# 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}共 46 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 、 DataWriteToRegSelect 、 DoubleForwardSelect 、 FlowReg_EX_MEM 、 FlowReg_ID_EX 、 FlowReg_IF_ID 、 FlowReg_MEM_WB 、 InstructionMemory 、 ProgramCounter 、 RegDstSelect 、 RegisterFile 、 RiskSolveUnit、SingleForwardSelect 共计 18 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
```

| 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 |

### 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 cuDmBranchInID
    output [3:0] cmpOperationInID
    output memWriteEnabledInID
    output [3:0] loadWriteMoodInID
    output [1:0] regDstSelectInID
    output regWriteEnabledInID
    output regConditionMoveInID
    output dmConditionMoveInID
    output [1:0] pcControlInID
    output [5:0] dataToRegSelectInID
    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 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 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 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 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 [5:0] dataToRegSelectInID | 写入寄存器的数据选择信号，流水到 MEM 逐级使用 |
| 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. 寄存器写入数据选择器 DataWriteToRegSelect

模块定义

```
module DataWriteToRegSelect(
   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. 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 |

### 9. 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 |

### 10. 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] pcControlInMEM
    input toBranchInID
    input cuRegBranchInID
    input toBranchInMEM
    input cuDmBranchInMEM
    input [31:0] branchAddrInID
    input [31:0] branchAddrInMEM
    input [25:0] jumpAddrFromImm
    input [31:0] jumpAddrFormReg1InID
    input [31:0] jumpAddrFormReg2InID
    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] 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 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] 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] 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 |

## 11. 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 |

## 12. 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 |

## 13. 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
    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 |
| 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 |

## 14. 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 |

### 15. 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
);
```

### 16. 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 [3:0] dataToRegSelectInID
input [4:0] regFinalDstInID
input regWriteEnabledInID
input [31:0] dataWriteToRegInID
input [31:0] regData1ForwardedInID
input [31:0] regData2ForwardedInID
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 [3:0] dataToRegSelectInEX
    output [4:0] regFinalDstInEX
    output regWriteEnabledInEX
    output [31:0] dataWriteToRegFormIDInEX
    output [31:0] regData1InEX
    output [31:0] regData2InEX
);
```

### 17. 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
);
```

## 18. 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
```

```
    output [31:0] currentCommandAddrInWB
    output regWriteEnabledInWB
    output [31:0] dataWriteToRegFormMEMInWB
    output [4:0] regFinalDstInWB
);
```

## （三）控制模块设计

### 1.控制单元真值表

为便于表格呈现，将表格分为三部分，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 | AluOpr | AluOpr | AluOpr | Overflow | RegBranch |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lw | 1 | 0 | 0 | 0 | 1 | 1 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 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 |
| lb | 1 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 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 |
| sb | 1 | 0 | 1 | 0 | 0 | 0 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 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 |
| lhu | 1 | 0 | 0 | 1 | 0 | 1 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 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 |
| add | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | X | X | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| addu | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | X | X | 0 | 0 | 0 | 1 | 1 | 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 |
| subu | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | X | X | 0 | 0 | 0 | 1 | 1 | 1 | 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 |

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| or | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | X | X | 0 | 0 | 0 | 0 | 0 | 1 | 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 |
| nor | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | X | X | 0 | 0 | 0 | 0 | 1 | 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 |
| sll | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 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 |
| sra | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 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 |
| srlv | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | X | X | 0 | 0 | 1 | 0 | 0 | 1 | 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 |
| sltu | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 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 |
| bne | 0 | 0 | 0 | 1 | 0 | 1 | X | X | X | X | X | X | 0 | 0 | X | X | X | X | X | X | 0 | 1 |
| bgtz | 0 | 0 | 0 | 1 | 1 | 1 | X | X | X | X | X | X | 0 | 0 | X | X | X | X | X | X | 0 | 1 |
| blez | 0 | 0 | 0 | 1 | 1 | 0 | X | X | X | X | X | X | 0 | 0 | X | X | X | X | X | X | 0 | 1 |
| bltz | 0 | 0 | 0 | 0 | 0 | 1 | \N | \0 | \0 | \0 | \0 | \0 | 0 | 0 | X | X | X | X | X | X | 0 | 1 |
| bgez | 0 | 0 | 0 | 0 | 0 | 1 | \N | \0 | \0 | \0 | \0 | \1 | 0 | 0 | X | X | X | X | X | X | 0 | 1 |
| addi | 0 | 0 | 1 | 0 | 0 | 0 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| addiu | 0 | 0 | 1 | 0 | 0 | 1 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 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 |
| ori | 0 | 0 | 1 | 1 | 0 | 1 | X | X | X | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 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 |
| lui | 0 | 0 | 1 | 1 | 1 | 1 | X | X | X | X | X | X | 1 | 1 | X | X | X | X | X | X | 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 |
| sltiu | 0 | 0 | 1 | 0 | 1 | 1 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 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 |
| jr | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | 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 |
| jalr | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | X | X | X | X | X | X | X | X | 0 | 0 |
| movz | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | X | X | X | X | X | X | X | X | X | 0 |
| movn | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | X | X | X | X | X | X | X | X | X | 0 |
| bltzal | 0 | 0 | 0 | 0 | 0 | 1 | \N | \1 | \0 | \0 | \0 | \0 | X | X | X | X | X | X | X | X | 0 | 1 |
| bgezal | 0 | 0 | 0 | 0 | 0 | 1 | \N | \1 | \0 | \0 | \0 | \1 | X | X | X | X | X | X | X | X | 0 | 1 |
| bltzalr | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | 0 | 1 |
| bgezalr | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | X | X | X | X | X | X | X | X | 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 |
| bmgezalr | 1 | 1 | 1 | 1 | 1 | 1 | X | X | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

第二部分：

| name | DmBranch | CmpIDOpr | CmpIDOpr | CmpIDOpr | CmpIDOpr | MemWrite | DMSel | DMSel | DMSel | DMSel | RegDst | RegDst | RegWrite | MoveSelD | MoveSelMEM | PcControl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lw | 0 | X | X | X | X | 0 | 0 | 0 | 0 | X | 0 | 0 | 1 | 1 | 1 | 0 |
| sw | 0 | X | X | X | X | 1 | 0 | 0 | 0 | X | X | X | 0 | X | X | 0 |
| lb | 0 | X | X | X | X | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| lbu | 0 | X | X | X | X | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| sb | 0 | X | X | X | X | 1 | 0 | 1 | 0 | X | X | X | 0 | X | X | 0 |
| lh | 0 | X | X | X | X | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| lhu | 0 | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| sh | 0 | X | X | X | X | 1 | 0 | 0 | 1 | X | X | X | 0 | X | X | 0 |
| add | 0 | X | X | X | X | 0 | X | X | X | X | 0 | 1 | 1 | 1 | 1 | 0 |
| addu | 0 | X | X | X | X | 0 | X | X | X | X | 0 | 1 | 1 | 1 | 1 | 0 |

