# 使用verilog的MIPS 微系统设计文档

## 0. 综述

本CPU为Verilog实现的MIPS微系统，支持中断与异常。支持的指令集包含{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} + {eret, mfc0, mtc0}+{MOVZ、RtBgezal}。为了实现这些指令，CPU主要包含了：IFU、GRF、ALU、MDU、DM、Ext、Control、TransStallController、BrCmp、Datapath、MUX_4、NPC、settings、CP0、Bridge、Timer等模块。

支持的指令分类图：

| | | | | | | |
|---|---|---|---|---|---|---|
| calr(24) | add | addu | sub | subu | | |
| | sll | srl | sra | sllv | srlv | srav |
| | and | or | xor | nor | | |
| | slt | sltu | | | | |
| | mult | multu | div | divu | | |
| | mthi | mtlo | mfhi | mflo | | |
| cali(8) | addi | addiu | | | | |
| | andi | ori | xori | | | |
| | lui | | | | | |
| | slti | stliu | | | | |
| 内存读(5) | lw | | | | | |
| | lb | lbu | | | | |
| | lh | lhu | | | | |
| 内存写(3) | sw | sh | sb | | | |
| 分支与跳转(10) | beq | bne | blez | bgtz | bltz | bgez |
| | j | jal | jr | jalr | | |
| CP0指令 | eret | mtc0 | mfc0 | | | |

支持的异常：

| ExcCode | 助记符 | 描述 |
|---|---|---|
| 0 | Int | 中断 |
| 4 | AdEL | 取数或取指时地址错误 |
| 5 | AdES | 存数时地址错误 |
| 10 | RI | 不认识的（或者非法的）指令码 |
| 12 | Ov | 自陷形式的整数算术指令（例如add）导致的溢出 |

- stage 部分模块：F_part模块（IFU、F_Exc）；D_part模块（GRF、Ext、NPCcalu、BrCmp、D_Exc、Trans_grf_mux1、Trans_grf_mux2）；Ex_part模块（ALU、MDU、E_Exc、

ETrans_ALUIn_MUX1、Trans_ALUIn_MUX2）；MEM_part（Dm、Trans_MemRD_MUX、M_Exc、CP0）；WB_part；

- 流水线寄存器模块：IFtoIDreg、IDtoEXreg、EXtoMEMreg、MEMtoWBreg等模块

# 一、模块规格

## 1. F_part

### 1.1 IFU模块

**端口定义：**

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Clk | Input | 1 | 时钟信号 |
| Reset | Input | 1 | 异步复位信号 |
| EN | In | 1 | 使能信号 |
| NPC | Input | [31:0] | 下一次PC值 |
| Instruction | Output | [31:0] | 当前指令 |
| PCOut | Output | [31:0] | 当前PC值 |

**具体功能：**

| 功能 | 描述 |
| --- | --- |
| 输出下一条指令 | 使能信号有效时，取出NPC对应指令 |
| 输出当前PC值 | |
| 复位 | reset信号变为1时，PC清零；以及取指地址异常时，Instr为0 |
| 取出评测机指令 | 我不是我没有别瞎说 |

### 1.2 F_Exc

**具体功能：**

检测`AdEL`类型异常

## 1to2. IFtoIDreg

## 2.D_part

### 2.1 GRF模块

**端口定义：**

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Clk | Input | 1 | 时钟信号 |
| Reset | Input | 1 | 异步复位信号 |
| WE | Input | 1 | 写使能端 |
| ReadAddr1 | Input | [4:0] | 读寄存器编号1 |
| ReadAddr2 | Input | [4:0] | 读寄存器编号2 |
| WriteAddr | Input | [4:0] | 写寄存器编号 |
| WData | Input | [31:0] | 写入数据 |
| RData1 | Output | [31:0] | 读寄存器值1 |
| RData2 | Output | [31:0] | 读寄存器值2 |

**具体功能：**

| 功能 | 描述 |
| --- | --- |
| 复位 | 当reset为1时，所有寄存器的值清零 |
| 读取数据 | RData1的值是寄存器编号为Read1的寄存器的值；RData2的值是寄存器编号为Read2的寄存器的值 |
| 写入数据 | 当写使能WE为1时，向编号为Write的寄存器写入WriteData |

