    tot++;
    break;
case 35:
    printf("slti %d, %d, %d\n", getR(), getR(), s16(mt));
    tot++;
    break;
case 36:
    printf("sltiu %d, %d, %d\n", getR(), getR(), s16(mt));
    tot++;
    break;
case 37:
    printf("mfhi %d\n", getR());
    tot++;
    break;
case 38:
    printf("mflo %d\n", getR());
    tot++;
    break;
case 39:
    printf("mthi %d\n", getR());
    tot++;
    break;
case 40:
    printf("mtlo %d\n", getR());
    tot++;
    break;
case 41:
    printf("mfc0 %d, %d\n", getR(), cp0(mt));
    tot++;
    break;
case 42:
    printf("mtc0 %d, %d\n", getR(), b(mt) ? 12 : 14);
    tot++;
    break;

```



```

case 43:
    printf("beq %d, %d, label%d\n", getR(), getR(), ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
    tot++;
    break;
case 44:
    printf("bne %d, %d, label%d\n", getR(), getR(), ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
    tot++;
    break;
case 45:
    printf("blez %d, label%d\n", getR(), ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
    tot++;
    break;
case 46:
    printf("bgtz %d, label%d\n", getR(), ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
    tot++;
    break;
case 47:
    printf("bltz %d, label%d\n", getR(), ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
    tot++;
    break;
case 48:
    printf("bgez %d, label%d\n", getR(), ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
    tot++;
    break;
case 49:
    printf("j label%d\n", ++cnt);
    solve(I(mt));
    solve(I(mt));
    printf("label%d: ", cnt);
    solve(I(mt));
    tot++;
    break;
case 50:
    printf("jal label%d\n", ++cnt);

```

```

        X = getR();
        printf("ori %d, $0, 16\n", X);
        solve(one(mt));
        printf("label%d: addu %d, %d, $31\n", cnt, X, X);
        printf("jr %d\n", X);
        puts("nop");//solve(I(mt));
        tot += 4;
        break;
    case 51:
        printf("jal label%d\n", ++cnt);
        X = getR();
        printf("ori %d, $0, 16\n", X);
        solve(one(mt));
        printf("label%d: addu %d, %d, $31\n", cnt, X, X);
        printf("jalr %d, %d\n", getR(), X);
        puts("nop");//solve(I(mt));
        tot += 4;
        break;
    }
}

int main(){
    r.push_back(grf(mt)), r.push_back(grf(mt)), r.push_back(grf(mt));
    freopen("test.asm", "w", stdout);
    puts("ori $28, $0, 0");
    puts("ori $29, $0, 0");
    puts("mtc0 $0, $12");
    while(tot < 900) solve(IJ(mt));
}

```

- 格式处理器:

```

#include <bits/stdc++.h>
#define maxn 100086

using namespace std;

vector<string> v, w;
char s[maxn];

int main(){
    freopen("v.out", "r", stdin);
    while(gets(s) != NULL){
        string S = s;
        v.push_back(s);
    }
    freopen("v.out", "w", stdout);

    for(int i = 0; i < v.size(); i++){
        if(v[i].length() <= 20 || v[i][20] != '@') continue;
        v[i] = v[i].substr(20);
        if(v[i][11] == '$' && v[i][12] == ' ' && v[i][13] == '0')
            continue;
        w.push_back(v[i]);
    }
    sort(w.begin(), w.end());
    for(int i = 0; i < w.size(); i++) printf("%s\n", w[i].c_str());
}

```

```

v.clear();
freopen("m.out", "r", stdin);
while(gets(s) != NULL){
    string S = s;
    v.push_back(s);
}
freopen("m.out", "w", stdout);
sort(v.begin(), v.end());
for(int i = 0; i < v.size(); i++){
    if(v[i][0] == '@') printf("%s\n", v[i].c_str());
}
}

```

- 评测代码:

```

#include <bits/stdc++.h>

using namespace std;

char s[10086];
int cnt = 0;

int main(){
    while(1){
        system("gen.exe");
        system("java -jar Mars_Changed.jar db mc CompactDataAtZero nc
test.asm > m.out");
        system("java -jar Mars_Changed.jar mc CompactDataAtZero a dump .text
HexText code.txt test.asm > log.txt");
        system("iverilog -o tb.out -y D:\\coding\\CO\\verilog\\P7
D:\\coding\\CO\\verilog\\P7\\tb.v");
        system("vvp tb.out > v.out");
        system("del.exe");
        system("fc v.out m.out > log.txt");
        freopen("log.txt", "r", stdin);
        gets(s), gets(s);
        printf("test%d:", ++cnt);
        if(s[0] != 'F'){
            puts("Wrong Answer!");
            break;
        }
        puts("Accepted!");
    }
}

```

- 效果图如下:

```
C:\Windows\system32\cmd.exe
test656:Accepted!
test657:Accepted!
test658:Accepted!
test659:Accepted!
test660:Accepted!
test661:Accepted!
test662:Accepted!
test663:Accepted!
test664:Accepted!
test665:Accepted!
test666:Accepted!
test667:Accepted!
test668:Accepted!
test669:Accepted!
test670:Accepted!
test671:Accepted!
test672:Accepted!
test673:Accepted!
test674:Accepted!
test675:Accepted!
test676:Accepted!
test677:Accepted!
test678:Accepted!
test679:Accepted!
test680:Accepted!
test681:Accepted!
test682:Accepted!
test683:Accepted!
test684:Accepted!
```

## 思考题

1. 我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？  
(Tips: 什么是接口？和我们到现在为止所学的有什么联系？)
  - “硬件/软件接口”是指令（机器码）。硬件实现了一些功能，并按照规约可以被相应的指令所操控。软件通过规约使用相应的指令操控硬件完成相应的功能，从而达到软件所期望的效果。指令在这个过程中实现了硬件软件的对接，因此是“硬件/软件接口”。
2. 在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。
  - 位于 CPU 外部和各种外接设备中。
3. BE 部件对所有的外设都是必要的吗？
  - 不是，只有对**字节/半字**有存取需求的才有必要。
4. 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转移图。
  - 见计时器说明文档。
5. 请开发一个主程序以及定时器的exception handler。整个系统完成如下功能：
  1. 定时器在主程序中被初始化为模式0；
  2. 定时器倒数计数至0产生中断；
  3. handler设置使能Enable为1从而再次启动定时器的计数器。2及3被无限重复。
  4. 主程序在初始化时将定时器初始化为模式0，设定初值寄存器的初值为某个值，如100或1000。（注意，主程序可能需要涉及对CP0.SR的编程，推荐阅读过后文后再进行。）

```
■ .text
li $t2, 0x0401
mtc0 $t2, $t2
li $t1, 100
li $t2, 9
sw $t1, 0x7f04($0)
sw $t2, 0x7f00($0)

for:j for
nop

.ktext 0x4180
li $t1, 100
li $t2, 9
sw $t1, 0x7f04($0)
```

```
sw $2, 0x7f00($0)
eret
```

6. 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？

- 鼠标和键盘产生中断信号，进入中断处理区的对应位置，将输入信号从鼠标和键盘中读入寄存器。