计算机组成原理实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的流水线 MIPS-CPU，支持的指令集包含{lw、sw、lb、lbu、sb、lh、lhu、sh、add、addu、sub、subu、and、or、slt、nor、xor、sll、srl、sra、sllv、srlv、srav、sltu、beq、bne、bgtz、blez、bltz、bgez、addi、addiu、andi、ori、xori、lui、slti、sltiu、j、jr、jal、jalr、mult、multu、div、divu、mflo、mfhi、mtlo、mthi、mfc0、mtc0、eret}共 53 条。CPU 主要包含了 AluSelect、ArithmeticLogicalUnit、BitExtender、CentralProcessingUnit、CompareUnit、ControlUnit 、 Coprocessor 、 DataMemory 、 DoubleForwardSelect 、 EXErrorJudgeUnit 、 EightDataToRegSelect、ExtenderForLoad、FlowReg_EX_MEM、FlowReg_ID_EX、FlowReg_IF_ID、FlowReg_MEM_WB、FourDataToRegSelect、InstructionMemory、JumpJudgeUnit、MulDivSolver、ProgramCounter、RegDstSelect、RegisterFile、RiskSolveUnit、SingleForwardSelect、StandardTimer、SystemBridge、TwoDataToRegSelect 共计 28 个模块。

（二）关键模块定义

1. ALU 输入数据选择器 AluSelect

模块定义

```
module AluSelect(
    input [31:0] regReadData1
    input [31:0] regReadData2
    input [31:0] immAfterExtend
    input [1:0] aluInputSelect
    output [31:0] aluInputA
    output [31:0] aluInputB
);
```

名称	功能
input [31:0] regReadData1	来自 reg 的第一个数据
input [31:0] regReadData2	来自 reg 的第二个数据
input [31:0] immAfterExtend	经过扩展的 32 位立即数
input [1:0] aluInputSelect	输出选择信号，两位分别作用于 regReadData1 和 regReadData2
output [31:0] aluInputA	经过选择输出的 Alu 的第一个操作数
output [31:0] aluInputB	经过选择输出的 Alu 的第二个操作数

2. 算术逻辑单元 ArithmeticLogicalUnit

模块定义

```
module ArithmeticLogicalUnit(
   input [31:0] aluInputA
   input [31:0] aluInputB
   input [3:0] aluOperation
   output [31:0] aluOutput
   output aluOverFlow
);
```

名称	功能
input [31:0] aluInputA	Alu 的第一个操作数
input [31:0] aluInputB	Alu 的第二个操作数
input [3:0] aluOperation	Alu 操作选择信号
output [31:0] aluOutput	Alu 运算结果
output aluOverFlow	Alu 溢出信号

操作定义

```
`define aluOfAnd 4'b0000
`define aluOfOr  4'b0001
`define aluOfNor 4'b0010
```

```
`define aluOfXor 4'b0011
`define aluOfSlt 4'b0100
`define aluOfSltu4'b0101
`define aluOfAdd 4'b0110
`define aluOfSub 4'b0111
`define aluOfSll 4'b1000
`define aluOfSrl 4'b1001
`define aluOfSra 4'b1010
`define aluOfClo 4'b1011
`define aluOfClz 4'b1100
```

操作名	选择信号	功能
aluOfAnd	4'b0000	A 与 B
aluOfOr	4'b0001	A 或 B
aluOfNor	4'b0010	A 或非 B
aluOfXor	4'b0011	A 异或 B
aluOfSlt	4'b0100	A 补码比较小于 B 置为 1
aluOfSltu	4'b0101	A 原码比较小于 B 置为 1
aluOfAdd	4'b0110	A 加 B
aluOfSub	4'b0111	A 减 B
aluOfSll	4'b1000	B 逻辑左移 A 的后五位
aluOfSrl	4'b1001	B 逻辑右移 A 的后五位
aluOfSra	4'b1010	B 算术右移 A 的后五位
aluOfClo	4'b1011	计算 A 的前导 1 的个数
aluOfClz	4'b1100	计算 A 的前导 0 的个数

3. 数据扩展器 BitExtender

模块定义

```
module BitExtender(
```

```
    input [1:0] extendMood
    input [15:0] immToExtend
    output [31:0] immAfterExtend
);
```

名称	功能
input [1:0] extendMood	数据扩展方式选择
input [15:0] immToExtend	待扩展的立即数
output [31:0] immAfterExtend	扩展得到的 32 位数据

4. 数据比较单元 CompareUnit

模块定义

```
module CompareUnit(
    input [31:0] cmpInputA
    input [31:0] cmpInputB
    input [3:0] cmpOperation
    output toBranch
);
```

名称	功能
input [31:0] cmpInputA	进行比较的第一个操作数
input [31:0] cmpInputB	进行比较的第二个操作数
input [3:0] cmpOperation	比较操作选择信号
output toBranch	跳转标志信号

操作定义

```
`define cmpOfBeq    4'b0000
`define cmpOfBne    4'b0001
`define cmpOfRsBgez 4'b0010
`define cmpOfRsBgtz 4'b0011
`define cmpOfRsBlez 4'b0100
```

```
`define cmpOfRsBltz 4'b0101
`define cmpOfRsBeqz 4'b0110
`define cmpOfRsBnez 4'b0111
`define cmpOfRtBgez 4'b1000
`define cmpOfRtBgtz 4'b1001
`define cmpOfRtBlez 4'b1010
`define cmpOfRtBltz 4'b1011
`define cmpOfRtBeqz 4'b1100
`define cmpOfRtBnez 4'b1101
```

操作名	选择信号	功能
cmpOfBeq	4'b0000	A 等于 B 置 1
cmpOfBne	4'b0001	A 不等于 B 置 1
cmpOfRsBgez	4'b0010	A 大于等于 0 置 1
cmpOfRsBgtz	4'b0011	A 大于 0 置 1
cmpOfRsBlez	4'b0100	A 小于等于 0
cmpOfRsBltz	4'b0101	A 小于 0 置 1
cmpOfRsBeqz	4'b0110	A 等于 0 置 1
cmpOfRsBnez	4'b0111	A 不等于置 1
cmpOfRtBgez	4'b1000	B 大于等于 0 置 1
cmpOfRtBgtz	4'b1001	B 大于 0 置 1
cmpOfRtBlez	4'b1010	B 小于等于 0
cmpOfRtBltz	4'b1011	B 小于 0 置 1
cmpOfRtBeqz	4'b1100	B 等于 0 置 1
cmpOfRtBnez	4'b1101	B 不等于置 1

5. 控制单元 ControlUnit

模块定义

```
module ControlUnit(
```

```
    input [31:0] currentCommand
    output [1:0] extendMoodInID
    output [1:0] aluInputSelectInID
    output [3:0] aluOperationInID
    output cuOverFlowInID
    output cuRegBranchInID
    output cuMdBranchInID
    output cuDmBranchInID
    output [3:0] cmpOperationInID
    output memWriteEnabledInID
    output [2:0] loadWriteMoodInID
    output [1:0] regDstSelectInID
    output regWriteEnabledInID
    output regConditionMoveInID
    output exConditionMoveInID
    output dmConditionMoveInID
    output writeEnabledOfCP0InID
    output [1:0] pcControlInID
    output [5:0] dataToRegSelectInID
    output mulOrDivInID
    output [3:0] mulDivOperationInID
    output knowCommandInID
    output toFlushCP0InID
    output [2:0] tUseOf2521InID
    output [2:0] tUseOf2016InID
    output [2:0] tNewInID
);
```

名称	功能

input [31:0] currentCommand	当前需要译码的指令
output [1:0] extendMoodInID	跳转模式选择信号，仅作用于 ID
output [1:0] aluInputSelectInID	Alu 输入选择信号，流水到 EX 使用
output [3:0] aluOperationInID	Alu 操作选择信号，流水到 EX 使用
output cuOverFlowInID	溢出判断信号，流水到 EX 使用
output cuRegBranchInID	ID 阶段跳转选择信号，仅作用于 ID
output cuMdBranchInID	EX 阶段跳转选择信号，流水到 EX 使用
output cuDmBranchInID	MEM 阶段跳转选择信号，流水到 MEM 使用
output [3:0] cmpOperationInID	比较器操作选择信号，视 cuRegBranchInID 和 cuDmBranchInID 在 ID 或 EX 或 MEM 使用
output memWriteEnabledInID	DM 写使能信号，流水到 MEM 使用
output [2:0] loadWriteMoodInID	DM 读写模式选择信号，流水到 MEM 和 WB 使用
output [1:0] regDstSelectInID	寄存器写地址选择信号，仅作用于 ID
output regWriteEnabledInID	寄存器写使能信号，流水到 WB 使用
output regConditionMoveInID	ID 阶段条件写入信号，仅作用于 ID
output exConditionMoveInID	EX 阶段条件写入信号，流水到 EX 使用
output dmConditionMoveInID	MEM 阶段条件写入信号，流水到 MEM 使用
output writeEnabledOfCP0InID	CP0 写使能信号，流水到 EX 使用
output [1:0] pcControlInID	PC 控制信号，视 cuRegBranchInID 和 cuDmBranchInID 在 ID 或 MEM 使用
output [5:0] dataToRegSelectInID	写入寄存器的数据选择信号，流水到 MEM 逐级使用
output mulOrDivInID	乘除法器相关指令标记，为乘除法器相关指令时为 1
output [3:0] mulDivOperationInID	乘除法器操作控制信号
output knowCommandInID	指令合法信号，遇到 unknown 指令时为 0
output toFlushCP0InID	eret 刷新信号，为 1 时跳转到 epc
output [2:0] tUseOf2521InID	Rs 需要使用的最晚时间
output [2:0] tUseOf2016InID	Rd 需要使用的最晚时间