## 2.2 Ext模块

**端口定义：**

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Imm16 | Input | [15:0] | 输入的16位立即数 |
| EXtCtrl | Input | [1:0] | 控制信号 |
| Imm32 | Output | [31:0] | 扩展结果 |

**具体功能：**

| 功能 | 描述 |
| --- | --- |
| 0扩展 | `Imm32={{16{0}},Imm16}` |
| 符号扩展 | `Imm32={{16{Imm16[15]}},Imm16}` |
| 把数加载到高位 | `Imm32={Imm16,{16{0}}}` |
| 1扩展 | `Imm32={{16{1}},Imm16} |

### 2.3 NPC

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| ifBr | I | 1 | 是否满足B类跳转条件 |
| isBr | I | 1 | 是否是Branch指令 |
| isJump | I | 1 | 是否是J类型指令 |
| isJr | I | 1 | 是否是Jr类型指令 |
| isERET | I | 1 | 是否是eret |
| BrImm | I | [31:0] | Br类型跳转立即数 |
| JImm | I | [31:0] | J类型跳转立即数 |
| JrImm | I | [31:0] | Jr类型跳转立即数 |
| PC | I | [31:0] | PC值 |
| EPC | I | [31:0] | 受害指令的PC值 |
| NPC | O | [31:0] | Next PC值 |

**具体功能：**

计算下一条PC值

### 2.4 BrCmp

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Instr | I | [31:0] | 指令 |
| RData1 | I | [31:0] | 读入数据1 |
| RData2 | I | [31:0] | 读入数据2 |
| ifBr | O | 1 | 是否满足B类跳转条件 |

**具体功能：**

计算是否满足B类跳转需满足的条件

### 2.5 CdtWECmp

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Instr | I | [31:0] | 指令 |
| RData1 | I | [31:0] | 读入数据1 |
| RData2 | I | [31:0] | 读入数据2 |
| ifBr | I | 1 | 是否满足B类跳转条件 |
| ifCdtWE | O | 1 | 是否满足条件写的条件 |

**具体功能：**

条件写类型指令，判断是否满足条件写要满足的条件

### 2.6 D_Exc

**具体功能：**

检测`RI`类型异常

## 2to3. IDtoEXreg

## 3. Ex_part

### 3.1 ALU

**端口定义：**

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| A | Input | [31:0] | 输入数据1 |
| B | Input | [31:0] | 输入数据2 |
| ALUOp | Input | [4:0] | 选择ALU功能 |
| Shamt | Input | [4:0] | 左移位数 |
| Overflow | Out | 1 | 是否运算溢出 |
| Result | Output | [31:0] | 运算结果 |

**具体功能：**

| 功能 | 描述 |
| --- | --- |
| 与 | `Result=A&B` |
| 或 | `Result=A\|B` |
| 异或 | `Result=A^B` |
| 减（比较） | `Result=A-B`；若`A==B,Zero=0` |
| 加 | `Result=A+B` |
| 逻辑左移 | `Result=B<<Shamt` |

### 3.2 MDU

**端口定义：**

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| clk | In | 1 | 时钟信号 |
| reset | In | 1 | 复位信号 |
| A | In | [31:0] | 输入数据1 |
| B | In | [31:0] | 输入数据2 |
| Start | In | 1 | 启动运算信号 |
| MDUOp | In | [4:0] | 乘除模块运算类型 |
| Result | Out | [31:0] | 取出hi或lo的值 |
| Busy | Out | 1 | 模拟延迟 |

**具体功能：**

执行`MULT、MULTU、DIV、DIVU、MTLO、MTHI`指令的乘除部分要求

### 3.3 E_Exc

**具体功能：**

根据ALU的溢出信号，检测是否为`Ov,AdEL,AdEs`类型的异常

## 3to4. EXtoMEMreg

## 4. MEM_part

### 4.1 Dm

**端口定义：**

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Clk | Input | 1 | 时钟信号 |
| Reset | Input | 1 | 异步复位信号 |
| WE | Input | 1 | 写使能信号 |
| isMemb | I | 1 | 对内存操作按字节 |
| isMemh | I | 1 | 对内存操作按半字 |
| Addr | Input | [4:0] | 写入地址 |
| WD | Input | [31:0] | 写入数据 |
| RD | Output | [31:0] | 读取数据 |

