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、movz、movn、bgezal、bltzal、clo、clz、mult、multu、div、divu、madd、maddu、msub、msubu、mflo、mfhi、mtlo、mthi}共 60 条，其中 add、addi 和 sub 不支持溢出中断，其行为与 addu、addiu 和 subu 完全一致。CPU 主要包含了 AluSelect、ArithmeticLogicalUnit、BitExtender、CompareUnit 、 ControlUnit 、 DataMemory 、 DoubleForwardSelect 、 EightDataToRegSelect 、 ExtenderForLoad、FlowReg_EX_MEM、FlowReg_ID_EX、FlowReg_IF_ID、FlowReg_MEM_WB、FourDataToRegSelect 、 InstructionMemory 、 MulDivSolver 、 ProgramCounter 、 RegDstSelect 、 RegisterFile、RiskSolveUnit、SingleForwardSelect 共计 21 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 [3:0] loadWriteMoodInID
    output [1:0] regDstSelectInID
    output regWriteEnabledInID
    output regConditionMoveInID
    output exConditionMoveInID
    output dmConditionMoveInID
    output [1:0] pcControlInID
    output [6:0] dataToRegSelectInID
    output mulOrDivInID
    output [3:0] mulDivOperationInID
    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 [3:0] 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 [1:0] pcControlInID	PC 控制信号，视 cuRegBranchInID 和 cuDmBranchInID 在 ID 或 MEM 使用
output [6:0] 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 [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. 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

7. 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

8. 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

9. 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

10. 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

11. 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

12. 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}

13. 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
    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
output [31:0] currentCommandAddr	当前指令地址
output [31:0] nextCommandAddr	下一条指令地址

14. 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

15. 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

16. 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
    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
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

17. 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

18. 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
    output [31:0] currentCommandInID
    output [31:0] currentCommandAddrInID
```

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

19. 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 [3:0] loadWriteMoodInID
    input [1:0] pcControlInID
    input [4:0] 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
    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 [3:0] loadWriteMoodInEX
    output [1:0] pcControlInEX
    output [4:0] dataToRegSelectInEX
    output [4:0] regFinalDstInEX
    output regWriteEnabledInEX
    output [31:0] dataWriteToRegFormIDInEX
    output [31:0] regData1InEX
    output [31:0] regData2InEX
    output [3:0] mulDivOperationInEX
    output exConditionMoveInEX
    output cuMdBranchInEX
);
```

20. 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 [3:0] loadWriteMoodInEX
input [1:0] pcControlInEX
input [1:0] 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 [3:0] loadWriteMoodInMEM
```

```
    output [1:0] pcControlInMEM
    output [1:0] dataToRegSelectInMEM
    output regWriteEnabledInMEM
    output [4:0] regFinalDstInMEM
    output [31:0] dataWriteToRegFormEXInMEM
    output [31:0] regData1InMEM
    output [31:0] regData2InMEM
);
```

21. 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 [3:0] loadWriteMoodInMEM
    input [1:0] addrOfDataInMEM
    output [31:0] currentCommandAddrInWB
    output regWriteEnabledInWB
    output [31:0] dataWriteToRegFormMEMInWB
    output [4:0] regFinalDstInWB
    output [3:0] loadWriteMoodInWB
    output [1:0] addrOfDataInWB
);
```