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

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sltiu | 0 | X | X | X | X | 0 | X | X | X | X | 0 | 0 | 1 | 1 | 1 | 0 |
| j | 0 | X | X | X | X | 0 | X | X | X | X | X | X | 0 | X | X | 1 |
| jr | 0 | X | X | X | X | 0 | X | X | X | X | X | X | 0 | X | X | 1 |
| jal | 0 | X | X | X | X | 0 | X | X | X | X | 1 | X | 1 | 1 | 1 | 1 |
| jalr | 0 | X | X | X | X | 0 | X | X | X | X | 0 | 1 | 1 | 1 | 1 | 1 |
| movz | 0 | 1 | 1 | 0 | 0 | 0 | X | X | X | X | 0 | 1 | 1 | 0 | 1 | 0 |
| movn | 0 | 1 | 1 | 0 | 1 | 0 | X | X | X | X | 0 | 1 | 1 | 0 | 1 | 0 |
| bltzal | 0 | 0 | 1 | 0 | 1 | 0 | X | X | X | X | 1 | X | 1 | 0 | 1 | 0 |
| bgezal | 0 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | 1 | X | 1 | 0 | 1 | 0 |
| bltzalr | 0 | 0 | 1 | 0 | 1 | 0 | X | X | X | X | 0 | 1 | 1 | 0 | 1 | 1 |
| bgezalr | 0 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | 0 | 1 | 1 | 0 | 1 | 1 |
| bmltzalr | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | X | 1 | 1 | 0 | 1 |
| bmgezalr | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | X | 1 | 1 | 0 | 1 |

第三部分：

| name | PcControl | SelDataID | SelDataID | SelDataEX | SelDataEX | SelDataMEM | SelDataMEM | TuseRs | TuseRs | TuseRs | TuseRt | TuseRt | TuseRt | Tnew | Tnew | Tnew |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lw | 0 | X | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| sw | 0 | X | X | X | X | X | X | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| lb | 0 | X | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| lbu | 0 | X | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| sb | 0 | X | X | X | X | X | X | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| lh | 0 | X | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| lhu | 0 | X | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| sh | 0 | X | X | X | X | X | X | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| add | 0 | X | X | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| addu | 0 | X | X | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

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

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sltiu | 0 | X | X | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| j | 0 | X | X | X | X | X | X | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| jr | 1 | X | X | X | X | X | X | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| jal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| jalr | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| movz | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| movn | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| bltzal | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| bgezal | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| bltzalr | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| bgezalr | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| bmltzalr | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| bmgezalr | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 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 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;
```

### 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，下面的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。

```
#include <stdio.h>
#include <string.h>

int buffertop;
char buffer[505];
const int totnum=50;
const int totbit=42;
const int totsplit=3;

struct command
{
    char name[20]; short left;
    long long realControlNumber;
    int highSix,midFive,lowSix;
} command[50];

inline int find(char s[])
{
    for (int i=0;i<totnum;i++)
    if (strcmp(s,command[i].name)==0)
```

```
    return i+1; return 0;
}

inline int readchar()
{
    while (buffer[buffertop]!='0' && buffer[buffertop]!='1' && buffer[buffertop]!='X')
    if (buffer[buffertop]) buffertop++; else return -1;
    char c=buffer[buffertop++]; return c=='1';
}

inline long long readnum(int tot)
{
    long long ans=0; for (int i=0;i<tot;i++)
    ans=(ans<<1)|readchar(); return ans;
}

inline void tryread(int now)
{
    for (int tmp;(tmp=readchar())!=-1;command[now].left--)
    command[now].realControlNumber=(command[now].realControlNumber<<1)|tmp;
    return;
}

inline void read(int num)
{
    gets(buffer),buffertop=0;
    int tmp; char tmpname[20];
    sscanf(buffer,"%s",tmpname);
```

```
    if (!(tmp=find(tmpname)))
    {
            strcpy(command[num].name,tmpname);
            command[num].left=totbit;
            command[num].highSix=readnum(6);
            if (command[num].highSix==1) command[num].midFive=readnum(5);
            else command[num].lowSix=readnum(6);
    }
    else num=tmp-1;
    while (buffer[buffertop] && buffer[buffertop]!=32) buffertop++;
    tryread(num); return;
}

inline void print(int num)
{
    printf("current command is %s\n",command[num].name);
    if (command[num].highSix==0) printf("determined by
    lowsix %x\n",command[num].lowSix);
    else if (command[num].highSix==1) printf("determined by
    midfive %x\n",command[num].midFive);
    else printf("determined by highsix %x\n",command[num].highSix);
    printf("real control
    number %03X%08X\n",(int)((command[num].realControlNumber&((long long)(-
    1)<<32))>>32),(unsigned int)(command[num].realControlNumber));
    putchar('\n'); return;
}

inline void printbin(int num)
```

```
{
    int stk[10],tot=0;
    if (command[num].highSix==0)
    {
            for (int i=0;i<6;i++)
            stk[tot++]=command[num].lowSix&1,command[num].lowSix>>=1;
            printf("6'b"); for (int i=0;i<6;i++) putchar(stk[5-i]^48);
    }
    else if (command[num].highSix==1)
    {
            for (int i=0;i<5;i++)
            stk[tot++]=command[num].midFive&1,command[num].midFive>>=1;
            printf("5'b"); for (int i=0;i<5;i++) putchar(stk[4-i]^48);
    }
    else
    {
            for (int i=0;i<6;i++)
            stk[tot++]=command[num].highSix&1,command[num].highSix>>=1;
            printf("6'b"); for (int i=0;i<6;i++) putchar(stk[5-i]^48);
    }
    return;
}

inline void printstart()
{
    puts("\t//------------------------------------------------");
    puts("\tassign packedControl=(|currentCommand)?");
    puts("\t\t(currentCommand[31:26]==0)?judgeByLowerSix:");
```

```
    puts("\t\t(currentCommand[31:26]==1)?judgeByMiddleFive:");
    printf("\t\tjudgeByUpperSix:%d'b0;\n",totbit);
    puts("\t//------------------------------------------------");
    return;
}

inline void printhigh(bool notfinal)
{
    int con=0,fst=0; printf("\tassign judgeByUpperSix=");
    for (int i=0;i<totnum;i++) if (command[i].highSix!=0 && command[i].highSix!=1)
    con++;
    for (int i=0;i<totnum;i++) if (command[i].highSix!=0 && command[i].highSix!=1)
    {
        if (fst) putchar('\t'),putchar('\t'); else fst=1;
      printf("(currentCommand[31:26]==`%s)?(%d'h%03X%08X):",command[i].name,totbit
                ,(int)((command[i].realControlNumber&((long long)(-1)<<32))>>32)
                ,(unsigned int)(command[i].realControlNumber)),con--;
        if (con) puts("");
    }
    printf("%d'h0;",totbit);
    if (notfinal) puts("\n\t//------------------------------------------------");
    else printf("\n\nendmodule"); return;
}

inline void printmid(bool notfinal)
{
    int con=0,fst=0; printf("\tassign judgeByMiddleFive=");
    for (int i=0;i<totnum;i++) if (command[i].highSix==1) con++;
```

```
    for (int i=0;i<totnum;i++) if (command[i].highSix==1)
    {
            if (fst) putchar('\t'),putchar('\t'); else fst=1;
        printf("(currentCommand[20:16]==`%s)?(%d'h%03X%08X):",command[i].name,totbit
                ,(int)((command[i].realControlNumber&((long long)(-1)<<32))>>32)
                ,(unsigned int)(command[i].realControlNumber)),con--;
            if (con) puts("");
    }
    printf("%d'h0;",totbit);
    if (notfinal) puts("\n\t//--------------------------------------------------");
    else printf("\n\nendmodule"); return;
}

inline void printlow(bool notfinal)
{
    int con=0,fst=0; printf("\tassign judgeByLowerSix=");
    for (int i=0;i<totnum;i++) if (command[i].highSix==0) con++;
    for (int i=0;i<totnum;i++) if (command[i].highSix==0)
    {
            if (fst) putchar('\t'),putchar('\t'); else fst=1;
            printf("(currentCommand[5:0]==`%s)?(%d'h%03X%08X):",command[i].name,totbit
                ,(int)((command[i].realControlNumber&((long long)(-1)<<32))>>32)
                ,(unsigned int)(command[i].realControlNumber)),con--;
            if (con) puts("");
    }
    printf("%d'h0;",totbit);
    if (notfinal) puts("\n\t//--------------------------------------------------");
    else printf("\n\nendmodule"); return;
```