output [2:0] tNewInID	当前指令产生的数据可用的最早时间

6. 零号协处理器 Coprocessor0

模块定义

```
module Coprocessor0(
    input clk
    input reset
    input flush
    input debugMood
    input [4:0] readOrWriteAddrInCP0
    input writeEnabledOfCP0
    input [31:0] commandAddrToCP0
    input [5:0] interruptRequest
    input [4:0] errorCode
    input [31:0] dataWriteToCP0
    input isInBranchDelay
    output [31:0] commandAddrFromCP0
    output [31:0] dataGetFromCP0
    output jumpTo4180
);
```

名称	功能
input clk	时钟信号
input reset	重置信号
input flush	刷新信号，表示进入内核态
input debugMood	全局 debug 信号，为 1 时行为切换到和 Mars 一致
input [4:0] readOrWriteAddrInCP0	读写寄存器的地址
input writeEnabledOfCP0	CP0 写使能信号
input [31:0] commandAddrToCP0	宏观指令地址，用以存入 epc

input [5:0] interruptRequest	六位外部中断请求
input [4:0] errorCode	当前指令异常类型码
input [31:0] dataWriteToCP0	要存入 CP0 的数据
input isInBranchDelay	当前指令是延迟槽指令
output [31:0] commandAddrFromCP0	epc 寄存器中存储的指令地址
output [31:0] dataGetFromCP0	CP0 输出的对应寄存器地址的数据
output jumpTo4180	异常中断响应信号，用以进入内核态

7. 二路寄存器写入数据选择器 TwoDataToRegSelect

模块定义

```
module FourDataToRegSelect (
   input [31:0] dataToSelectBy0
   input [31:0] dataToSelectBy1
   input dataToRegSelect
   output [31:0] dataWriteToReg
);
```

名称	功能
input [31:0] dataToSelectBy0	选择信号为 0 时要输出的数据
input [31:0] dataToSelectBy1	选择信号为 1 时要输出的数据
input dataToRegSelect	数据选择信号
output [31:0] dataWriteToReg	经过选择得到的数据

8. 四路寄存器写入数据选择器 FourDataToRegSelect

模块定义

```
module FourDataToRegSelect (
   input [31:0] dataToSelectBy0
   input [31:0] dataToSelectBy1
   input [31:0] dataToSelectBy2
   input [31:0] dataToSelectBy3
```

```
    input [1:0] dataToRegSelect
    output [31:0] dataWriteToReg
);
```

名称	功能
input [31:0] dataToSelectBy0	选择信号为 0 时要输出的数据
input [31:0] dataToSelectBy1	选择信号为 1 时要输出的数据
input [31:0] dataToSelectBy2	选择信号为 2 时要输出的数据
input [31:0] dataToSelectBy3	选择信号为 3 时要输出的数据
input [1:0] dataToRegSelect	数据选择信号
output [31:0] dataWriteToReg	经过选择得到的数据

9. 八路寄存器写入数据选择器 EightDataToRegSelect

模块定义

```
module EightDataToRegSelect(
    input [31:0] dataToSelectBy0
    input [31:0] dataToSelectBy1
    input [31:0] dataToSelectBy2
    input [31:0] dataToSelectBy3
    input [31:0] dataToSelectBy4
    input [31:0] dataToSelectBy5
    input [31:0] dataToSelectBy6
    input [31:0] dataToSelectBy7
    input [2:0] dataToRegSelect
    output [31:0] dataWriteToReg
);
```

名称	功能
input [31:0] dataToSelectBy0	选择信号为 0 时要输出的数据
input [31:0] dataToSelectBy1	选择信号为 1 时要输出的数据

input [31:0] dataToSelectBy2	选择信号为 2 时要输出的数据
input [31:0] dataToSelectBy3	选择信号为 3 时要输出的数据
input [31:0] dataToSelectBy4	选择信号为 4 时要输出的数据
input [31:0] dataToSelectBy5	选择信号为 5 时要输出的数据
input [31:0] dataToSelectBy6	选择信号为 6 时要输出的数据
input [31:0] dataToSelectBy7	选择信号为 7 时要输出的数据
input [2:0] dataToRegSelect	数据选择信号
output [31:0] dataWriteToReg	经过选择得到的数据

10. 单数据前推选择器 SingleForwardSelect

模块定义

```
module SingleForwardSelect(
   input [31:0] srcDataA
   input [31:0] srcDataB
   input [31:0] dataCanUse
   input [1:0] dataForwardSelect
   output [31:0] dataASelected
   output [31:0] dataBSelected
);
```

名称	功能
input [31:0] srcDataA	第一个原始数据
input [31:0] srcDataB	第二个原始数据
input [31:0] dataCanUse	可供前推的数据
input [1:0] dataForwardSelect	前推数据选择信号，[1]和[0]分别作用于第一个原始数据和第二个原始数据
output [31:0] dataASelected	经过前推的第一个数据
output [31:0] dataBSelected	经过前推的第二个数据

11. 双数据前推选择器 DoubleForwardSelect

模块定义

```
module DoubleForwardSelect(
    input [31:0] srcDataA
    input [31:0] srcDataB
    input [31:0] dataCanUseEarlier
    input [31:0] dataCanUseLater
    input [3:0] dataForwardSelect
    output [31:0] dataASelected
    output [31:0] dataBSelected
);
```

名称	功能
input [31:0] srcDataA	第一个原始数据
input [31:0] srcDataB	第二个原始数据
input [31:0] dataCanUseEarlier	来自早一级的前推数据
input [31:0] dataCanUseLater	来自晚一级的前推数据
input [3:0] dataForwardSelect	前推数据选择信号，[3:2]和[1:0]分别作用于第一个原始数据和第二个原始数据
output [31:0] dataASelected	经过前推的第一个数据
output [31:0] dataBSelected	经过前推的第二个数据

12. 跳转指令判断器 JumpJudgeUnit

模块定义

```
module JumpJudgeUnit(
    input [31:0] currentCommand
    output isJumpCommand
);
```

名称	功能
input [31:0] currentCommand	当前指令数据
output isJumpCommand	跳转判断信号，为 1 则是跳转指令

13. 回写数据扩展器 ExtenderForLoad

模块定义

```
module ExtenderForLoad(
   input [1:0] addrOfDataInWB
   input [3:0] loadWriteMood
   input [31:0] dataToExtend
   output [31:0] dataExtended
);
```

名称	功能
input [1:0] addrOfDataInWB	MEM 级数据地址的低两位
input [3:0] loadWriteMood	扩展模式选择信号
input [31:0] dataToExtend	需要进行扩展的数据
output [31:0] dataExtended	进行扩展之后的数据

14. 乘除法运算单元 MulDivSolver

模块定义

```
module MulDivSolver(
   input clk
   input reset
   input [3:0] mulDivOperation
   input [31:0] mulDivInputA
   input [31:0] mulDivInputB
   output busySignal
   output startSignal
   output [31:0] registerHI
```

```
    output [31:0] registerLO
);
```

名称	功能
input clk	时钟信号
input reset	重置信号
input [3:0] mulDivOperation	乘除法器操作选择信号
input [31:0] mulDivInputA	输入乘除法器的第一个操作数
input [31:0] mulDivInputB	输入乘除法器的第二个操作数
output busySignal	忙碌信号，表示当前乘除法器正在计算
output startSignal	开始信号，表示当前乘除法器开始计算
output [31:0] registerHI	乘除法器 HI 寄存器的中值
output [31:0] registerLO	乘除法器 LO 寄存器中的值

操作定义

```
`define mdmthi    4'b0001
`define mdmtlo    4'b0010
`define mddiv     4'b0011
`define mddivu    4'b0100
`define mdmult    4'b0101
`define mdmultu   4'b0110
`define mdmadd    4'b0111
`define mdmaddu   4'b1000
`define mdmsub    4'b1001
`define mdmsubu   4'b1010
```

操作名	选择信号	功能
`define mdmthi	4'b0001	将输入数据 A 存入 HI
`define mdmtlo	4'b0010	将输入数据 A 存入 LO
`define mddiv	4'b0011	进行符号除法输入数据 A 除输入数据 B，商存入 LO 余数存入 HI