（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	ExtendSel	ExtendSel	AluSel	AluSel	AluOpr	Alu0pr	Alu0pr	Alu0pr	Overflow	RegBranch	MDBranch
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
movz	0	0	0	0	0	0	0	0	1	0	1	0	X	X	X	X	X	X	X	X	0	0	0
movn	0	0	0	0	0	0	0	0	1	0	1	1	X	X	X	X	X	X	X	X	0	0	0
bltzal	0	0	0	0	0	1	\N	\1	\0	\0	\0	\0	X	X	X	X	X	X	X	X	0	1	0
bgezal	0	0	0	0	0	1	\N	\1	\0	\0	\0	\1	X	X	X	X	X	X	X	X	0	1	0
madd	0	1	1	1	0	0	0	0	0	0	0	0	X	X	0	0	X	X	X	X	0	0	0
maddu	0	1	1	1	0	0	0	0	0	0	0	1	X	X	0	0	X	X	X	X	0	0	0
msub	0	1	1	1	0	0	0	0	0	1	0	0	X	X	0	0	X	X	X	X	0	0	0
msubu	0	1	1	1	0	0	0	0	0	1	0	1	X	X	0	0	X	X	X	X	0	0	0
clo	0	1	1	1	0	0	1	0	0	0	0	1	X	X	0	0	1	0	1	1	0	0	0
clz	0	1	1	1	0	0	1	0	0	0	0	0	X	X	0	0	1	1	0	0	0	0	0
bltzalr	0	0	0	0	0	0	1	1	1	0	0	0	X	X	X	X	X	X	X	X	0	1	0
bgezalr	0	0	0	0	0	0	1	1	1	0	0	1	X	X	X	X	X	X	X	X	0	1	0
bloltzalr	0	0	0	0	0	0	1	0	1	0	0	0	X	X	X	X	X	X	X	X	0	0	1
blogezalr	0	0	0	0	0	0	1	0	1	0	0	1	X	X	X	X	X	X	X	X	0	0	1
bmltzalr	1	1	1	1	1	0	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0
bmgezalr	1	1	1	1	1	1	X	X	X	X	X	X	0	0	0	1	0	1	1	0	0	0	0

第二部分：

name	DmBranch	CmpIDOpr	CmpIDOpr	CmpIDOpr	CmpIDOpr	MemWrite	DMSel	DMSel	DMSel	DMSel	RegDst	RegDst	RegWrite	MoveSelID	MoveSelEX	MoveSelMEM	PcControl

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

blez	0	0	1	0	0	0	X	X	X	X	X	X	0	X	X	X	0
bltz	0	0	1	0	1	0	X	X	X	X	X	X	0	X	X	X	0
bgez	0	0	0	1	0	0	X	X	X	X	X	X	0	X	X	X	0
addi	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
addiu	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
andi	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
ori	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
xori	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
lui	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
slti	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
sltiu	0	X	X	X	X	0	0	0	0	0	0	0	1	1	1	1	0
j	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	1
jr	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	1
jal	0	X	X	X	X	0	0	0	0	0	1	X	1	1	1	1	1
jalr	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	1	1
mfhi	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	1	0
mflo	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	1	0
mthi	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
mtlo	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
mult	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
multu	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
div	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
divu	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
movz	0	1	1	0	0	0	0	0	0	0	0	1	1	0	1	1	0
movn	0	1	1	0	1	0	0	0	0	0	0	1	1	0	1	1	0
bltzal	0	0	1	0	1	0	0	0	0	0	1	X	1	0	1	1	0
bgezal	0	0	0	1	0	0	0	0	0	0	1	X	1	0	1	1	0

madd	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
maddu	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
msub	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
msubu	0	X	X	X	X	0	X	X	X	X	X	X	0	X	X	X	0
clo	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	1	0
clz	0	X	X	X	X	0	0	0	0	0	0	1	1	1	1	1	0
bltzalr	0	0	1	0	1	0	0	0	0	0	0	1	1	0	1	1	1
bgezalr	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	1	1
bloltzalr	0	0	1	0	1	0	0	0	0	0	0	1	1	1	0	1	1
blogezalr	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	1	1
bmltzalr	1	0	1	0	1	0	0	0	0	0	1	X	1	1	1	0	1
bmgezalr	1	0	0	1	0	0	0	0	0	0	1	X	1	1	1	0	1