```
}

inline void printdefine()
{
    puts("`timescale 1ns/1ps\n");
    for (int i=0;i<totnum;i++)
    {
            printf("`define %s",command[i].name);
            int len=strlen(command[i].name);
            if (len>7) putchar('\t');
            else if (len>3) putchar('\t'),putchar('\t');
            else putchar('\t'),putchar('\t'),putchar('\t');
            printbin(i),puts("");
    }
    printf("\nmodule ControlUnit(");
    return;
}

int main()
{
    for (int k=0;k<totsplit && gets(buffer);k++)
    for (int i=0;i<totnum;i++) read(i);
    for (int i=0;i<totnum;i++) print(i);
    freopen("CodeGenerated.txt","w",stdout);
    printstart(),printhigh(1),printlow(1),printmid(0);
    freopen("DefineGenerated.txt","w",stdout);
    printdefine(); return 0;
}
```

### 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，下面给出的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。。

```
#include <io.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

const int tabnum=1;
int filetot,finaltot; char filedir[305],buffer[50005];
char dirfind[55][305],namefind[55][105],finalout[55][10005];

inline bool uniqueName(char s[])
{
    for (int i=0;i<filetot;i++)
    if (!strcmp(s,namefind[i]))
    return 0; return 1;
}

inline void printAllFind()
{
    puts("\nfinding finished!\n");
    for (int i=0;i<filetot;i++)
    puts(namefind[i]); puts("");
    for (int i=0;i<filetot;i++)
    puts(dirfind[i]); return;
```

```
}

inline int cmp(const void *a,const void *b)
{
    return strcmp((char*)a,(char*)b);
}

void dfs(char* dir)
{
    long long handle;
    char dirbuffer[305]; _finddata_t findData;
    strcpy(dirbuffer,dir),strcat(dirbuffer,"\\*.*");
    handle=_findfirst(dirbuffer,&findData);
    if (handle==-1) return;
    do
    {
            if (findData.attrib&_A_SUBDIR)
            {
                    if (!strcmp(findData.name,".") || !strcmp(findData.name,".."))
                    continue;
                    int len=strlen(buffer);
                    sprintf(buffer,"%s\\%s",buffer,findData.name);
                    strcpy(dirbuffer,dir),strcat(dirbuffer,"\\");
                    strcat(dirbuffer,findData.name);
                    dfs(dirbuffer),buffer[len]=0;
            }
            else
            {
```

```
                    int len=strlen(findData.name);
                    if (len>1 && findData.name[len-1]=='v' && findData.name[len-
                    2]=='.')
                    {
                        if (uniqueName(findData.name))
                        strcpy(namefind[filetot],findData.name),
                sprintf(dirfind[filetot++],"%s%s\\%s",filedir,buffer,findData.name);
                    }
            }
    }
    while (!_findnext(handle,&findData));
    _findclose(handle); return;
}

inline bool judge(int index,bool sel)
{
    if (sel) return buffer[index]==')' && buffer[index+1]==';';
    if (!(buffer[index]=='m' && buffer[index+1]=='o' && buffer[index+2]=='d' &&
    buffer[index+3]=='u'
    && buffer[index+4]=='l' && buffer[index+5]=='e' && buffer[index+6]==32))
    return 0; index+=6;
    while (buffer[index]==32) index++;
    if (!isalpha(buffer[index])) return 0;
    while (buffer[index]!='\r' && buffer[index]!='\n')
    if (buffer[index]=='(') return 1;
    else index++; return 0;
}
```

```
inline void solve(char* dir)
{
    FILE* in=fopen(dir,"rb");
    printf("\nreading \"%s\" ...\n",dir);
    int n=fread(buffer,1,50005,in);
    int mood=0,end=0,f=0; buffer[n]=0;
    for (int i=0;buffer[i];i++)
    {
            if (mood && judge(i,1))
            {
                    end=mood=finalout[finaltot++][end]=0;
                    while (finalout[finaltot-1][end]!='(')
                    putchar(finalout[finaltot-1][end++]); end=0;
                    puts (" read successfully!"),f=1;
            }
            else if (judge(i,0)) mood=1;
            if (mood) finalout[finaltot][end++]=buffer[i];
    }
    if (!f) puts("");
    fclose(in); return;
}

inline bool isreg(char s[],int index)
{
    return s[index]=='r' && s[index+1]=='e'
    && s[index+2]=='g' && s[index+3]==32;
}
```

```
inline void format(char s[])
{
    int end=0,bend=0;
    while (s[end]!='\r' && s[end]!='\n') buffer[bend++]=s[end++];
    char tmp=buffer[bend-1]; buffer[bend-1]=0;
    printf("formating %s ...\n",buffer),buffer[bend-1]=tmp;
    while (s[end]=='\r' || s[end]=='\n') end++;
    for (buffer[bend++]='\n';true;)
    {
            for (int i=0;i<tabnum;i++) buffer[bend++]='\t';
            while (s[end] && !isalpha(s[end])) end++;
            while (s[end] && s[end]!='\r' && s[end]!='\n' && s[end]!=',')
            if (isreg(s,end)) end+=4; else buffer[bend++]=s[end++];
            buffer[bend++]='\n'; if (s[end]!=',') break;
    }
    buffer[bend++]=')',buffer[bend++]=';';
    buffer[bend]=0; strcpy(s,buffer);
    puts("format successfully!\n");
    return;
}

inline void print(int num,FILE* out)
{
    int end=0,nam=0;
    fprintf(out,"%d. ",num+1);
    while (finalout[num][end]!=32) end++; end++;
    while (isalpha(finalout[num][end]) || finalout[num][end]=='_')
    namefind[num][nam++]=finalout[num][end],fputc(finalout[num][end++],out);
```

```
        namefind[num][nam]=0,fputc('\n',out);
        fputs(finalout[num],out),fputc('\n',out),fputc('\n',out);
        return;
    }