`define mddivu	4'b0100	进行无符号除法输入数据 A 除输入数据 B，商存入 LO 余数存入 HI
`define mdmult	4'b0101	进行符号乘法输入数据 A 乘输入数据 B，结果存入{HI,LO}
`define mdmultu	4'b0110	进行无符号乘法输入数据 A 乘输入数据 B，结果存入{HI,LO}
`define mdmadd	4'b0111	{HI,LO}加符号乘法输入数据 A 除乘入数据 B，结果存入{HI,LO}
`define mdmaddu	4'b1000	{HI,LO}加无符号乘法输入数据 A 乘输入数据 B，结果存入{HI,LO}
`define mdmsub	4'b1001	{HI,LO}减符号乘法输入数据 A 乘输入数据 B，结果存入{HI,LO}
`define mdmsubu	4'b1010	{HI,LO}减无符号乘法输入数据 A 乘输入数据 B，结果存入{HI,LO}

15. 指令计数器 ProgramCounter

模块定义

```
module ProgramCounter(
    input clk
    input reset
    input pcEnabled
    input [1:0] pcControlInID
    input [1:0] pcControlInEX
    input [1:0] pcControlInMEM
    input toBranchInID
    input cuRegBranchInID
    input toBranchInEX
    input cuMdBranchInEX
```

```
    input toBranchInMEM
    input cuDmBranchInMEM
    input [31:0] branchAddrInID
    input [31:0] branchAddrInEX
    input [31:0] branchAddrInMEM
    input [25:0] jumpAddrFromImm
    input [31:0] jumpAddrFormReg1InID
    input [31:0] jumpAddrFormReg2InID
    input [31:0] jumpAddrFormReg1InEX
    input [31:0] jumpAddrFormReg2InEX
    input [31:0] jumpAddrFormReg1InMEM
    input [31:0] jumpAddrFormReg2InMEM
    input jumpToAddrInCP0
    input jumpToAddrIn4180
    input [31:0] commandAddrFromCP0
    output [31:0] currentCommandAddr
    output [31:0] nextCommandAddr
);
```

名称	功能
input clk	时钟信号
input reset	重置信号
input pcEnabled	使能信号
input [1:0] pcControlInID	ID 阶段的 PC 控制信号
input [1:0] pcControlInEX	EX 阶段的 PC 控制信号
input [1:0] pcControlInMEM	MEM 阶段的 PC 控制信号
input toBranchInID	ID 阶段跳转指令判断信号
input cuRegBranchInID	ID 阶段分支选择信号
input toBranchInEX	EX 阶段跳转指令判断信号

input cuMdBranchInEX	EX 阶段分支选择信号
input toBranchInMEM	MEM 阶段跳转指令判断信号
input cuDmBranchInMEM	MEM 阶段分支选择信号
input [31:0] branchAddrInID	ID 阶段分支地址
input [31:0] branchAddrInEX	EX 阶段分支地址
input [31:0] branchAddrInMEM	MEM 阶段分支地址
input [25:0] jumpAddrFromImm	立即数跳转地址
input [31:0] jumpAddrFormReg1InID	ID 阶段来自 Rs 的跳转地址
input [31:0] jumpAddrFormReg2InID	ID 阶段来自 Rt 的跳转地址
input [31:0] jumpAddrFormReg1InEX	EX 阶段来自 Rs 的跳转地址
input [31:0] jumpAddrFormReg2InEX	EX 阶段来自 Rt 的跳转地址
input [31:0] jumpAddrFormReg1InMEM	MEM 阶段来自 Rs 的跳转地址
input [31:0] jumpAddrFormReg2InMEM	MEM 阶段来自 Rt 的跳转地址
input jumpToAddrInCP0	地址选择信号，为 1 时选择 epc 中的地址
input jumpToAddrIn4180	地址选择信号，为 1 时跳转到 0x4180
input [31:0] commandAddrFromCP0	epc 中存储的指令地址
output [31:0] currentCommandAddr	当前指令地址
output [31:0] nextCommandAddr	下一条指令地址

16. 寄存器写入地址选择器 RegDstSelect

模块定义

```
module RegDstSelect(
   input [4:0] currentCommand2016
   input [4:0] currentCommand1511
   input [1:0] regDstSelect
   output [4:0] regFinalDst
);
```

名称	功能

input [4:0] currentCommand2016	当前指令的 Rt 部分
input [4:0] currentCommand1511	当前指令的 Rd 部分
input [1:0] regDstSelect	寄存器写入地址选择
output [4:0] regFinalDst	经过选择的写入地址

17. 寄存器堆 RegisterFile

模块定义

```
module RegisterFile(
   input clk
   input reset
   input regWriteEnabled
   input [4:0] regReadAddr1
   input [4:0] regReadAddr2
   input [4:0] regWriteAddr
   input [31:0] dataWriteToReg
   input [31:0] currentCommandAddr
   output [31:0] regReadData1
   output [31:0] regReadData2
);
```

名称	功能
input clk	时钟信号
input reset	重置信号
input regWriteEnabled	数据写使能信号
input [4:0] regReadAddr1	第一个输出数据的地址
input [4:0] regReadAddr2	第二个输出数据的地址
input [4:0] regWriteAddr	数据写入地址
input [31:0] dataWriteToReg	写入寄存器堆的数据
input [31:0] currentCommandAddr	当前指令地址，用来按要求输出结果

output [31:0] regReadData1	第一个地址对应的数据
output [31:0] regReadData2	第二个地址对应的数据

18. 冒险处理单元 RiskSolveUnit

模块定义

```
module RiskSolveUnit(
    input [2:0] tNewInEX
    input [2:0] tNewInMEM
    input [2:0] tUseOf2521InID
    input [2:0] tUseOf2016InID
    input [4:0] currentCommand2521InID
    input [4:0] currentCommand2016InID
    input [4:0] currentCommand2521InEX
    input [4:0] currentCommand2016InEX
    input [4:0] currentCommand2521InMEM
    input [4:0] currentCommand2016InMEM
    input regWriteEnabledInEX
    input regConditionMoveInEX
    input regWriteEnabledInMEM
    input regConditionMoveInMEM
    input regWriteEnabledInWB
    input [4:0] regFinalDstInEX
    input [4:0] regFinalDstInMEM
    input [4:0] regFinalDstInWB
    input toMulOrDivInID
    input runMulOrDivInEX
    input errorOccored
    output [3:0] forwardInID
    output [3:0] forwardInEX
```

```
    output [1:0] forwardInMEM
    output stallAndFlush
);
```

名称	功能
input [2:0] tNewInEX	EX 阶段的 Tnew
input [2:0] tNewInMEM	MEM 阶段的 Tnew
input [2:0] tUseOf2521InID	ID 阶段 Rs 的 Tuse
input [2:0] tUseOf2016InID	ID 阶段 Rt 的 Tuse
input [4:0] currentCommand2521InID	ID 阶段指令的 Rs 部分
input [4:0] currentCommand2016InID	ID 阶段指令的 Rt 部分
input [4:0] currentCommand2521InEX	EX 阶段指令的 Rs 部分
input [4:0] currentCommand2016InEX	EX 阶段指令的 Rt 部分
input [4:0] currentCommand2521InMEM	MEM 阶段指令的 Rs 部分
input [4:0] currentCommand2016InMEM	MEM 阶段指令的 Rt 部分
input regWriteEnabledInEX	EX 阶段寄存器写使能信号
input regConditionMoveInEX	EX 阶段寄存器条件写信号
input regWriteEnabledInMEM	MEM 阶段寄存器写使能信号
input regConditionMoveInMEM	MEM 阶段寄存器条件写信号
input regWriteEnabledInWB	WB 阶段寄存器写使能信号
input [4:0] regFinalDstInEX	EX 阶段寄存器写地址
input [4:0] regFinalDstInMEM	MEM 阶段寄存器写地址
input [4:0] regFinalDstInWB	WB 阶段寄存器写地址
input toMulOrDivInID	ID 阶段指令类型信号，为乘除法相关指令为 1
input runMulOrDivInEX	EX 阶段乘除法器正在计算信号
input errorOccored	异常/中断导致跳转时的标记信号
output [3:0] forwardInID	ID 阶段前推选择信号
output [3:0] forwardInEX	EX 阶段前推选择信号
output [1:0] forwardInMEM	MEM 阶段前推选择信号

output stallAndFlush	流水线暂停信号

19. EX 级异常处理器 EXErrorJudgeUnit

模块定义

```
module EXErrorJudgeUnit(
    input [5:0] operationCodeInEX
    input aluOverFlowInEX
    input cuOverFlowInEX
    input [31:0] ansFromAluInEX
    output [4:0] errorCodeInEX
);
```