**具体功能：**

| 功能 | 描述 |
| --- | --- |
| 复位 | Reset 为 1 时，数据清 0 |
| 写入数据 | 当时钟上升沿时，如果写使能信号WE有效，就将WD写入到地址为Addr处 |
| 读取数据 | RD为地址Addr处的数据的值 |

### 4.2 M_Exc

**具体功能：**

检查`AdEL`,`AdES`类型异常

## 4to5. MEMtoWBreg

## 5. WB_part

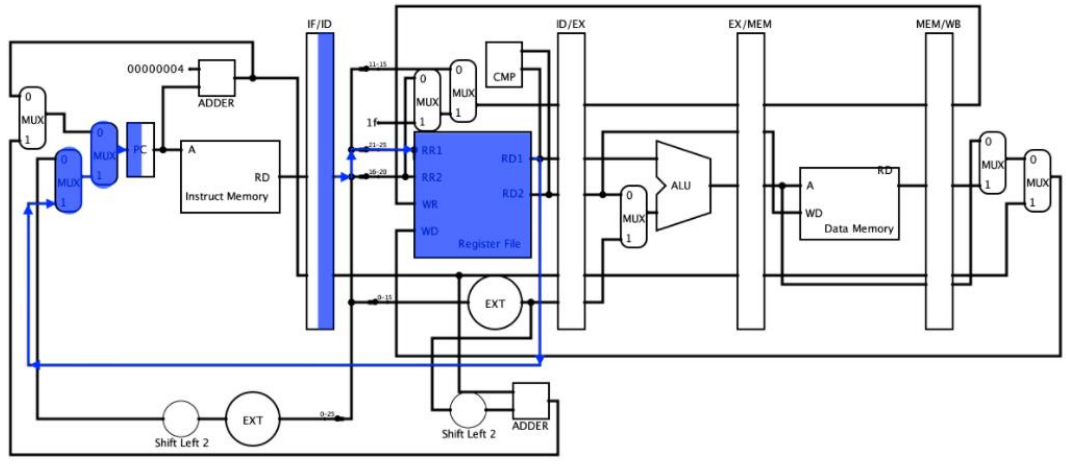
回写。

## 6. datapath

将前述5个模块实例化并连接

| 端口名 | 方向 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| clk | In | 1 | 时钟信号 |
| reset | In | 1 | 异步复位信号 |
| Trans_grf_Sel1 | In | | 转发更新RD1的MUX控制信号 |
| Trans_grf_Sel2 | In | | 转发更新RD2的MUX控制信号 |
| Trans_ALUIn_Sel1 | In | | 转发更新E_RD1的MUX控制信号 |
| Trans_ALUIn_Sel2 | In | | 转发更新E_RD2的MUX控制信号 |
| Trans_MemRD_Sel | In | | 转发更新M_RD2的MUX控制信号 |
| stall | In | | 暂停信号 |
| Tuse1 | Out | | D级指令的Tuse1，用于暂停判断 |
| Tuse2 | Out | | D级指令的Tuse2，用于暂停判断 |
| D_ReadA1 | Out | | D级指令要读取的寄存器1地址 |
| D_ReadA2 | Out | | D级指令要读取的寄存器2地址 |
| E_ReadA1 | Out | | E级指令要读取的寄存器1地址 |
| E_ReadA2 | Out | | E级指令要读取的寄存器2地址 |
| E_WriteA | Out | | E级指令写入寄存器的地址 |
| E_Tnew | Out | | E级指令的Tnew |
| M_ReadA2 | Out | | M级指令要读取的寄存器2地址 |
| M_WriteA | Out | | M级指令写入寄存器的地址 |
| M_Tnew | Out | | M级指令的Tnew |
| W_WriteA | Out | | W级指令写入寄存器的地址 |
| W_Tnew | Out | | W级指令的Tnew |