    int main(int argc,char *argv[])
    {
        puts("finding....");
        strcpy(filedir,argv[0]);
        for (int i=0;filedir[i];i++) if (filedir[i]=='/') filedir[i]=0;
        dfs(filedir),printAllFind();
        for (int i=0;i<filetot;i++) solve(dirfind[i]);
        qsort(finalout,finaltot,sizeof(finalout[0]),cmp);
        for (int i=0;i<finaltot;i++) format(finalout[i]);
        puts("\nprinting....");
        FILE *out=fopen("ModuleFound.txt","w");
        for (int i=0;i<finaltot;i++) print(i,out);
        for (int i=0;i<finaltot;i++) fprintf(out,"%s%s",
                namefind[i],i+1==finaltot?"":"、");
        puts("\nprinted successfully!\n");
        return 0;
    }
```

# 二、测试方案

## （一）典型测试样例

见思考题第九题和第十题。

### （二）数据构造方法

见思考题第九题和第十题。

### （三）自动测试工具

见思考题第九题和第十题。

## 三、思考题

### （一）

在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

数据通路方面，PC 的值更新有四种可能，第一种为默认信号，即 PC 更新为 PC+4，第二种为根据立即数的分支更新，即 PC 更新为(immAfterExtendInID<<2)+nextCommandAddrInID，第三种为根据立即数的跳转更新，即 PC 更新为{nextCommandAddr[31:28],jumpAddrFromImm[25:0],{2{1'b0}}}，第四种为跳转到寄存器的值，即 PC 更新为输入的 32 位数。

控制信号方面，需要对上述信号进行选择，具体逻辑为如果当前指令为跳转指令(cuRegBranchInID==1)则检测 toBranchInID 是否为 1，如果为 1 则检测是否根据寄存器跳转(即处理 b_condition_alr 的情况)对按第二种方式更新 PC 和按第四种方式更新 PC 进行选择，如果 toBranchInID 为 0 则按第一种方式更新 PC；如果当前指令不是跳转指令(cuRegBranchInID==0)，则进一步判断当前指令类型，如果当前指令是 j/jal，按照第三种方式更新 PC，如果当前指令是 jr/jalr，按照第四种方式更新 PC，否则按第一种方式更新 PC。此外，更新使能为 0 时对 PC 进行更新，因此数据无法及时产生需要暂停流水线的时候，只要将 PC 的写使能置 1 即可。

### （二）

对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

需要将指令地址写回寄存器的指令包括 jal/jalr/b_condition_al/b_condition_alr，这些指令都可以在 ID 阶段完成所有工作并向 PC 提供可用的下一个地址，记当前指令的地址为 PC，则由

于延迟槽下一条即 PC+4 的指令必定执行，且在执行之后跳转才会发生，PC+8 则是不跳转相比于跳转第一条出现差异的指令，写回 PC+8 既不会导致调跳转后直接 jr 时重复执行不跳转时应该执行的指令，也不会导致调跳转后直接 jr 时跳过了一些不跳转时应该执行的指令而没有执行它们。

## （三）

为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由 ALU 或者 DM 等部件来提供数据？

因为包括 ALU 和 DM 在内的重要功能部件功能较为复杂，延时较长，如果采用来自这些部件的数据，等待这些数据可以使用需要时间，将这些数据前推后在使用前推的数据的部件中仍需要时间来使数据可以使用，这就导致了更长的最短周期，又由于整个流水线 CPU 的周期是一定的，采用 ALU 或者 DM 等部件提供的数据会无差别的拖慢流水线的最短周期，相对的从存储了上一级传来的各种数据的流水级寄存器获取数据只需要经过使用前推的数据的部件这一次延迟，流水线的最短周期将会小于采用 ALU 或者 DM 等部件提供的数据的情况，且由于暂停流水线是有条件的而不是无差别的，采用从存储了上一级传来的各种数据的流水级寄存器获取数据相比于采用 ALU 或者 DM 等部件提供的数据可以有更强的均摊性能，因此所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是 ALU 或者 DM 等部件。

## （四）

如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。

当冲突发生时会造成获取的数据错误，相当多的指令序列都可以引发这个问题，列举几种满足要求的指令序列如下。

```
addi    $31,$0,12
lw      $31,0($31)
lw      $31,0($31)
lw      $31,0($31)
lw      $31,0($31)
```

```
lw      $31,0($31)

addi    $31,$0,0
lw      $31,0($31)
addu    $31,$t0,$31
jal     TAG_9
addu    $31,$31,$t0
addu    $31,$t0,$31
TAG_9:
lw      $31,-12288($31)
sw      $31,4096($31)

addi    $31,$0,4
addu    $31,$31,$t0
addi    $31,$31,4
addu    $31,$31,$t0
addu    $31,$t0,$31
addi    $t1,$31,1
beq     $31,$t1,TAG_84
lw      $31,0($31)
addi    $31,$31,4
TAG_84:

la      $t3,TAG_227
addi    $31,$0,12308
jal     TAG_226
addi    $31,$31,-2904
TAG_226:
```

```
jalr    $31,$t3
addi    $31,$31,-2916
TAG_227:
addu    $31,$31,$t0
addu    $31,$31,$t0
jr      $31
TAG_228:
la      $t9,TAG_228

la      $t3,TAG_149
la      $t4,TAG_150
addi    $31,$0,8
jal     TAG_148
addi    $31,$31,4
sw      $31,-8192($31)
TAG_148:
jalr    $31,$t3
addu    $31,$t0,$31
addu    $31,$31,$t0
TAG_149:
jalr    $31,$t4
addu    $31,$t0,$31
sw      $31,-8192($31)
TAG_150:
addi    $31,$31,-12288
addi    $t1,$31,0
beq     $t1,$31,TAG_151
sw      $31,4096($31)
```

```
addu     $31,$t0,$31
TAG_151:
```

注：以上指令序列由 c 语言生成，更详细的说明可参考思考题第 9 题和第 10 题。

## （五）

我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？

对 GRF 采用内部转发机制不是必须的，如果不对 GRF 进行内部转发，需要在冲突模块中检测需要回写且回写的地址与读取数据的地址相同，检测到上述条件成立时即可将 WB 阶段写回 GRF 的数据在 ID 阶段前推，从而实现转发需求，但这样一来我们就需要在冲突处理部分增加判断，在 ID 阶段的前推中选择原信号/来自 EX 的数据/来自 MEM 的数据/来自 WB 的数据，需要对较多的模块进行较多的修改，而如果在 GRF 内部转发，只需要在 GRF 中增加如下的两个判断：

```
assign regReadData1=(regWriteEnabled && regReadAddr1!=0 &&
    regReadAddr1==regWriteAddr)?dataWriteToReg:registerFile[regReadAddr1];
assign regReadData2=(regWriteEnabled && regReadAddr2!=0 &&
    regReadAddr2==regWriteAddr)?dataWriteToReg:registerFile[regReadAddr2];
```

即可完成数据的转发，不需要修改其它模块或在顶层模块中增加内容，从高内聚低耦合的角度来讲对 GRF 采用内部转发来实现数据的转发更优。

## （六）

为什么 0 号寄存器需要特殊处理？

因为 0 号寄存器的特点在于需要始终保持其值是 0，如果不对其进行特殊处理，当出现向 0 号寄存器写入非 0 值(当然这样是无效的)，随后从 0 号寄存器读取值的情况时，向 0 号寄存器写入的非 0 值可能被前推给需要用到来自 0 号寄存器值的部件，从而导致指令结果错误。

## （七）

什么是“最新产生的数据”？

对指令序列，考虑其中有写行为的有序子集 A1A2...An，这 n 条指令中最后执行的 An 即为最新产生数据的指令，对流水线 CPU，最新产生的数据为最新产生数据的指令得到结果的部件流水一级后的结果，例如最新产生数据的指令为 r 型指令，其结果在 ALU 产生，则最新产生的数据为经过 EX/MEM 的上述 ALU 结果，最新产生数据的指令为 l 型指令，其结果在 DM 产生，则最新产生的数据为经过 MEM/WB 的上述 DM 结果。

## （八）

在 AT 方法讨论转发条件的时候，只提到了“供给者需求者的 A 相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了用且仅用 A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢？

由于转发条件为供给者需求者的 A 相同且不为 0，只需要在翻译 A 的过程中根据 we 进行一次选择，如果 we 为 0 则将 A 改为 0，即可实现对 we 的判断。此外，笔者的实现中没有采取上述方式，而是选择将 we 信号一并传入冲突模块进行处理，这是基于更灵活的处理 movz/b_condition_al/b_condition_alr 等条件写入指令乃至在 DM 阶段才可以确定是否写入的指令考虑的。

## （九）

请在文档中附上你的测试程序，并明确说明测试程序的测试期望，即应该得到怎样的运行结果。P5 课上将会重点查看测试程序。

测试程序分两部分，第一部分为正确性测试，沿用了 P4 的测试数据，只是增加了延迟槽中的指令来确保测试程序行为的正确性，同时对 CPU 中延迟槽的功能是否正常进行测试，共计 11 个测试点，概述如下，但为保证测试强度，这些数据都有较大的规模，将其粘贴在此处将严重影响阅读体验，因此这些在此不给出测试程序，测试程序已在辅助测试工具部分提交，且在讨论区公开。

testpoint1：主要测试四个跳转指令，延迟槽采用双 i 验证

包含的指令：add,addi,beq,j,jal,jalr,jr,slti,sw

testpoint2：主要测试四个跳转指令，延迟槽采用双 l 验证

包含的指令：add,addi,beq,j,jal,jalr,jr,slti,sw

testpoint3：主要测试六个条件跳转指令，延迟槽采用双 i 验证

包含的指令：addi,addiu,beq,bgez,bgtz,blez,bltz,bne,j,slti,sw

testpoint4：主要测试六个条件跳转指令，延迟槽采用双 l 验证

包含的指令：addi,addiu,beq,bgez,bgtz,blez,bltz,bne,j,slti,sw

testpoint5：主要测试若干 R 类型的指令，有点多所以分了两个测试点

包含的指令：addi,addiu,addu,and,lw,nor,or,sllv,slt,sltu,srav,srlv,subu,sw,xor

testpoint6：主要测试若干 R 类型的指令，有点多所以分了两个测试点

包含的指令：add,addi,addiu,addu,and,lw,nor,or,sllv,slt,sltu,srav,srlv,sub,subu,sw,xor

testpoint7：主要测试八个内存读写指令

包含的指令：addi,addiu,beq,j,lb,lbu,lh,lhu,lw,sb,sh,sw

注意：DM 用的比较多，要开到 0x2ffc

testpoint8：主要测试若干 I 类型的指令，有点多所以分了两个测试点

包含的指令：addi,addiu,andi,ori,sll,srl,sw

testpoint9：主要测试若干 I 类型的指令，有点多所以分了两个测试点

包含的指令：addi,lui,slti,sltiu,sra,sw,xori

testpoint10：实质是跑水仙花数，找个比较长的有意义的程序跑下试试

包含的指令：add,addi,beq,j,jal,jr,slt,slti,sw

注意：10ns 一周期，要 20000us

testpoint11：实质是跑 n=100 的快排，找个有递归的有意义的程序跑下试试

包含的指令：add,addi,addiu,beq,bgez,bgtz,blez,bltz,bne,j,jal,jr,lw,sll,srl,sub,sw

注意：有递归，DM 要开到 0x2ffc

第二部分为冲突专项测试，挑选了一些有代表性的指令人为构造冲突序列进行测试，这部分采用了覆盖性测试，测试点由 C 语言程序生成共计 1360 个，相关测试点已打包在讨论区公开，测试点生成程序如下，该程序也在辅助测试工具部分进行了提交，具体的测试点构造思路见思考题第 10 题。