第三部分：

name	PcControl	SelDataID	SelDataID	SelDataEX	SelDataEX	SelDataEX	SelDataMEM	SelDataMEM	MorD	MDOpr	MDOpr	MDOpr	MDOpr	TuseRs	TuseRs	TuseRs
lw	0	X	X	X	X	X	0	1	0	0	0	0	0	0	1	0
sw	0	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0
lb	0	X	X	X	X	X	0	1	0	0	0	0	0	0	1	0
lbu	0	X	X	X	X	X	0	1	0	0	0	0	0	0	1	0
sb	0	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0
lh	0	X	X	X	X	X	0	1	0	0	0	0	0	0	1	0
lhu	0	X	X	X	X	X	0	1	0	0	0	0	0	0	1	0
sh	0	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0
add	0	X	X	0	0	1	0	0	0	0	0	0	0	0	1	0
addu	0	X	X	0	0	1	0	0	0	0	0	0	0	0	1	0
sub	0	X	X	0	0	1	0	0	0	0	0	0	0	0	1	0

subu	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
and	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
or	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
slt	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
nor	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
xor	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
sll	0	X	X	0	0	1	0	0	0	0	0	0	1	1	1
srl	0	X	X	0	0	1	0	0	0	0	0	0	1	1	1
sra	0	X	X	0	0	1	0	0	0	0	0	0	1	1	1
sllv	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
srlv	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
srav	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
sltu	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
beq	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1
bne	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1
bgtz	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1
blez	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1
bltz	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1
bgez	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1
addi	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
addiu	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
andi	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
ori	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
xori	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
lui	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1
slti	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0
sltiu	0	X	X	0	0	1	0	0	0	0	0	0	0	1	0

j	0	X	X	X	X	X	X	X	0	0	0	0	0	1	1	1
jr	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1
jal	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
jalr	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
mfhi	0	X	X	0	1	0	0	0	1	0	0	0	0	1	1	1
mflo	0	X	X	0	1	1	0	0	1	0	0	0	0	1	1	1
mthi	0	X	X	X	X	X	X	X	1	0	0	0	1	0	1	0
mtlo	0	X	X	X	X	X	X	X	1	0	0	1	0	0	1	0
mult	0	X	X	X	X	X	X	X	1	0	1	0	1	0	1	0
multu	0	X	X	X	X	X	X	X	1	0	1	1	0	0	1	0
div	0	X	X	X	X	X	X	X	1	0	0	1	1	0	1	0
divu	0	X	X	X	X	X	X	X	1	0	1	0	0	0	1	0
movz	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1
movn	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1
bltzal	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
bgezal	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
madd	0	X	X	X	X	X	X	X	1	0	1	1	1	0	1	0
maddu	0	X	X	X	X	X	X	X	1	1	0	0	0	0	1	0
msub	0	X	X	X	X	X	X	X	1	1	0	0	1	0	1	0
msubu	0	X	X	X	X	X	X	X	1	1	0	1	0	0	1	0
clo	0	X	X	0	0	1	0	0	0	0	0	0	0	0	1	0
clz	0	X	X	0	0	1	0	0	0	0	0	0	0	0	1	0
bltzalr	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
bgezalr	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
bloltzalr	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
blogezalr	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
bmltzalr	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

bmgezalr	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

第四部分：

name	TuseRt	TuseRt	TuseRt	Tnew	Tnew	Tnew
lw	0	1	1	1	0	0
sw	0	1	1	0	0	0
lb	0	1	1	1	0	0
lbu	0	1	1	1	0	0
sb	0	1	1	0	0	0
lh	0	1	1	1	0	0
lhu	0	1	1	1	0	0
sh	0	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
movz	0	0	1	0	1	0