1. ID



## 7. Timer

外部设备，用以隔一段时间产生外部中断信号。

## 8. Bridge

实现CPU与外部设备沟通的系统桥。CPU需增加如下信号：

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| PrRD[31:0] | I | 从Bridge读入数据 |
| HWInt[7:2] | I | 6个硬件中断请求 |
| PrAddr[31:2] | O | 32位地址总线 |
| PrWD[31:0] | O | 输出至Bridge数据 |
| PrWE | O | CPU写使能 |

```
assign PrAddr = {MemAddr[31:2],2'b0};
assign PrWD = MemWData;
assign PrWE = MemWrite && (!IntExcReq) && (MemAddr>=`Timer0AddrMin);
```

Bridge：

```
assign DevAddr = PrAddr[31:2];
assign Dev0WE = (PrWE && HitDev0) ? 1 : 0;
assign Dev1WE = (PrWE && HitDev1) ? 1 : 0;
assign DevWD = PrWD;

assign PrRD = (HitDev0) ? Dev0RD :
              (HitDev1) ? Dev1RD :
              0;
```

**具体功能：**

1. 用来判断是否对Timer的寄存器进行写入，以及写入哪一个Timer。
2. 读取Timer中寄存器的数据传入到CPU中

## 9. CP0（Coprocessor 0）

0号协处理器。[ 1号协处理器：浮点协处理器 ]

- SR(`status Reister`)：储存处理器的控制信息
  - `SR = {16'b0, IM, 8'b0, EXL, IE};`
  - `[15:10] IM`：中断屏蔽功能
  - `EXL`：任何异常发生时置位，用来屏蔽中断信号
  - `IE`：全局的中断使能位
- Cause：获取处理器当前所处于的状态（什么原因导致的异常或是中断）
  - `Cause = {BD, 15'b0, hwint_pend, 3'b0, ExcCodeReg, 2'b0}`
  - `BD`：异常指令是否为分支延迟槽指令
  - `[15:10] hwint_pend`：待决断的中断
  - `[4:0] ExcCodeReg`：储存发生的异常

- EPC：获取被异常/中断的指令地址
  - `EPC = {EPCReg, 2'b0}`
  - `[31:2] EPCReg`：计算存储EPC；如果是分支延迟槽指令，则为受害指令的上一条指令对应PC；否则为受害指令的PC。
- PRId：读取处理器ID
  - my birthday　(:3_ヽ)_