```
#pragma GCC optimize(2)

#pragma GCC optimize(3,"Ofast","inline")
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define DEBUGMOOD 0

const int jumptonew=880,totjump=7;
const int totdb=5,totuse=11,totnorm=7;
const int levelbefore=4,dbrep=2,tonewfile=850;

int fleptr,tagptr,nownum,hasdeal;
char buffer[40][65]; int buffertop,uset;
int initarray[35],stack[10],forwardtag[40];

inline void swap(int *x,int *y)
{
    int tmp=*x; *x=*y;
    *y=tmp; return;
}

inline void reinitfst()
{
    char tmp[111];
    sprintf(tmp,"testpoint%d.asm",fleptr);
    tagptr=0,nownum=100,fleptr++;
    freopen(tmp,"w",stdout);
    for (int i=0;i<30;i++)
    {
            printf("addi\t$s0,$0,%d\n",initarray[i]);
```

```
            printf("sw\t\t$s0,%d($0)\n",(i+1)<<2);
    }
    puts("\n#--------------\n");
    for (int i=1;i<32;i++)
            printf("addi\t$%d,$0,23333\n",i);
    printf("\naddi\t$t0,$0,8\n");
    printf("addi\t$t2,$0,-12288\n");
    puts("\n#--------------\n");
    return;
}

inline void reinitnxt()
{
    char tmp[111];
    sprintf(tmp,"testpoint%d.asm",fleptr);
    tagptr=151,nownum=434,fleptr++;
    freopen(tmp,"w",stdout);
    puts("j\t\tSTART\n\t\tnop");
    for (int i=0;i<150;i++)
            printf("TAG_%d%sjr\t$t9\n\t\tnop\n",i,i>99?":":":\t");
    puts("START:");
    puts("\n#--------------\n");
    for (int i=0;i<50;i++)
    {
            printf("addi\t$s0,$0,%d\n",12292+(i<<2));
            printf("sw\t\t$s0,%d($0)\n",i<<2);
    }
    puts("\n#--------------\n");
```

```
    for (int i=1;i<32;i++) printf("addi\t$%d,$0,23333\n",i);
    puts("\naddi\t$t0,$0,4");
    puts("\n#--------------\n");
    return;
}

inline void endfile(int toinit)
{
    int bgn=2048,mem=0; puts("#--------------\n");
    printf("addi\t$t0,$0,2222\n"),printf("sw\t\t$t0,%d($0)\n",bgn+(mem<<2)),mem++;
    printf("addi\t$t0,$0,3333\n"),printf("sw\t\t$t0,%d($0)\n",bgn+(mem<<2)),mem++;
    printf("addi\t$t0,$0,4444\n"),printf("sw\t\t$t0,%d($0)",bgn+(mem<<2)),mem++;
    if (toinit==1) reinitfst(); else if (toinit) reinitnxt(); return;
}

inline void bothinit()
{
    puts("testpoint generate started!"),puts("waiting...");
    srand(19260817); for (int i=0;i<30;i++) initarray[i]=(i+1)<<2;
    for (int i=0;i<30;i++) swap(&initarray[i],&initarray[rand()%30]);
    fleptr=1; reinitfst(); return;
}

inline void printgap(int doit,int num)
{
    if (num==0)
    {
            strcpy(buffer[buffertop++],hasdeal?"lw\t\t$31,-
```

```
            12288($31)":"lw\t\t$31,0($31)");
            if (hasdeal && !doit) hasdeal=0;
    }
    else if (num==1) strcpy(buffer[buffertop++],hasdeal?"sw\t\t$31,-
    8192($31)":"sw\t\t$31,4096($31)");
    else if (num==2) strcpy(buffer[buffertop++],"addu\t$31,$t0,$31");
    else if (num==3) strcpy(buffer[buffertop++],"addu\t$31,$31,$t0");
    else if (num==4) strcpy(buffer[buffertop++],"addi\t$31,$31,4");
    return;
}

inline void printget(int num)
{
    if (num==0) strcpy(buffer[buffertop++],hasdeal?"lw\t\t$31,-
    12288($31)":"lw\t\t$31,0($31)"),nownum++,hasdeal=0;
    else if (num==1)
    strcpy(buffer[buffertop++],hasdeal?"addu\t$31,$t2,$31":"addu\t$31,$t0,$31"),nownum
    ++,hasdeal=0;
    else if (num==2)
    strcpy(buffer[buffertop++],hasdeal?"addu\t$31,$31,$t2":"addu\t$31,$31,$t0"),nownum
    ++,hasdeal=0;
    else if (num==3) strcpy(buffer[buffertop++],hasdeal?"addi\t$31,$31,-
    12288":"addi\t$31,$31,4"),nownum++,hasdeal=0;
    else if (num==4)
    {
            sprintf(buffer[buffertop++],"jal\t\tTAG_%d",tagptr),nownum+=3,hasdeal=1;
            printgap(0,rand()%5),printgap(1,rand()%5),sprintf(buffer[buffertop++],"TAG
            _%d:",tagptr++);
```