名称	功能
input [5:0] operationCodeInEX	当前指令的 Op 部分
input aluOverFlowInEX	来自 Alu 的溢出信号
input cuOverFlowInEX	译码得到的算术指令溢出信号
input [31:0] ansFromAluInEX	EX 级 Alu 的计算结果
output [4:0] errorCodeInEX	EX 级的错误码，没有错误则为 0

20. IF 到 ID 阶段流水线寄存器 FlowReg_IF_ID

模块定义

```
module FlowReg_IF_ID(
    input clk
    input resetOf_IF_ID
    input stallOf_IF_ID
    input [31:0] currentCommandInIF
    input [31:0] currentCommandAddrInIF
    input [31:0] nextCommandAddrInIF
    input [4:0] errorCodeInIF
    input isDelayBranchInIF
```

```
    input errorAddrTagInIF
    output [31:0] currentCommandInID
    output [31:0] currentCommandAddrInID
    output [31:0] nextCommandAddrInID
    output [4:0] errorSrcCodeInID
    output isDelayBranchInID
    output errorAddrTagInID
);
```

21. ID 到 EX 阶段流水线寄存器 FlowReg_ID_EX

模块定义

```
module FlowReg_ID_EX(
    input clk
    input resetOf_ID_EX
    input stallOf_ID_EX
    input [2:0] tNewInID
    input [31:0] currentCommandInID
    input [31:0] currentCommandAddrInID
    input [3:0] aluOperationInID
    input [1:0] aluInputSelectInID
    input cuOverFlowInID
    input [31:0] immAfterExtendInID
    input cuDmBranchInID
    input [3:0] cmpOperationInID
    input dmConditionMoveInID
    input memWriteEnabledInID
    input [2:0] loadWriteMoodInID
    input [1:0] pcControlInID
    input [3:0] dataToRegSelectInID
```

```
input [4:0] regFinalDstInID
input regWriteEnabledInID
input [31:0] dataWriteToRegInID
input [31:0] regData1ForwardedInID
input [31:0] regData2ForwardedInID
input [3:0] mulDivOperationInID
input exConditionMoveInID
input cuMdBranchInID
input writeEnabledOfCP0InID
input [4:0] errorCodeInID
input isDelayBranchInID
input toFlushCP0InID
input errorAddrTagInID
output [2:0] tNewInEX
output [31:0] currentCommandInEX
output [31:0] currentCommandAddrInEX
output [3:0] aluOperationInEX
output [1:0] aluInputSelectInEX
output cuOverFlowInEX
output [31:0] immAfterExtendInEX
output cuDmBranchInEX
output [3:0] cmpOperationInEX
output dmConditionMoveInEX
output memWriteEnabledInEX
output [2:0] loadWriteMoodInEX
output [1:0] pcControlInEX
output [3:0] dataToRegSelectInEX
output [4:0] regFinalDstInEX
```

```
    output regWriteEnabledInEX
    output [31:0] dataWriteToRegFromIDInEX
    output [31:0] regData1InEX
    output [31:0] regData2InEX
    output [3:0] mulDivOperationInEX
    output exConditionMoveInEX
    output cuMdBranchInEX
    output writeEnabledOfCP0InEX
    output [4:0] errorSrcCodeInEX
    output isDelayBranchInEX
    output toFlushCP0InEX
    output errorAddrTagInEX
);
```

22. EX 到 MEM 阶段流水线寄存器 FlowReg_EX_MEM

模块定义

```
module FlowReg_EX_MEM(
    input clk
    input resetOf_EX_MEM
    input stallOf_EX_MEM
    input [2:0] tNewInEX
    input [31:0] currentCommandInEX
    input [31:0] currentCommandAddrInEX
    input [31:0] aluOutputInEX
    input cuDmBranchInEX
    input [3:0] cmpOperationInEX
    input dmConditionMoveInEX
    input memWriteEnabledInEX
    input [2:0] loadWriteMoodInEX
```

```
    input [1:0] pcControlInEX
    input dataToRegSelectInEX
    input regWriteEnabledInEX
    input [4:0] regFinalDstInEX
    input [31:0] dataWriteToRegInEX
    input [31:0] regData1ForwardedInEX
    input [31:0] regData2ForwardedInEX
    output [2:0] tNewInMEM
    output [31:0] currentCommandInMEM
    output [31:0] currentCommandAddrInMEM
    output [31:0] aluOutputInMEM
    output cuDmBranchInMEM
    output [3:0] cmpOperationInMEM
    output dmConditionMoveInMEM
    output memWriteEnabledInMEM
    output [2:0] loadWriteMoodInMEM
    output [1:0] pcControlInMEM
    output dataToRegSelectInMEM
    output regWriteEnabledInMEM
    output [4:0] regFinalDstInMEM
    output [31:0] dataWriteToRegFromEXInMEM
    output [31:0] regData1InMEM
    output [31:0] regData2InMEM
);
```

23. MEM 到 WB 阶段流水线寄存器 FlowReg_MEM_WB

模块定义

```
module FlowReg_MEM_WB(
    input clk
```

```
    input resetOf_MEM_WB
    input stallOf_MEM_WB
    input [31:0] currentCommandAddrInMEM
    input regWriteEnabledInMEM
    input [31:0] dataWriteToRegInMEM
    input [4:0] regFinalDstInMEM
    input [2:0] loadWriteMoodInMEM
    input [1:0] addrOfDataInMEM
    output [31:0] currentCommandAddrInWB
    output regWriteEnabledInWB
    output [31:0] dataWriteToRegFromMEMInWB
    output [4:0] regFinalDstInWB
    output [2:0] loadWriteMoodInWB
    output [1:0] addrOfDataInWB
);
```

24. 中央处理器 CentralProcessingUnit

模块定义

```
module CentralProcessingUnit(
    input clk
    input reset
    input debugMood
    input [5:0] interruptRequest
    input [31:0] memDataToRegInMEM
    input [31:0] currentCommandInIF
    output [31:0] currentCommandAddrInIF
    output [2:0] loadWriteMoodInMEM
    output [31:0] memReadOrWriteAddr
    output memWriteEnabledInMEM
```

```
    output [31:0] dataWriteToMemInMEM
    output [31:0] currentCommandAddrInMEM
    output [31:0] commandAddrToCP0InEX
);
```

名称	功能
input clk	时钟信号
input reset	重置信号
input debugMood	全局 debug 信号，为 1 时行为和 Mars 一致
input [5:0] interruptRequest	六位全局中断请求信号
input [31:0] memDataToRegInMEM	
input [31:0] currentCommandInIF	从 IM 取出的 IF 级的指令
output [31:0] currentCommandAddrInIF	PC 给出的下一条指令的地址
output [2:0] loadWriteMoodInMEM	输出至 DataMemory 的读写模式选择信号
output [31:0] memReadOrWriteAddr	对外存取数据的地址
output memWriteEnabledInMEM	向外部存数据的写使能信号
output [31:0] dataWriteToMemInMEM	从 CPU 寄存器输出的数据
output [31:0] currentCommandAddrInMEM	当前 MEM 级指令地址，用来按要求输出
output [31:0] commandAddrToCP0InEX	对外输出的宏观指令地址

25. 指令存储器 InstructionMemory

模块定义

```
module InstructionMemory(
    input [31:0] currentCommandAddr
    output [31:0] currentCommand
);
```

名称	功能
input [31:0] currentCommandAddr	当前指令的地址
output [31:0] currentCommand	当前地址的指令

26. 数据存储器 DataMemory

模块定义

```
module DataMemory(
    input clk
    input reset
    input [3:0] loadWriteMood
    input [31:0] addrOfDataInMem
    input memWriteEnabled
    input [31:0] dataWriteToMem
    input [31:0] currentCommandAddr
    output [31:0] dataGiveToReg
);
```

名称	功能
input clk	时钟信号
input reset	重置信号
input [3:0] loadWriteMood	读写模式选择信号
input [31:0] addrOfDataInMem	数据写入地址
input memWriteEnabled	数据写使能信号
input [31:0] dataWriteToMem	需要写入 DM 的数据
input [31:0] currentCommandAddr	当前指令地址，用来按要求输出结果
output [31:0] dataGiveToReg	当前地址下的数据

27. 标准计数器 StandardTimer

模块定义

```
module StandardTimer(
    input clk
    input reset
    input debugMood
```

```
    input [31:0] timerAddr
    input timerWriteEnabled
    input [31:0] dataToTimer
    input [31:0] currentCommandAddr
    output [31:0] dataFromTimer
    output interruptRequest
);
```

名称	功能
input clk	时钟信号
input reset	重置信号
input debugMood	全局 debug 信号，为 1 时行为切换到和 Mars 一致
input [31:0] timerAddr	计时器存取地址
input timerWriteEnabled	计时器写使能信号
input [31:0] dataToTimer	存入计时器的数据
input [31:0] currentCommandAddr	当前指令地址
output [31:0] dataFromTimer	从计时器取出的数据
output interruptRequest	中断请求信号