输出信号：

- IntExcReq：中断请求信号
  - `IntReq = !EXL & IE & (|(HWInt & IM));`

    `ExcReq = !EXL & (|ExcCode);`

    `IntExcReq = IntReq | ExcReq;`
  - 该信号发生时，不能对DM写入，PC变为exception handler对应PC，并清空流水线，以及不发生阻塞
- EPC：EPC寄存器输出至NPC
- DOut：CP0寄存器的输出数据至通用寄存器（mfc0）
  - ```
    DOut = (ReadAddr==12) ? SR :
           (ReadAddr==13) ? Cause :
           (ReadAddr==14) ? EPC :
           (ReadAddr==15) ? PRId :
           0;
    ```

# 二、控制信号

## 1. 指令编码

| | addu | subu | ori | lw | sw | beq | lui | sll |
|---|---|---|---|---|---|---|---|---|
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000000 |
| func | 100001 | 100011 | n/a | n/a | n/a | n/a | n/a | 000000 |
| | j | jal | jr | | | | | |
| op | 000010 | 000011 | 000000 | | | | | |
| func | n/a | n/a | 001000 | | | | | |

## 2. 主控制信号真值表
    ```

| | RegWrite | EXTCtrl | ALUSrc | ALUOp | WE | MemToReg | isBr | isJump | isJR | MemType | MemSign | MDUStart | MDUOp | EResultSel | ConditionWE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 作用 | GRF写使能 | 扩展器控制 | MUX选择ALU端口B的输入 | ALU功能 | DM写使能 | MUX选择写回数据（有修改） | 是否是beq指令 | 是否j类跳转 | 是否JR | 对Mem的操作位宽 | 对Mem符号操作类型 | MDU模块运算启动信号 | MDU模块运算类型 | 选择ALU的MDU的运算结果 | 是否为条件写类型 |
| addu | 1 | x | 0 | add | 0 | 0 | 0 | 0 | 0 | | | | | | |
| subu | 1 | x | 0 | sub | 0 | 0 | 0 | 0 | 0 | | | | | | |
| ori | 1 | 00 | 1 | or | 0 | 0 | 0 | 0 | 0 | | | | | | |
| lw | 1 | 01 | 1 | add | 0 | 1 | 0 | 0 | 0 | word | Unsigned | | | | |
| sw | 0 | 01 | 1 | add | 1 | x | 0 | 0 | 0 | word | Unsigned | | | | |
| beq | 0 | x | 0 | sub | 0 | x | 1 | 0 | 0 | | | | | | |
| lui | 1 | 10 | 1 | add | 0 | 0 | 0 | 0 | 0 | | | | | | |
| nop | 0 | x | x | x | 0 | x | 0 | 0 | 0 | | | | | | |
| sll | 1 | x | 0 | sll | 0 | 0 | 0 | 0 | 0 | | | | | | |
| j | 0 | sign | 1 | add | 0 | x | 0 | 1 | 0 | | | | | | |
| jal | 1 | x | x | x | 0 | 10 | 0 | 1 | 0 | | | | | | |
| jr | 0 | x | 0 | x | 0 | 0 | 0 | 0 | 1 | | | | | | |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| RegDst | Rt | Rd | 31 | x |

注：在本人目前的设计中，写入地址在ATControl中产生，因此RegDst暂无用处。

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| ExtCtrl | 无符号扩展 | 符号扩展 | 加载到高位 | 1扩展 |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| MemtoReg | ALU结果 | DM结果 | x | x |

update：为将jal一类指令的Tnew缩短，本人将PC+8提前到了D_part，通过ImmSel选择输入ALU的立即数，ALU方法为加载B口数据，以及注意搭配ALUSrc使用。

# 3. AT信号译码

Tuse：这条指令位于D级的时候，再经过多少个时钟周期就必须要使用相应的数据。

Tnew：位于某个流水级的某个指令，它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里。

| | 转发接受位点 | Tuse | 转发输出位点 | Tnew(D_part) | WriteAddr |
|---|---|---|---|---|---|
| addu\|subu | E@ALUIn1&In2 | rs=rt=1 | EMreg | 2 | rd |
| sll | E@ALUIn2 | rt=1 | EMreg | 2 | rd |
| mult | E@MDUIn1&In2 | rs=rt=1 | | | |
| mfhi | | | EMreg | 2 | rd |
| mthi | E@MDUIn1 | rs=1 | | | |
| jr | D@jrImm(RD1) | rs=0 | | `n/a` | |