```
        }
        else if (num==5)
        {
    sprintf(buffer[buffertop],"la\t\t$t%d,TAG_%d",uset,tagptr),forwardtag[buffertop++]=1;
                sprintf(buffer[buffertop++],"jalr\t$31,$t%d",uset++),hasdeal=1,printgap(0,
                rand()%5),printgap(1,rand()%5);
                sprintf(buffer[buffertop++],"TAG_%d:",tagptr++),nownum+=4;
        }
        else if (num==6) strcpy(buffer[buffertop++],"nop"),nownum++;
        return;
    }

    inline void printjump(int num)
    {
        if (num==0) strcpy(buffer[buffertop++],"lw\t\t$31,-12288($31)"),nownum++;
        else if (num==1) strcpy(buffer[buffertop++],"addu\t$31,$t0,$31"),nownum++;
        else if (num==2) strcpy(buffer[buffertop++],"addu\t$31,$31,$t0"),nownum++;
        else if (num==3) strcpy(buffer[buffertop++],"addi\t$31,$31,4"),nownum++;
        else if (num==4)
        {
                sprintf(buffer[buffertop++],"jal\t\tTAG_%d",tagptr),nownum+=2;
                sprintf(buffer[buffertop++],"addi\t$31,$31,-%d",(nownum-12)<<2);
                sprintf(buffer[buffertop++],"TAG_%d:",tagptr++);
        }
        else if (num==5)
        {
    sprintf(buffer[buffertop],"la\t\t$t%d,TAG_%d",uset,tagptr),forwardtag[buffertop++]=1;
                sprintf(buffer[buffertop++],"jalr\t$31,$t%d",uset++),nownum+=3;
```

```
            sprintf(buffer[buffertop++],"addi\t$31,$31,-%d",(nownum-
            12)<<2),sprintf(buffer[buffertop++],"TAG_%d:",tagptr++);
    }
    else if (num==6) strcpy(buffer[buffertop++],"nop"),nownum++;
    return;
}

inline void printuse(int num)
{
    if (num==0) strcpy(buffer[buffertop++],hasdeal?"lw\t\t$31,-
    12288($31)":"lw\t\t$31,0($31)"),nownum++;
    else if (num==1) strcpy(buffer[buffertop++],hasdeal?"sw\t\t$31,-
    8192($31)":"sw\t\t$31,4096($31)"),nownum++;
    else if (num==2) strcpy(buffer[buffertop++],"addu\t$31,$t0,$31"),nownum++;
    else if (num==3) strcpy(buffer[buffertop++],"addu\t$31,$31,$t0"),nownum++;
    else if (num==4)
    {
            sprintf(buffer[buffertop++],"beq\t\t$31,$31,TAG_%d",tagptr);
            printgap(0,rand()%5),printgap(0,rand()%5),sprintf(buffer[buffertop++],"TAG
            _%d:",tagptr++),nownum+=3;
    }
    else if (num==5)
    {
            sprintf(buffer[buffertop++],"beq\t\t$31,$0,TAG_%d",tagptr);
            printgap(0,rand()%5),printgap(0,rand()%5),sprintf(buffer[buffertop++],"TAG
            _%d:",tagptr++),nownum+=3;
    }
    else if (num==6)
```

```
{
        strcpy(buffer[buffertop++],"addi\t$t1,$31,0"),sprintf(buffer[buffertop++],
        "beq\t\t$t1,$31,TAG_%d",tagptr);
        printgap(0,rand()%5),printgap(0,rand()%5),sprintf(buffer[buffertop++],"TAG
        _%d:",tagptr++),nownum+=4;
}
else if (num==7)
{
        strcpy(buffer[buffertop++],"addi\t$t1,$31,1"),sprintf(buffer[buffertop++],
        "beq\t\t$31,$t1,TAG_%d",tagptr);
        printgap(0,rand()%5),printgap(1,rand()%5),sprintf(buffer[buffertop++],"TAG
        _%d:",tagptr++),nownum+=4;
}
else if (num==8) strcpy(buffer[buffertop++],"addi\t$31,$31,4"),nownum++;
else if (num==9)
{
        sprintf(buffer[buffertop++],"la\t\t$31,TAG_%d",tagptr),strcpy(buffer[buffe
        rtop++],"jr\t\t$31"),hasdeal=1;
        printgap(0,rand()%5),printgap(1,rand()%5),nownum+=4,sprintf(buffer[buffert
        op++],"TAG_%d:",tagptr++);
}
else if (num==10)
{
        sprintf(buffer[buffertop++],"la\t\t$31,TAG_%d",tagptr),strcpy(buffer[buffe
        rtop++],"jalr\t$t3,$31"),hasdeal=1;
        printgap(0,rand()%5),printgap(1,rand()%5),sprintf(buffer[buffertop++],"TAG
        _%d:",tagptr++),nownum+=4;
}
```

```
    return;
}

inline int hasdb(int src)
{
    if (src>3 && src!=8) return 1;
    for (int i=0;i<levelbefore;i++)
    if (stack[i]>3) return 1; return 0;
}

inline void fixbuffer()
{
    for (int i=0;i<buffertop;i++) if (forwardtag[i]) puts(buffer[i]);
    for (int i=0;i<buffertop;i++) if (!forwardtag[i]) puts(buffer[i]);
    puts(""); return;
}

inline void printwithdb(int src)
{
    if (DEBUGMOOD) for (int i=0;i<levelbefore;i++) printf("%d ",stack[i]);
    if (DEBUGMOOD) printf("\n%d\n",src);
    for (int i=0,f=1,db=hasdb(src);f || (i<dbrep && db);i++,f=0)
    {
            if (!DEBUGMOOD && nownum>tonewfile) endfile(1);
            buffertop=0,uset=3,memset(forwardtag,0,sizeof(forwardtag));
            sprintf(buffer[buffertop++],"addi\t$31,$0,%d",(rand()&7)<<2);
            nownum++,hasdeal=0; for (int i=0;i<levelbefore;i++) printget(stack[i]);
            printuse(src),fixbuffer();
```

```
    }
    return;
}

inline void printwithj(int src)
{
    if (DEBUGMOOD) for (int i=0;i<levelbefore;i++) printf("%d ",stack[i]);
    if (DEBUGMOOD) printf("\n%d\n",src);
    for (int i=0;i<2;i++)
    {
        if (!DEBUGMOOD && nownum>jumptonew) endfile(2);
        buffertop=0,uset=3,memset(forwardtag,0,sizeof(forwardtag));
        sprintf(buffer[buffertop++],"addi\t$31,$0,%d",12292+((rand()&7)<<2));
        nownum+=2; for (int i=0;i<levelbefore;i++) printjump(stack[i]);
        if (i) strcpy(buffer[buffertop++],"jr\t\t$31"),
            sprintf(buffer[buffertop++],"TAG_%d:",tagptr),nownum+=2,
            sprintf(buffer[buffertop++],"la\t\t$t9,TAG_%d",tagptr++);
        else strcpy(buffer[buffertop++],"jalr\t$t9,$31\nnop"); fixbuffer();
    }
    return;
}

void dfsnorm(int num)
{
    if (num==levelbefore) for (int j=0;j<totuse;j++) printwithdb(j);
    else for (int i=0;i<totnorm;i++) stack[num]=i,dfsnorm(num+1);
    return;
}
```