28. 系统桥 SystemBridge

模块定义

```
module SystemBridge(
    input memWriteEnabled
    input [31:0] memReadOrWriteAddr
    input [31:0] dataToRegFromMem
    input [31:0] dataToRegFromTimer1
    input [31:0] dataToRegFromTimer2
    output [31:0] addrOfDataInMem
    output [31:0] addrOfDataInTimer1
```

```
    output [31:0] addrOfDataInTimer2
    output writeEnabledOfMem
    output writeEnabledOfTimer1
    output writeEnabledOfTimer2
    output [31:0] memDataToReg
);
```

名称	功能
input memWriteEnabled	来自 CPU 的写使能信号
input [31:0] memReadOrWriteAddr	来自 CPU 的原始写入地址
input [31:0] dataToRegFromMem	来自 CPU 寄存器的存入的数据
input [31:0] dataToRegFromTimer1	从 Timer1 输出的数据
input [31:0] dataToRegFromTimer2	从 Timer2 输出的数据
output [31:0] addrOfDataInMem	从 DataMemory 输出的数据
output [31:0] addrOfDataInTimer1	输出至 Timer1 的经过处理的地址
output [31:0] addrOfDataInTimer2	输出至 Timer2 的经过处理的地址
output writeEnabledOfMem	输出至 DataMemory 的经过处理的地址
output writeEnabledOfTimer1	输出至 Timer1 的写使能信号
output writeEnabledOfTimer2	输出至 Timer2 的写使能信号
output [31:0] memDataToReg	输出至 DataMemory 的写使能信号

（三）控制模块设计

1.控制单元真值表

为便于表格呈现，将表格分为四部分，将通过 20-16 位确定类型的指令的 20-16 位填在 4-0 位的位置，并以前导/进行区分。

第一部分：

name	31	30	29	28	27	26	5	4	3	2	1	0	Extend	Extend	AluSel	AluSel	AluOpr	Alu0	Alu0	Alu0	Overfl	RegBra	MDBran

													Sel	Sel				pr	pr	pr	ow	nch	ch
lw	1	0	0	0	1	1	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
sw	1	0	1	0	1	1	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
lb	1	0	0	0	0	0	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
lbu	1	0	0	1	0	0	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
sb	1	0	1	0	0	0	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
lh	1	0	0	0	0	1	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
lhu	1	0	0	1	0	1	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
sh	1	0	1	0	0	1	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
add	0	0	0	0	0	0	1	0	0	0	0	0	X	X	0	0	0	1	1	0	1	0	0
addu	0	0	0	0	0	0	1	0	0	0	0	1	X	X	0	0	0	1	1	0	0	0	0
sub	0	0	0	0	0	0	1	0	0	0	1	0	X	X	0	0	0	1	1	1	1	0	0
subu	0	0	0	0	0	0	1	0	0	0	1	1	X	X	0	0	0	1	1	1	0	0	0
and	0	0	0	0	0	0	1	0	0	1	0	0	X	X	0	0	0	0	0	0	0	0	0
or	0	0	0	0	0	0	1	0	0	1	0	1	X	X	0	0	0	0	0	1	0	0	0
slt	0	0	0	0	0	0	1	0	1	0	1	0	X	X	0	0	0	1	0	0	0	0	0
nor	0	0	0	0	0	0	1	0	0	1	1	1	X	X	0	0	0	0	1	0	0	0	0
xor	0	0	0	0	0	0	1	0	0	1	1	0	X	X	0	0	0	0	1	1	0	0	0
sll	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
srl	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0
sra	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	1	0	0	0	0
sllv	0	0	0	0	0	0	0	0	0	1	0	0	X	X	0	0	1	0	0	0	0	0	0
srlv	0	0	0	0	0	0	0	0	0	1	1	0	X	X	0	0	1	0	0	1	0	0	0
srav	0	0	0	0	0	0	0	0	0	1	1	1	X	X	0	0	1	0	1	0	0	0	0
sltu	0	0	0	0	0	0	1	0	1	0	1	1	X	X	0	0	0	1	0	1	0	0	0
beq	0	0	0	1	0	0	X	X	X	X	X	X	0	0	X	X	X	X	X	X	0	1	0
bne	0	0	0	1	0	1	X	X	X	X	X	X	0	0	X	X	X	X	X	X	0	1	0

bgtz	0	0	0	1	1	1	X	X	X	X	X	X	0	0	X	X	X	X	X	X	0	1	0
blez	0	0	0	1	1	0	X	X	X	X	X	X	0	0	X	X	X	X	X	X	0	1	0
bltz	0	0	0	0	0	1	\N	\0	\0	\0	\0	\0	0	0	X	X	X	X	X	X	0	1	0
bgez	0	0	0	0	0	1	\N	\0	\0	\0	\0	\1	0	0	X	X	X	X	X	X	0	1	0
addi	0	0	1	0	0	0	X	X	X	X	X	X	0	0	0	1	0	1	1	0	1	0	0
addiu	0	0	1	0	0	1	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
andi	0	0	1	1	0	0	X	X	X	X	X	X	0	1	0	1	0	0	0	0	0	0	0
ori	0	0	1	1	0	1	X	X	X	X	X	X	0	1	0	1	0	0	0	1	0	0	0
xori	0	0	1	1	1	0	X	X	X	X	X	X	0	1	0	1	0	0	1	1	0	0	0
lui	0	0	1	1	1	1	X	X	X	X	X	X	1	1	X	X	X	X	X	X	0	0	0
slti	0	0	1	0	1	0	X	X	X	X	X	X	0	0	0	1	0	1	0	0	0	0	0
sltiu	0	0	1	0	1	1	X	X	X	X	X	X	0	0	0	1	0	1	0	1	0	0	0
j	0	0	0	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0
jr	0	0	0	0	0	0	0	0	1	0	0	0	X	X	X	X	X	X	X	X	0	0	0
jal	0	0	0	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0
jalr	0	0	0	0	0	0	0	0	1	0	0	1	X	X	X	X	X	X	X	X	0	0	0
mfhi	0	0	0	0	0	0	0	1	0	0	0	0	X	X	X	X	X	X	X	X	0	0	0
mflo	0	0	0	0	0	0	0	1	0	0	1	0	X	X	X	X	X	X	X	X	0	0	0
mthi	0	0	0	0	0	0	0	1	0	0	0	1	X	X	X	X	X	X	X	X	0	0	0
mtlo	0	0	0	0	0	0	0	1	0	0	1	1	X	X	X	X	X	X	X	X	0	0	0
mult	0	0	0	0	0	0	0	1	1	0	0	0	X	X	0	0	X	X	X	X	0	0	0
multu	0	0	0	0	0	0	0	1	1	0	0	1	X	X	0	0	X	X	X	X	0	0	0
div	0	0	0	0	0	0	0	1	1	0	1	0	X	X	0	0	X	X	X	X	0	0	0
divu	0	0	0	0	0	0	0	1	1	0	1	1	X	X	0	0	X	X	X	X	0	0	0
mfc0	0	1	0	0	0	0	\E	\0	\0	\0	\0	\0	X	X	X	X	X	X	X	X	0	0	0

mtc0	0	1	0	0	0	0	\E	\0	\0	\1	\0	\0	X	X	X	X	X	X	X	X	0	0	0
eret	0	1	0	0	0	0	0	1	1	0	0	0	X	X	X	X	X	X	X	X	0	0	0

第二部分：

name	DmBranch	CmpIDOpr	CmpIDOpr	CmpIDOpr	CmpIDOpr	MemWrite	DMSel	DMSel	DMSel	RegDst	RegDst	RegWrite	MoveSelID	MoveSelEX	MoveSelMEM	CP0Write	PcControl
lw	0	X	X	X	X	0	0	0	X	0	0	1	1	1	1	0	0
sw	0	X	X	X	X	1	0	0	X	X	X	0	X	X	X	0	0
lb	0	X	X	X	X	0	1	0	1	0	0	1	1	1	1	0	0
lbu	0	X	X	X	X	0	1	0	0	0	0	1	1	1	1	0	0
sb	0	X	X	X	X	1	1	0	X	X	X	0	X	X	X	0	0
lh	0	X	X	X	X	0	0	1	1	0	0	1	1	1	1	0	0
lhu	0	X	X	X	X	0	0	1	0	0	0	1	1	1	1	0	0
sh	0	X	X	X	X	1	0	1	X	X	X	0	X	X	X	0	0
add	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
addu	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
sub	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
subu	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
and	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
or	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
slt	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
nor	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
xor	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
sll	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
srl	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
sra	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
sllv	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0

srlv	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
srav	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
sltu	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
beq	0	0	0	0	0	0	X	X	X	X	X	0	X	X	X	0	0
bne	0	0	0	0	1	0	X	X	X	X	X	0	X	X	X	0	0
bgtz	0	0	0	1	1	0	X	X	X	X	X	0	X	X	X	0	0
blez	0	0	1	0	0	0	X	X	X	X	X	0	X	X	X	0	0
bltz	0	0	1	0	1	0	X	X	X	X	X	0	X	X	X	0	0
bgez	0	0	0	1	0	0	X	X	X	X	X	0	X	X	X	0	0
addi	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
addiu	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
andi	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
ori	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
xori	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
lui	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
slti	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
sltiu	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
j	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	1
jr	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	1
jal	0	X	X	X	X	0	0	0	0	1	X	1	1	1	1	0	1
jalr	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	1
mfhi	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
mflo	0	X	X	X	X	0	0	0	0	0	1	1	1	1	1	0	0
mthi	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	0
mtlo	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	0
mult	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	0
multu	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	0

div	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	0
divu	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	0
mfc0	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	0	0
mtc0	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	1	0
eret	0	X	X	X	X	0	X	X	X	X	X	0	X	X	X	0	0