| lui | | `n/a` | DEreg | 1 | rt |
| ori | E@ALUIn1 | rs=1 | EMreg | 2 | rt |
| lw | E@ALUIn1 | rs=1 | MWreg | 3 | rt |
| sw | E@ALU<br>M@DmIn | rs=1,rt=2 | | `n/a` | |
| beq | D@cmpIn1&In2 | rs=rt=0 | | `n/a` | |
| j | | `n/a` | | `n/a` | |
| jal | | `n/a` | DEreg | 1 | ra |

类`addu`：`add,sub,ans,or,xor,nor,sllv,srlv,srav,slt,sltu`;

类`sll`：`srl,sra`;

类`mult`：`multu,div,divu`;

类`mfhi`：`mflo`;

类`mthi`：`mtlo`;

# 4. 转发控制

需要设置五个MUX

1. 更新RData1

$$MUX1\begin{cases}GRFRdata1\\DE\_reg转发过来的数据\\EM\_reg转发过来的数据\\MW\_reg转发过来的数据\end{cases}$$

MUX控制信号生成：

$$if\ 读取地址为0,Sel=0$$
$$else\ if\ 读取地址与E级要写入地址相同，且E级写使能为1且此时要写入数据已产生,Sel=1$$
$$else\ if\ ...M...,Sel=2$$
$$else\ if\ ...W...,Sel=3<就近原则>$$
$$else\ 没有需要转发的数据,Sel=0$$

2. 更新RData2

   与更新RData1类似

3. 更新经DEreg流水线寄存器流水过来的E_RData1

$$E\_Trans\_RData1\begin{cases}E\_RData1\\EM\_reg\text{转发过来的数据}(E\text{级的}ALUResult\text{存到}reg\text{中的数据})\\MW\_reg\text{转发过来的数据}(M\text{级选择好的要写回}GRF\text{的数据，存到}reg\text{中})\end{cases}$$

[注：因为在D级更新RData1时，后续的阶段可能尚未计算出转发的结果，因此需要在E级继续转发更新]

MUX控制信号生成：

$$if\ \text{读取地址为}0, Sel=0$$
$$else\ if\ \text{读取地址与}M\text{级要写入地址相同，且}M\text{级写使能为}1\text{且此时要写入数据已产生}, Sel=1$$
$$else\ if\ \ldots W\ldots, Sel=2 < \text{就近原则} >$$
$$else\ \text{没有需要转发的数据}, Sel=0$$

4. 更新经DEreg流水线寄存器流水过来的E_RData2

   与E_RData1类似

   注意向后流水的时候是更新后的E_Data2(E_Trans_RData2)

5. 更新写入dm的数据

$$MemWData\begin{cases}GRFRData2(E\_Trans\_RData2\text{经}EM\text{寄存器流过来的数据})\\W\_GRFWData\end{cases}$$

MUX控制信号生成：

$$if\ \text{读取地址为}0, Sel=0$$
$$else\ if\ \text{读取地址与}W\text{级要写入地址相同，且}W\text{级写使能为}1\text{且此时要写入数据已产生}, Sel=1$$
$$else\ \text{没有需要转发的数据}, Sel=0$$

## 5. 阻塞控制

为了方便处理，下述暂停是指将指令暂停在D级。暂停操作：

- 冻结PC的值
- 冻结F/D级流水线寄存器的值
- 将D/E级流水线寄存器清零（这等价于插入了一个nop指令）

当D级指令读取寄存器的地址与E级或M级的指令写入寄存器的地址相等且不为0，且D级指令的Tuse小于对应E级或M级指令的Tnew时，我们就需要在D级暂停指令。在其他情况下，数据冒险均可通过转发机制解决。

$$if\ \text{当前处于}E\text{级的指令与}D\text{级指令构成转发关系}(\text{写后读}), Tnew1=E\_Tnew$$
$$else\ if\ \ldots D\ldots, Tnew1=D\_Tnew$$
$$else\ if\ W\ldots\quad [Tnew1\text{是}rs\text{对应的}Tnew]$$
$$Tnew2\text{同理；}$$

阻塞信号：`stall=(D_ReadA1!=0 && Tnew1>Tuse1) || (D_ReadA2!=0 && Tnew2>Tuse2)`

# 三、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？）

   硬件类接口：同一计算机不同功能层之间的通信规则称为硬件接口。

   软件类接口： 对协定进行定义的引用类型、其他类型实现接口，以保证它们支持某些操作。接口指定必须由类提供的成员或实现它的其他接口。与类相似，接口可以包含方法、属性、索引器和事件作为成员。

   而本课程中的硬件\软件接口是指程序以及异常处理部分在软件、硬件中的一些协议约定，从而使得两部分可以分开工作。

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。

   位于主存，而后CPU通过cache与主存交换数据。

3. BE 部件对所有的外设都是必要的吗?

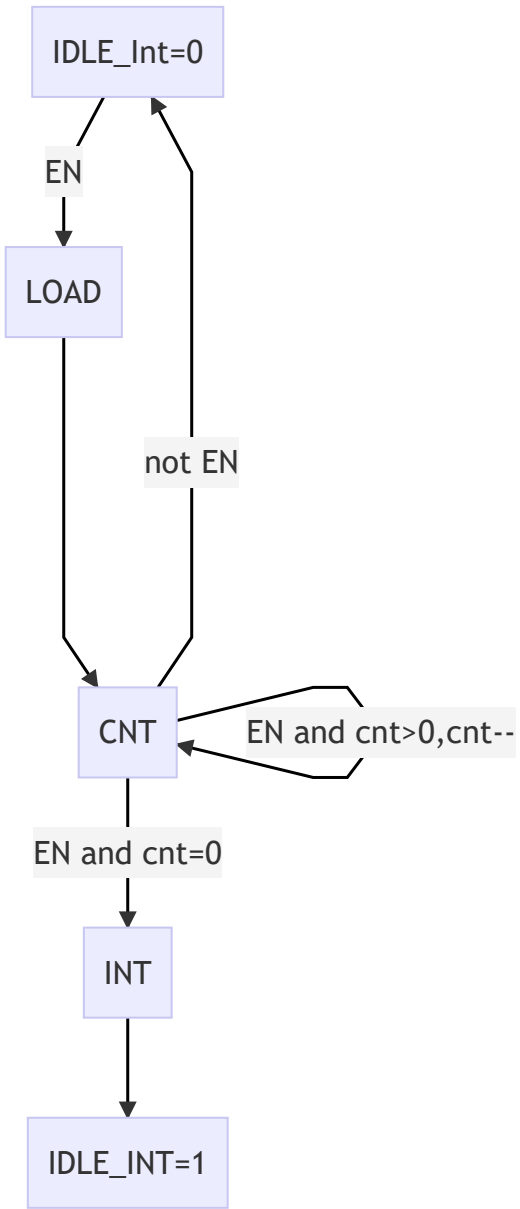
   不是。若外设以字来读写，就无需BE。而以字节或半字读写就需要用BE。

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

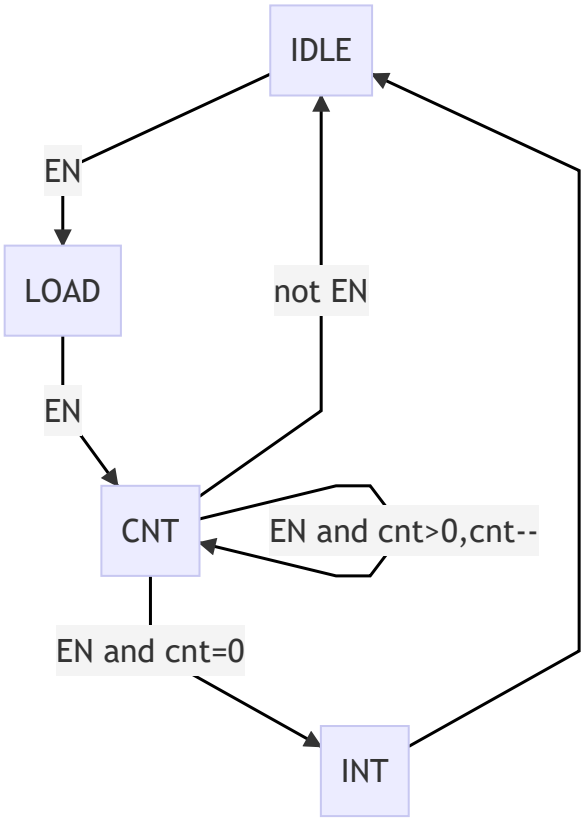
   方式0：产生持续的中断信号；

   方式1：产生持续一个周期的中断信号，而后转移到IDLE，清零

   方式0：




   方式1：




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。）