```
void dfsjump(int num)
{
    if (num==levelbefore) for (int j=0;j<totjump;j++) printwithj(j);
    else for (int i=0;i<totjump;i++) stack[num]=i,dfsjump(num+1);
    return;
}

inline void allend()
{
    freopen("CON","w",stdout);
    printf("all %d testpoint generated!",fleptr-1);
    puts("\nwish you ak them all!"); return;
}

int main()
{
    bothinit(),dfsnorm(0),endfile(2);
    dfsjump(0),endfile(0),allend();
    return 0;
}
```

测试程序的期望输出由魔改版 Mars 给出，即对其中的 RegisterFile 类和 Memory 进行了修改，在对 GRF 或 DM 进行了写入操作后，输出写入的信息，测试点正确性的验证和输出的结果与答案对比的过程同样由 C 语言程序自动执行，测试点正确性验证和自动测试程序代码如下，上述程序也在辅助测试工具部分进行了提交。

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

FILE* check;
char syscal[505],buffer[1000005];

int main()
{
    int n; char* p; scanf("%d",&n);
    for (int i=0;i<n;i++)
    {
            sprintf(buffer,"testpoint%d.asm",i+1);
            sprintf(syscal,"java -jar mars.jar db nc mc CompactDataAtZero dump .text
            HexText code.txt >ans.txt %s",buffer);
            system(syscal),check=fopen("ans.txt","r");
            fread(buffer,1000000,1,check),fclose(check);
            if ((p=strstr(buffer,"Error")))
            {
                    char* np=strstr(p,"errors.");
                    *(np+7)=0; puts(""),puts(p); break;
            }
            else printf("testpoint%d.asm successfully checked\n",i+1);
    }
    return 0;
}

#pragma GCC optimize(2)
```

```
#pragma GCC optimize(3,"Ofast","inline")


#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SPLITTEST 0


bool isAK=1;
const int time=20000;
const int pointnum=11;
const char isedir[105]="D:\\XilinxISE\\14.7\\ISE_DS\\ISE";
char filedir[105],testpoint[105],buffer1[505],buffer2[505];


inline void gettcl()
{
    FILE* fpr=fopen("mips.tcl","w");
    fprintf(fpr,"run %dus;\nexit\n",time);
    fclose(fpr); return;
}


void dfs(char* dir,FILE* fpr)
{
    long long handle;
    char dirbuffer[105];
    _finddata_t findData;
    strcpy(dirbuffer,dir),strcat(dirbuffer,"\\*.*");
    handle=_findfirst(dirbuffer,&findData);
```

```
    if (handle==-1) return;
    do
    {
        if (findData.attrib&_A_SUBDIR)
        {
            if (!strcmp(findData.name,".") || !strcmp(findData.name,".."))
            continue;
            int len=strlen(buffer1);
            sprintf(buffer1,"%s\\%s",buffer1,findData.name);
            strcpy(dirbuffer,dir),strcat(dirbuffer,"\\");
            strcat(dirbuffer,findData.name);
            dfs(dirbuffer,fpr),buffer1[len]=0;
        }
        else
        {
            int len=strlen(findData.name);
            if (len>1 && findData.name[len-1]=='v' && findData.name[len-
            2]=='.')
            fprintf(fpr,"verilog work \"%s\\%s\"\n",buffer1,findData.name);
        }
    }
    while (!_findnext(handle,&findData));
    _findclose(handle); return;
}

inline void getprj()
{
    FILE* fpr=fopen("mips.prj","w");
```

```
    strcpy(buffer1,filedir);
    dfs(filedir,fpr); return;
}

inline int buffercmp(char* buffer1,char* buffer2)
{
    int index1=0,index2=0;
    while (buffer1[index1] && buffer1[index1]!='@') index1++;
    while (buffer2[index2] && buffer2[index2]!='@') index2++;
    return strcmp(buffer1+index1,buffer2+index2);
}

inline void filecompare(const char* s)
{
    int cnt=0;
    FILE* ans=fopen("ans.txt","r");
    FILE* out=fopen("out.txt","r");
    for (int i=0;i<5;i++) fgets(buffer1,512,out);
    while (fgets(buffer2,512,ans))
    {
        cnt++;
        if (strlen(buffer2)<3) break;
        if (!fgets(buffer1,512,out))
        {
            printf("test point %s wrong at line %d\n",s,cnt);
            puts("your answer is fewer than expected answer");
            isAK=0; return;
        }
```

```
                else if (buffercmp(buffer1,buffer2))
                {
                        printf("test point %s wrong at line %d\n",s,cnt);
                        printf("expected answer is %s\n",buffer2);
                        printf("your answer is %s\n",buffer1);
                        isAK=0; return;
                }
        }
        printf("test point %s accepted\n",s);
        return;
    }

    inline void solve(const char* s)
    {
        printf("\ntesting %s\n",s);
        sprintf(buffer1,"java -jar mars.jar a nc mc CompactDataAtZero dump .text HexText
code.txt %s",s),system(buffer1);
        sprintf(buffer1,"java -jar mars.jar db nc mc CompactDataAtZero dump .text HexText
code.txt >ans.txt %s",s),system(buffer1);
        sprintf(buffer1,"%s\\bin\\nt64\\fuse --nodebug  --prj mips.prj -o mips.exe
mips_tb >log.txt",isedir),system(buffer1);
        system("mips.exe -nolog -tclbatch mips.tcl >out.txt");
        filecompare(s); return;
    }

    int main(int argc,char *argv[])
    {
        puts("autotest started");
```

```
        strcpy(filedir,argv[0]);
        for (int i=0;filedir[i];i++)
                if (filedir[i]=='/') filedir[i]=0;
        sprintf(buffer1,"XILINX=%s",isedir);
        putenv(buffer1),gettcl(),getprj();
        puts("initial secceed");
        for (int i=0;i<pointnum;i++)
        {
                sprintf(testpoint,"testpoint%d.asm",i+1);
                solve(testpoint); if (SPLITTEST) getchar();
        }
        if (isAK) puts("\ncongratulation!\nyou all killed!");
        return 0;
}
```

（十）

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。

与冲突相关的测试为使用 C 语言批量构造的覆盖性测试，考虑到 Mips 中指令有较明显的类别，可将待测试指令分为 l 型 s 型 r 型 i 型 b 型 j 型，分别选取其中有代表性的指令进行测试，本测试中 l 型指令选择 lw，s 型指令选择 sw，r 型指令选择 addu，i 型指令选择 addi(不考虑溢出)，b 型指令选择 beq，j 型指令则较为特殊，jal 产生数据不需要数据，jr 需要数据不产生数据，jalr 既需要数据又产生数据，因此对这三个 j 指令都进行测试，即测试 lw/sw/addu/addi/beq/jal/jr/jalr，注意到流水线中最多只可能存在五条正在执行的指令，冲突只会在五条指令间发生，因此只需要枚举五条指令，前四条指令产生数据或为 nop(通过插入 nop 来

保证不仅仅只测试相邻两条冲突的情况)，最后一条指令使用数据，在上述要测试的指令集中，产生数据的指令包括 lw/addu/addi/jal/jalr，使用数据的指令包括 lw/sw/addu/addi/beq/jr/jalr，因此按照四条 lw/addu/addi/jal/jalr/nop 的组合和 lw/sw/addu/addi/beq/jr/jalr 枚举即可。

为了保证测试程序的正确性，进行的操作包括对内存进行如下初始化以保证可以连续 lw 而不导致错误发生。

```
addi  $s0,$0,104
sw    $s0,4($0)
addi  $s0,$0,12
sw    $s0,8($0)
addi  $s0,$0,92
sw    $s0,12($0)
addi  $s0,$0,80
sw    $s0,16($0)
addi  $s0,$0,76
sw    $s0,20($0)
addi  $s0,$0,68
sw    $s0,24($0)
addi  $s0,$0,56
sw    $s0,28($0)
addi  $s0,$0,108
sw    $s0,32($0)
addi  $s0,$0,64
sw    $s0,36($0)
addi  $s0,$0,4
sw    $s0,40($0)
addi  $s0,$0,16
sw    $s0,44($0)
addi  $s0,$0,116
```

```
sw    $s0,48($0)
addi  $s0,$0,28
sw    $s0,52($0)
addi  $s0,$0,96
sw    $s0,56($0)
addi  $s0,$0,44
sw    $s0,60($0)
addi  $s0,$0,8
sw    $s0,64($0)
addi  $s0,$0,20
sw    $s0,68($0)
addi  $s0,$0,40
sw    $s0,72($0)
addi  $s0,$0,88
sw    $s0,76($0)
addi  $s0,$0,72
sw    $s0,80($0)
addi  $s0,$0,32
sw    $s0,84($0)
addi  $s0,$0,100
sw    $s0,88($0)
addi  $s0,$0,52
sw    $s0,92($0)
addi  $s0,$0,120
sw    $s0,96($0)
addi  $s0,$0,24
sw    $s0,100($0)
addi  $s0,$0,60
```

```
sw    $s0,104($0)
addi  $s0,$0,84
sw    $s0,108($0)
addi  $s0,$0,112
sw    $s0,112($0)
addi  $s0,$0,48
sw    $s0,116($0)
addi  $s0,$0,36
sw    $s0,120($0)
```