第三部分：

name	PcControl	SelDataID	SelDataID	SelDataEX	SelDataEX	SelDataEX	SelDataMEM	MorD	MDOpr	MDOpr	MDOpr	MDOpr	Knowlt	ErepBack	TuseRs	TuseRs	TuseRs
lw	0	X	X	X	X	X	1	0	0	0	0	0	1	0	0	1	0
sw	0	X	X	X	X	X	X	0	0	0	0	0	1	0	0	1	0
lb	0	X	X	X	X	X	1	0	0	0	0	0	1	0	0	1	0
lbu	0	X	X	X	X	X	1	0	0	0	0	0	1	0	0	1	0
sb	0	X	X	X	X	X	X	0	0	0	0	0	1	0	0	1	0
lh	0	X	X	X	X	X	1	0	0	0	0	0	1	0	0	1	0
lhu	0	X	X	X	X	X	1	0	0	0	0	0	1	0	0	1	0
sh	0	X	X	X	X	X	X	0	0	0	0	0	1	0	0	1	0
add	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
addu	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
sub	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
subu	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
and	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
or	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
slt	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
nor	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
xor	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
sll	0	X	X	0	0	1	0	0	0	0	0	0	1	0	1	1	1

srl	0	X	X	0	0	1	0	0	0	0	0	0	1	0	1	1	1
sra	0	X	X	0	0	1	0	0	0	0	0	0	1	0	1	1	1
sllv	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
srlv	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
srav	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
sltu	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
beq	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	0	1
bne	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	0	1
bgtz	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	0	1
blez	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	0	1
bltz	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	0	1
bgez	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	0	1
addi	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
addiu	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
andi	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
ori	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
xori	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
lui	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1
slti	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
sltiu	0	X	X	0	0	1	0	0	0	0	0	0	1	0	0	1	0
j	0	X	X	X	X	X	X	0	0	0	0	0	1	0	1	1	1
jr	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	0	1
jal	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1
jalr	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
mfhi	0	X	X	0	1	0	0	1	0	0	0	0	1	0	1	1	1
mflo	0	X	X	0	1	1	0	1	0	0	0	0	1	0	1	1	1
mthi	0	X	X	X	X	X	X	1	0	0	0	1	1	0	0	1	0

mtlo	0	X	X	X	X	X	X	1	0	0	1	0	1	0	0	1	0
mult	0	X	X	X	X	X	X	1	0	1	0	1	1	0	0	1	0
multu	0	X	X	X	X	X	X	1	0	1	1	0	1	0	0	1	0
div	0	X	X	X	X	X	X	1	0	0	1	1	1	0	0	1	0
divu	0	X	X	X	X	X	X	1	0	1	0	0	1	0	0	1	0
mfc0	0	X	X	1	0	0	0	0	0	0	0	0	1	0	1	1	1
mtc0	0	X	X	X	X	X	X	0	0	0	0	0	1	0	1	1	1
eret	0	X	X	X	X	X	X	0	0	0	0	0	1	1	1	1	1

第四部分：

name	TuseRt	TuseRt	TuseRt	Tnew	Tnew	Tnew
lw	0	1	1	1	0	0
sw	1	1	1	0	0	0
lb	0	1	1	1	0	0
lbu	0	1	1	1	0	0
sb	1	1	1	0	0	0
lh	0	1	1	1	0	0
lhu	0	1	1	1	0	0
sh	1	1	1	0	0	0
add	0	1	0	0	1	1
addu	0	1	0	0	1	1
sub	0	1	0	0	1	1
subu	0	1	0	0	1	1
and	0	1	0	0	1	1
or	0	1	0	0	1	1
slt	0	1	0	0	1	1
nor	0	1	0	0	1	1
xor	0	1	0	0	1	1

sll	0	1	0	0	1	1
srl	0	1	0	0	1	1
sra	0	1	0	0	1	1
sllv	0	1	0	0	1	1
srlv	0	1	0	0	1	1
srav	0	1	0	0	1	1
sltu	0	1	0	0	1	1
beq	0	0	1	0	0	0
bne	0	0	1	0	0	0
bgtz	1	1	1	0	0	0
blez	1	1	1	0	0	0
bltz	1	1	1	0	0	0
bgez	1	1	1	0	0	0
addi	1	1	1	0	1	1
addiu	1	1	1	0	1	1
andi	1	1	1	0	1	1
ori	1	1	1	0	1	1
xori	1	1	1	0	1	1
lui	1	1	1	0	1	0
slti	1	1	1	0	1	1
sltiu	1	1	1	0	1	1
j	1	1	1	0	0	0
jr	1	1	1	0	0	0
jal	1	1	1	0	1	0
jalr	1	1	1	0	1	0
mfhi	1	1	1	0	1	1
mflo	1	1	1	0	1	1

mthi	1	1	1	0	0	0
mtlo	1	1	1	0	0	0
mult	0	1	0	0	0	0
multu	0	1	0	0	0	0
div	0	1	0	0	0	0
divu	0	1	0	0	0	0
mfc0	1	1	1	0	1	1
mtc0	0	1	0	0	0	0
eret	1	1	1	0	0	0

2.冲突模块逻辑

ID 阶段前推逻辑

```
assign forwardInID[3:2]={currentCommand2521InID!=0 &&
    (currentCommand2521InID==regFinalDstInMEM) &&
    regWriteEnabledInMEM && regConditionMoveInMEM
      ,currentCommand2521InID!=0 &&
    (currentCommand2521InID==regFinalDstInEX) && regWriteEnabledInEX
    && regConditionMoveInEX};
assign forwardInID[1:0]={currentCommand2016InID!=0 &&
    (currentCommand2016InID==regFinalDstInMEM) &&
    regWriteEnabledInMEM && regConditionMoveInMEM
      ,currentCommand2016InID!=0 &&
    (currentCommand2016InID==regFinalDstInEX) && regWriteEnabledInEX
    && regConditionMoveInEX};
```

EX 阶段前推逻辑

```
assign forwardInEX[3:2]={currentCommand2521InEX!=0 &&
    (currentCommand2521InEX==regFinalDstInWB) && regWriteEnabledInWB
      ,currentCommand2521InEX!=0 &&
    (currentCommand2521InEX==regFinalDstInMEM) &&
```

```
    regWriteEnabledInMEM && regConditionMoveInMEM};
assign forwardInEX[1:0]={currentCommand2016InEX!=0 &&
    (currentCommand2016InEX==regFinalDstInWB) && regWriteEnabledInWB
      ,currentCommand2016InEX!=0 &&
    (currentCommand2016InEX==regFinalDstInMEM) &&
    regWriteEnabledInMEM && regConditionMoveInMEM};
```

MEM 阶段前推逻辑

```
assign forwardInMEM[1]=currentCommand2521InMEM!=0 &&
    (currentCommand2521InMEM==regFinalDstInWB) &&
    regWriteEnabledInWB;
assign forwardInMEM[0]=currentCommand2016InMEM!=0 &&
    (currentCommand2016InMEM==regFinalDstInWB) &&
    regWriteEnabledInWB;
```

阻塞部分逻辑

```
assign stallDueToMulOrDiv=toMulOrDivInID && runMulOrDivInEX;
assign stallDueTo2521AndEX=(tUseOf2521InID<tNewInEX) &&
    regWriteEnabledInEX && currentCommand2521InID!=0 &&
    (currentCommand2521InID==regFinalDstInEX);
assign stallDueTo2521AndMEM=(tUseOf2521InID<tNewInMEM) &&
    regWriteEnabledInMEM   && currentCommand2521InID!=0 &&
    (currentCommand2521InID==regFinalDstInMEM);
assign stallDueTo2016AndEX=(tUseOf2016InID<tNewInEX) &&
    regWriteEnabledInEX && currentCommand2016InID!=0 &&
    (currentCommand2016InID==regFinalDstInEX);
assign stallDueTo2016AndMEM=(tUseOf2016InID<tNewInMEM) &&
    regWriteEnabledInMEM   && currentCommand2016InID!=0 &&
    (currentCommand2016InID==regFinalDstInMEM);
assign stallAndFlush=!errorOccored && (stallDueTo2521AndEX ||
```

```
stallDueTo2521AndMEM || stallDueTo2016AndEX ||
stallDueTo2016AndMEM || stallDueToMulOrDiv);
```