movn	0	0	1	0	1	0
bltzal	1	1	1	0	1	0
bgezal	1	1	1	0	1	0
madd	0	1	0	0	0	0
maddu	0	1	0	0	0	0
msub	0	1	0	0	0	0
msubu	0	1	0	0	0	0
clo	1	1	1	0	1	1
clz	1	1	1	0	1	1
bltzalr	0	0	1	0	1	0
bgezalr	0	0	1	0	1	0
bloltzalr	0	0	0	0	1	1
blogezalr	0	0	0	0	1	1
bmltzalr	0	1	1	1	0	0
bmgezalr	0	1	1	1	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=stallDueTo2521AndEX ||
    stallDueTo2521AndMEM || stallDueTo2016AndEX ||
    stallDueTo2016AndMEM || stallDueToMulOrDiv;
```

3.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。

4.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

见思考题第五题和自动测试或辅助工具提交窗口中提交的内容。

（二）数据构造方法

见思考题第五题和自动测试或辅助工具提交窗口中提交的内容。

（三）自动测试工具

见思考题第五题和自动测试或辅助工具提交窗口中提交的内容。

三、思考题

（一）

为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

乘除法的计算时间较长，如果整合进 ALU 会使 EX 阶段的最坏时间变得相当长，从而拖慢整个流水线的时钟周期，因此需要设置单独的乘除部件来实现乘除槽并行执行乘除法运算。又因为乘法随随便便就会溢出，因此设置 HI 和 LO 两个寄存器可以保证数据的精度没有损失，除法的本质是试商，这个过程中商算出来了余数也就算出来了，对于既需要商又需要余数的情况，如果只提供商运算和余数运算就要进行两次基本没有区别的试商过程，这是没有意义的，使用 HI 和 LO 分别记录商和余数可以较好的解决这一问题。

（二）

参照你对延迟槽的理解，试解释“乘除槽”。

延迟槽的意义在于不管当前的指令的执行情况而执行下一条指令，相比于在当前指令后插入 nop，引入延迟槽可以在编译过程中尽可能加入无关指令以提高性能，实在无法插入无关指令再插入 nop，同理对于需要消耗较长时间的乘除运算，在进行此条指令后提供执行无关指令的能力也可以在编译阶段在乘除法计算过程中尽可能插入无关的指令来提高性能，最坏情况才是插入来等待计算完成，也就是退化到了和没有设置乘除槽的时候一样的情况。

（三）

举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑 C 语言中字符串的情况）

C 语言的一个 char 是一个字节，用一个字存一个字节显然是谋财害命行为，正常的处理方式是一个字存四个字节，这种情况下如果我需要获得一个字节但没有按字节访问的能力，而只能按字访问，读一个字节就需要读出整个字，按照需要把不要的那 24 位变成 0，按照需要移位把需要的 8 位移到低 8 位，一条指令可以完成的事情变成了判断之后三条指令才能完成的事情，而对于写一个字节，我们需要读出这个字节所在的字，把这位个字的某 8 位换成我们要存入的 8 位，如果不改 ALU 的话，判断之后要三条 r 型指令，就算修改 ALU 支持了位替换，

也要一条指令，最后再把这个改完的字存回去，这样处理性能将会是非常糟糕的，提供按字节访问内存的能力相对于只能按字访问内存，性能上将会有不小的提高。

（四）

为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

主要包括三点，一是对 ALU 和 CMP 等需要支持较多的功能的部件进行宏定义，二是变量命名把这个信号是在哪一级干啥的写清楚，格式上学的 Java 把模块命名格式当类名，信号命名格式当方法名，三是对于 ControlUnit 这种比较复杂写起来容易出错的，直接写个 C 语言程序根据真值表来生成 ControlUnit 避免出错，代码已在上文中给出，从而不再需要担心译码的时候写错了什么，要有问题一定是真值表有问题，而不会是真值表转换成 ControlUnit 的时候手残写错了之类的，而对于数据冲突，译码出统一通用的 Tnew 和 Tuse 即可简洁方便的实现冲突处理部分。

（五）

在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

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 15 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 这些涉及写 HI 和 LO 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。为了保证对 word 和 half 操作时地址合法，考虑用 cpp 模拟一个 Mars 来保证合法性，顺便也解决了除 0 问题和 add/sub/addi 溢出的问题，也可以保证跳转指令的尽量跳转和尽量不跳转，jalr 部分生成难度较大，故单独构造，此工具的代码已在自动测试或辅助工具提交窗口中提交。