设置如下公共跳转区来保证跳转前不需要通过 la 准备跳转地址，从而可以测试到 l 型指令或 r 型指令接跳转指令写跳转目标由上述 l 型指令或 r 型指令得到的情况。

```
j     START
         nop
TAG_0:   jr $t9
         nop
TAG_1:   jr $t9
         nop
TAG_2:   jr $t9
         nop
TAG_3:   jr $t9
         nop
TAG_4:   jr $t9
         nop
TAG_5:   jr $t9
         nop
TAG_6:   jr $t9
         nop
TAG_7:   jr $t9
```

```
        nop
TAG_8:  jr $t9
        nop
TAG_9:  jr $t9
        nop
TAG_10: jr $t9
        nop
TAG_11: jr $t9
        nop
TAG_12: jr $t9
        nop
TAG_13: jr $t9
        nop
TAG_14: jr $t9
        nop
TAG_15: jr $t9
        nop
TAG_16: jr $t9
        nop
TAG_17: jr $t9
        nop
TAG_18: jr $t9
        nop
TAG_19: jr $t9
        nop
TAG_20: jr $t9
        nop
TAG_21: jr $t9
```

```
        nop
TAG_22: jr $t9
        nop
TAG_23: jr $t9
        nop
TAG_24: jr $t9
        nop
TAG_25: jr $t9
        nop
TAG_26: jr $t9
        nop
TAG_27: jr $t9
        nop
TAG_28: jr $t9
        nop
TAG_29: jr $t9
        nop
TAG_30: jr $t9
        nop
TAG_31: jr $t9
        nop
TAG_32: jr $t9
        nop
TAG_33: jr $t9
        nop
TAG_34: jr $t9
        nop
TAG_35: jr $t9
```

```
        nop
TAG_36: jr $t9
        nop
TAG_37: jr $t9
        nop
TAG_38: jr $t9
        nop
TAG_39: jr $t9
        nop
TAG_40: jr $t9
        nop
TAG_41: jr $t9
        nop
TAG_42: jr $t9
        nop
TAG_43: jr $t9
        nop
TAG_44: jr $t9
        nop
TAG_45: jr $t9
        nop
TAG_46: jr $t9
        nop
TAG_47: jr $t9
        nop
TAG_48: jr $t9
        nop
TAG_49: jr $t9
```

```
        nop
TAG_50: jr $t9
        nop
TAG_51: jr $t9
        nop
TAG_52: jr $t9
        nop
TAG_53: jr $t9
        nop
TAG_54: jr $t9
        nop
TAG_55: jr $t9
        nop
TAG_56: jr $t9
        nop
TAG_57: jr $t9
        nop
TAG_58: jr $t9
        nop
TAG_59: jr $t9
        nop
TAG_60: jr $t9
        nop
TAG_61: jr $t9
        nop
TAG_62: jr $t9
        nop
TAG_63: jr $t9
```

```
        nop
TAG_64: jr $t9
        nop
TAG_65: jr $t9
        nop
TAG_66: jr $t9
        nop
TAG_67: jr $t9
        nop
TAG_68: jr $t9
        nop
TAG_69: jr $t9
        nop
TAG_70: jr $t9
        nop
TAG_71: jr $t9
        nop
TAG_72: jr $t9
        nop
TAG_73: jr $t9
        nop
TAG_74: jr $t9
        nop
TAG_75: jr $t9
        nop
TAG_76: jr $t9
        nop
TAG_77: jr $t9
```

```
        nop
TAG_78: jr $t9
        nop
TAG_79: jr $t9
        nop
TAG_80: jr $t9
        nop
TAG_81: jr $t9
        nop
TAG_82: jr $t9
        nop
TAG_83: jr $t9
        nop
TAG_84: jr $t9
        nop
TAG_85: jr $t9
        nop
TAG_86: jr $t9
        nop
TAG_87: jr $t9
        nop
TAG_88: jr $t9
        nop
TAG_89: jr $t9
        nop
TAG_90: jr $t9
        nop
TAG_91: jr $t9
```

```
        nop
TAG_92: jr $t9
        nop
TAG_93: jr $t9
        nop
TAG_94: jr $t9
        nop
TAG_95: jr $t9
        nop
TAG_96: jr $t9
        nop
TAG_97: jr $t9
        nop
TAG_98: jr $t9
        nop
TAG_99: jr $t9
        nop
TAG_100:jr $t9
        nop
TAG_101:jr $t9
        nop
TAG_102:jr $t9
        nop
TAG_103:jr $t9
        nop
TAG_104:jr $t9
        nop
TAG_105:jr $t9
```

```
        nop
TAG_106:jr $t9
        nop
TAG_107:jr $t9
        nop
TAG_108:jr $t9
        nop
TAG_109:jr $t9
        nop
TAG_110:jr $t9
        nop
TAG_111:jr $t9
        nop
TAG_112:jr $t9
        nop
TAG_113:jr $t9
        nop
TAG_114:jr $t9
        nop
TAG_115:jr $t9
        nop
TAG_116:jr $t9
        nop
TAG_117:jr $t9
        nop
TAG_118:jr $t9
        nop
TAG_119:jr $t9
```

```
        nop
TAG_120:jr $t9
        nop
TAG_121:jr $t9
        nop
TAG_122:jr $t9
        nop
TAG_123:jr $t9
        nop
TAG_124:jr $t9
        nop
TAG_125:jr $t9
        nop
TAG_126:jr $t9
        nop
TAG_127:jr $t9
        nop
TAG_128:jr $t9
        nop
TAG_129:jr $t9
        nop
TAG_130:jr $t9
        nop
TAG_131:jr $t9
        nop
TAG_132:jr $t9
        nop
TAG_133:jr $t9
```

```
        nop
TAG_134:jr $t9
        nop
TAG_135:jr $t9
        nop
TAG_136:jr $t9
        nop
TAG_137:jr $t9
        nop
TAG_138:jr $t9
        nop
TAG_139:jr $t9
        nop
TAG_140:jr $t9
        nop
TAG_141:jr $t9
        nop
TAG_142:jr $t9
        nop
TAG_143:jr $t9
        nop
TAG_144:jr $t9
        nop
TAG_145:jr $t9
        nop
TAG_146:jr $t9
        nop
TAG_147:jr $t9
```

```
        nop
TAG_148:jr $t9
        nop
TAG_149:jr $t9
        nop
```

START:此外，为保证测试程序的正确性和强度，还进行了包括寄存器值适配(如寄存器的值被写为 0x3000 附近，第一时间将其调增到 0 附近以保证 lw/sw 的正确性)，地址准备语句提前(在一个测试单元生成后，将其中 la 等无关冲突的语句提前，以保证冲突的连贯性)等操作，在此不一一赘述，可参考思考题第九题中给出的代码。