3.零号协处理器设计

状态机部分

```
if (reset) begin
   if (debugMood) statusRegister<=32'b0;
   else statusRegister<=32'hfc01; causeRegister<=0;
   errorProgramCounter<=0; processorId<=32'h125e591;
end
else begin
   if (!debugMood) causeRegister[15:10]<=interruptRequest;
   if (flush) begin statusRegister[1]<=1'b0; end else begin end
   if (jumpTo4180) begin
      if (debugMood) begin
      causeRegister[6:2]<=jumpDueToInterrupt?5'b0:errorCode;
      causeRegister[31]<=isInBranchDelay; end
      else
      causeRegister<={isInBranchDelay,15'b0,interruptRequest,3'b
      0,jumpDueToInterrupt?5'b0:errorCode,2'b0};
      if (isInBranchDelay)
      errorProgramCounter<=((commandAddrToCP0>>2)<<2)-32'h4;
      else errorProgramCounter<=((commandAddrToCP0>>2)<<2);
      statusRegister[1]<=1'b1;
   end
   if (writeEnabledOfCP0) begin
      if (readOrWriteAddrInCP0==12)
      statusRegister<=dataWriteToCP0;
      else if (readOrWriteAddrInCP0==13)
```

```
        causeRegister<=dataWriteToCP0;
        else if (readOrWriteAddrInCP0==14)
        errorProgramCounter<=dataWriteToCP0;
        else if (readOrWriteAddrInCP0==15)
        processorId<=dataWriteToCP0;
        else begin end
    end
    else begin end
end
```

组合逻辑部分

```
assign dataGetFromCP0=writeEnabledOfCP0?dataWriteToCP0:
   (readOrWriteAddrInCP0==12)?statusRegister:
   (readOrWriteAddrInCP0==13)?causeRegister:
   (readOrWriteAddrInCP0==14)?errorProgramCounter:
   (readOrWriteAddrInCP0==15)?processorId:32'b0;
assign commandAddrFromCP0=(writeEnabledOfCP0 &&
(readOrWriteAddrInCP0==14))?dataWriteToCP0:errorProgramCounter;
assign jumpDueToError=(errorCode==4)|(errorCode==5)|
(errorCode==10)|(errorCode==12);
assign jumpDueToInterrupt=|(interruptRequest&
statusRegister[15:10])&~statusRegister[1]&statusRegister[0];
assign jumpTo4180=jumpDueToError|jumpDueToInterrupt;
```

4.桥与 IO 设计

```
assign isSolvingMem=memReadOrWriteAddr<=32'h3000;
assign isSolvingTimer1=(32'h7F00<=memReadOrWriteAddr)
&& (memReadOrWriteAddr<=32'h7F0F);
assign isSolvingTimer2=(32'h7F10<=memReadOrWriteAddr)
&& (memReadOrWriteAddr<=32'h7F1F);
```

```
assign addrOfDataInMem=memReadOrWriteAddr;
assign addrOfDataInTimer1=memReadOrWriteAddr;
assign addrOfDataInTimer2=memReadOrWriteAddr;
assign writeEnabledOfMem=memWriteEnabled&isSolvingMem;
assign writeEnabledOfTimer1=memWriteEnabled&isSolvingTimer1;
assign writeEnabledOfTimer2=memWriteEnabled&isSolvingTimer2;
assign memDataToReg=isSolvingMem?dataToRegFromMem:
   isSolvingTimer1?dataToRegFromTimer1:
   isSolvingTimer2?dataToRegFromTimer2:32'b1;
```

5.辅助编码控制单元的程序

我们需要将上述三个真值表转换为 ControlUnit 中的代码，手工编码是一件痛苦的事情，且容易出错，因此考虑使用 C 语言根据真值表直接生成这部分的代码，此程序可根据真值表直接完成 ControlUnit 中的宏定义部分和三类控制信号格式的三目运算符连接，已提交到自动测试或辅助工具提交窗口。

6.辅助抓取模块定义的程序

在撰写文档时，需要获得所有.v 文件中的模块定义，出于便捷考虑，编写 C 语言程序来实现这一要求，此程序可搜索目录下所有的.v 文件，汇总其中所有的模块定义到一个文件，并自动忽略 tb 模块中的内容，已提交到自动测试或辅助工具提交窗口。

二、测试方案

（一）典型测试样例与数据构造方法

正确性及异常测试部分：手动构造数据，说明如下：

testpoint1：主要测试若干涉及跳转的指令，为 53 条指令中标准的跳转相关指令，测试的指令：j/jal/jalr/jr，存在其它指令但保证在标准的 53 条指令范围内。

testpoint2：主要测试若干涉及分支的指令，为 53 条指令中标准的分支相关指令，测试的指令：beq/bne/bgez/bgtz/blez/bltz，存在其它指令但保证在标准的 53 条指令范围内。

testpoint3：主要测试若干寄存器计算类型的指令，为 53 条指令中标准的寄存器计算指

令，测试的指令：and/add/addu/sub/subu/slt/sltu/sllv/srlv/srav/or/nor/xor，存在其它指令但保证在标准的 53 条指令范围内。

testpoint4：主要测试若干立即数计算类型的指令，为 53 条指令中标准的立即数计算指令，测试的指令：addi/addiu/andi/ori/sll/srl/sra/lui/slti/sltiu/xori，存在其它指令但保证在标准的 53 条指令范围内。

testpoint5：主要测试若干乘除法相关的指令，为 53 条指令中标准的乘除法相关指令，测试的指令：mflo/mfhi/mtlo/mthi/mult/multu/div/divu，存在其它指令但保证在标准的 53 条指令范围内。

testpoint6：主要测试若干涉及读写内存的指令，为 53 条指令中标准的内存读写指令，测试的指令：lb/lbu/lh/lhu/lw/sb/sh/sw，存在其它指令但保证在标准的 53 条指令范围内。

testpoint7：主要测试若干非法指令，为在英文指令集但不在 53 条指令中的指令，测试的指令：movz/movz/clo/clz/madd/msub/maddu/msubu，存在其它指令但保证在标准的 53 条指令范围内。

testpoint8：主要测试若干涉及存取数据的异常情况，异常的判定按 OJ 要求，测试的指令：lb/lbu/lh/lhu/lw/sb/sh/sw，存在其它指令但保证在标准的 53 条指令范围内。

testpoint9：主要测试若干涉及跳转的异常情况，异常的判定按 OJ 要求，测试的指令：jr/jalr，存在其它指令但保证在标准的 53 条指令范围内。

testpoint10：主要测试若干涉及延迟槽存取数据的异常情况，异常的判定按 OJ 要求，测试的指令：lb/lbu/lh/lhu/lw/sb/sh/sw，存在其它指令但保证在标准的 53 条指令范围内。

testpoint11：主要测试若干涉及延迟槽溢出和非法指令的异常情况，异常的判定按 OJ 要求，测试的指令：add/sub/addi/movz/movn/clo/clz，存在其它指令但保证在标准的 53 条指令范围内。

testpoint12：主要测试若干涉及跳转错误且延迟槽存取数据的异常情况，异常的判定按 OJ 要求，测试的指令：lb/lbu/lh/lhu/lw/sb/sh/sw，存在其它指令但保证在标准的 53 条指令范围内。

testpoint13：主要测试若干涉及跳转错误且延迟槽溢出和非法指令的异常情况，异常的判定按 OJ 要求，测试的指令：add/sub/addi/movz/movn/clo/clz，存在其它指令但保证在标准的 53 条指令范围内。

testpoint14：本测试点为一个有意义的程序，为跑所有水仙花数，包含的指令：add/addi/beq/j/jal/jr/slt/slti/sw。

testpoint15：本测试点为一个有意义的程序，为对 100 个数进行快速排序，包含的指令：add/addi/addiu/beq/bgez/bgtz/blez/bltz/bne/j/jal/jr/lw/sll/srl/sub/sw。