```
.ktext 0x4180
ori $t0, $0, 9
ori $t1, 0x7f00
sw  $t0, 0($t1)
eret
.text
ori $t1, 0x7f00
ori $t0, $0, 0xfc01
mtc0    $t0, $12
ori $t0, $0, 9
sw  $t0, 0($t1)
ori $t0, $0, 100
sw  $t0, 4($t1)
```

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？

    pic（programmable interrupt controller）即可编程中断控制器的芯片组，负责监控键盘鼠标等外部设备。一旦检测到外设中断，pic就向CPU发出中断，CPU进入中断处理程序来根据指令处理中断。

# 四、 测试

延迟槽指令测试：

```
li $5,0x3456789a
li $6,0x12349876
ori $7,$0,0xfc01
mtc0 $7,$12
mult $5,$6
nop
beq $5,$5,next
mflo $7
xori $1,$1,1
next:
mfhi $8
```

中断:

```
ori $7,$0,0xfc01
mtc0 $7,$12
li $5,0x98765432
li $6,0xfedcba98
div $5,$6
mflo $6
label:
beq $0,$0,label
nop
```

```
li $5,0x98765432
li $6,0xfedcba98
ori $7,$0,0xfc01
mtc0 $7,$12
div $5,$6
mflo $6
label:
beq $0,$0,label
nop
```

```
ori $7,$0,0xfc01
mtc0 $7,$12
li $5,0x5432
li $6,0xba98
mflo $6
label:
beq $0,$0,label
nop
```