流水线冲突处理测试部分：自动生成数据，说明如下：

根据真值表将指令进行细分，保证每一类中拥有高度相似的功能和两个 Tuse 一个 Tnew，共 16 类依次是 add/addu/and/nor/or/sllv/srlv/srav/slt/sltu/sub/subu/xor 这些 r 型指令，addi/addiu/andi/ori/slti/sltiu/xori 这些标准的 i 型指令，sll/srl/sra 这三个 Tuse 不一样的 i 指令，lb/lbu/lh/lhu/lw 这些读内存指令，sb/sh/sw 这些写内存指令，div/divu/mult/multu/mthi/mtlo/mtc0 这些涉及写 HI 和 LO 或 CP0 中寄存器的指令，mfhi/mflo/mfc0 这些涉及读 HI 和 LO 或 CP0 中寄存器的指令，beq/bne 这些需要两个操作数的分支指令，尽量保证跳转成立，beq/bne 这些需要两个操作数的分支指令，尽量保证跳转不成立，bgtz/bgez/bltz/blez 这些需要一个操作数的分支指令，尽量保证跳转成立，bgtz/bgez/bltz/blez 这些需要一个操作数的分支指令，尽量保证跳转不成立，lui 这个没有 Tuse 的 i 指令，此外 jr/jal/jalr 各不相同，因此分别单分一类，最后是 nop。同时考虑到 if 阶段没有操作数产生也不需要操作数，只需要按类枚举连续四条指令，前三条指令取自三个会产生操作数的指令类，第四条指令取自需要操作数的指令类。考虑到对一组连续四条指令，寄存器选哪个是无所谓的，因此只要枚举三次，用一个寄存器，用两个寄存器，涉及 0 号寄存器，进行了大分类后只要装填指令并打印即可。实现上先分配使用的寄存器，循环枚举所有寄存器来保证覆盖了所有的寄存器，然后分配类型下具体的指令，保证各类指令中每一个都被使用过。对不合法的存取地址，使用内置的 handle 进行处理，jalr 部分生成难度较大，故单独构造测试。具体见自动测试或辅助工具提交窗口中提交的内容。

中断处理部分：此部分通过对拍进行验证，说明如下：

由于难以通过单周期 CPU 的 Mars 进行验证，故通过和同样将 CP0 放在 EX 的林星涵同学的 CPU 对拍来验证，包含六个测试点，分别测试正常中断，异常中断，正常跳转的延迟槽正常中断，正常跳转的延迟槽异常中断，异常跳转的延迟槽正常中断，异常跳转的延迟槽异常中断，mtc0 上中断，内核态 eret 前一条中断，内核态 eret 上中断，内核态 eret 后一条中断，div-beq-mfc0 导致的延迟槽暂停时给出中断等，并手动构造带有错误立即数

的 beq/bne/bgtz/bgez/bltz/blez/j/jal 以测试对这些指令导致的异常处理的功能正确性。对于不需要修改跳转立即数的测试点，可使用程序辅助生成专用的 mips_tb.v，此程序见自动测试或辅助工具提交窗口中提交的内容，对于需要修改跳转立即数的测试点，可使用辅助程序对导出的二进制码进行修改，此程序见自动测试或辅助工具提交窗口中提交的内容，此外，对拍的辅助工具见自动测试或辅助工具提交窗口中提交的内容。

（二）自动测试工具

见自动测试或辅助工具提交窗口中提交的内容。

三、思考题

（一）

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？)

硬件接口是指两个硬件设备之间包括物理上的连接方式和逻辑上的数据传送协议在内的连接方式；软件接口是指软件系统不同组成部分衔接的约定，即提供应用程序与开发人员以访问一组例程而又无需访问源码或理解其内部工作机制的细节的能力。

（二）

在我们设计的流水线中，DM 处于 CPU 内部，请考虑现代计算机中它的位置应该在何处。

DM 应位于 CPU 外部并由 CPU 通过总线访问，此外综合考虑成本和性能问题，往往会在 CPU 和 DM 之间设置 Cache 以加快内存读写性能。

（三）

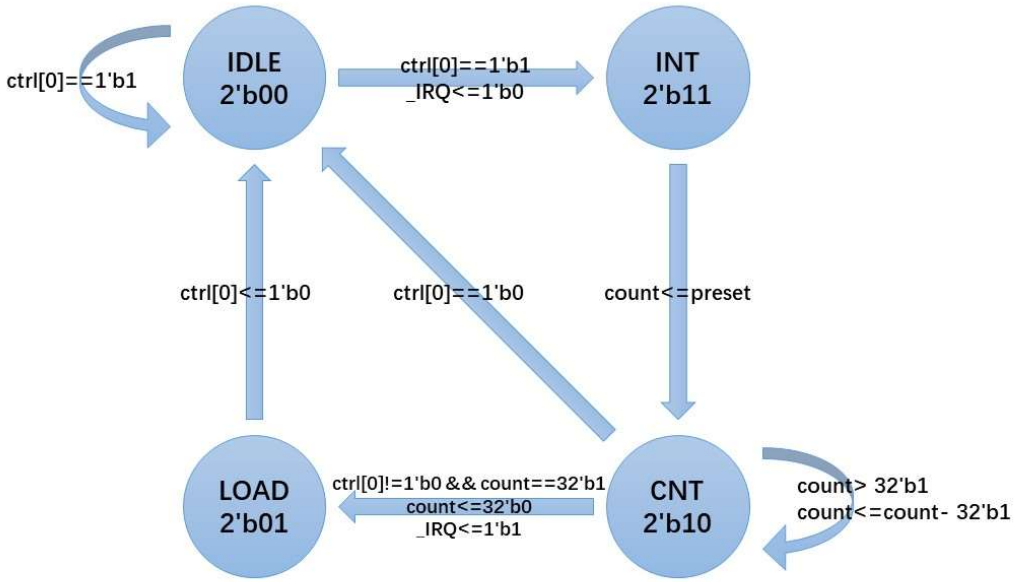
BE 部件对所有的外设都是必要的吗？

BE 部件仅对需要直接与 CPU 寄存器进行数据交换的外设是必要的，对不需要与 CPU 进行数据交换或不需要直接与 CPU 寄存器进行直接数据交换的外设，BE 部件不是必要的。

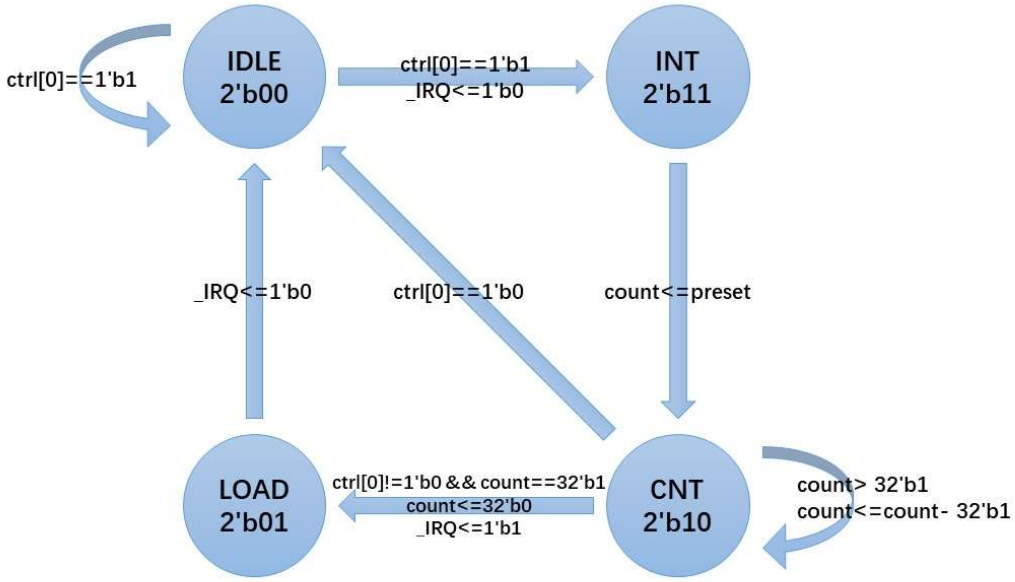
（四）

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相应的状态转移图。

控制信号的格式：ctrl 仅低四位有效，ctrl[3]用以控制中断请求，若为 1 则允许中断；ctrl[2:1]用以确定计数器模式，若为 0 则对应下述模式 0，否则对应下述模式 1；ctrl[0]用以确定是否计数，若为 1 则允许计数；在模式 0 下，计数器倒数至 0 时禁止计数，若允许中断请求，则持续



模式0 |ctrl[2:1]==1'b0



模式1 |ctrl[2:1]!=1'b0

产生中断请求信号，直到允许计数器计数时，将当前待计数的值 preset 加载至计数值 count 并开始计数，同时关闭中断请求；在模式 1 下，计数器倒数至 1 时将当前待计数的值 preset 加载至计数值 count 并开始计数，若允许中断请求，则产生一个周期的中断请求。

（五）

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：定时器在主程序中被初始化为模式 0；定时器倒计数至 0 产生中断；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。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。)

```
.test
   #a demo for timer 0
   addi  $t0,$0,2333
   sw    $t0,0x7F04($0)
   addi  $t0,$0,9
   sw    $t0,0x7F00($0)
   loop: j  loop
   nop
.ktext 0x4180
   sw    $t0,0x7F00($0)
   eret
```

（六）

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？

键盘鼠标等低速设备通过中断请求的方式进行 IO 操作，即当按下一个按键时键盘发出一个中断信号，中断信号经中断控制器传到 CPU，CPU 根据不同的中断号执行不同的中断响应程序并进行相应的 IO 操作，将按下的按键编码读入寄存器，最后放入内存中，鼠标操作同理。