# P6-较为复杂的流水线CPU

## 一、设计说明

1. 处理器应支持MIPS-C3指令集。

MIPS-C3={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、

SUB、 SUBU、 MULT、 MULTU、 DIV、 DIVU、 SLL、 SRL、 SRA、 SLLV、

SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、

XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、

BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}

2. 处理器为流水线设计。

## 二、数据通路设计

### 1. PC

#### (1) **模块接口定义**

表1 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| pc.v | module pc(  input clk,  input reset,  input [31:0] nextPC,  output reg [31:0] PC); |

#### (2) **接口说明**

表2 PC接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 同步复位信号  1：复位  0：无效 |
| 3 | pc\_en | I | PC更新使能信号 |
| 4 | nextPC | I | PC输入值 |
| 5 | PC | O | PC输出值 |

#### (3)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | reset有效时，PC被设置为0x00003000 |
| 2 | PC更新 | 时钟上升沿来临且pc\_en有效时，PC值更新为nextPC |

### 2. IM

#### (1) **模块接口定义**

表4**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| im.v | module im(  input [31:0] PC,  output [31:0] instruction); |

#### (2) **接口说明**

表5接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | PC | I | 32位PC值 |
| 2 | instruction | O | 取出32位指令值 |

#### (3)功能定义

表6功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 取指令 | 根据PC值取出指令 |

### 3. IF/ID

#### (1) **模块接口定义**

表7 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IFID.v | module IFID(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] PC\_F,  input [31:0] Instr\_F,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] PC\_D,  output reg[31:0] PC4\_D,  output reg[31:0] PC8\_D,  output reg[31:0] Instr\_D  ); |

#### (2) **接口说明**

表8 接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 同步复位信号，将Instr\_D的值全部清零  1：复位  0：无效 |
| 3 | i\_en | I | 由暂停模块控制的使能信号 |
| 4 | PC\_F | I | PC在F级的值 |
| 5 | Instr\_F | I | Instr在F级的值 |
| 6 | PC\_D | O | PC在D级的值 |
| 7 | PC4\_D | O | PC+4在D级的值 |
| 8 | PC8\_D | O | PC+8在D级的值 |
| 9 | Instr\_D | O | Instr在D级的值 |

#### (3)功能定义

表9 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存IF/ID的流水线信号 |

### 4. NPC

#### (1) **模块接口定义**

表10 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| npc.v | module npc(  input i\_Zero,  input [2:0] i\_branch,  input [25:0] i\_imm26,  input [31:0] i\_jr\_addr,  input [31:0] i\_PC,  input [31:0] i\_PC4\_D,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg [31:0] o\_nextPC  ); |

#### (2) **接口说明**

表11 接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | zero | I | 判断PC是否满足beq指令跳转条件  1：满足  0：不满足 |
| 2 | branch | I | 判断当前指令是否需要跳转 |
| 3 | imm26 | I | j/offset |
| 4 | jr\_addr | I | jr指令跳转地址 |
| 5 | PC | I | PC输入值 |
| 6 | PC4\_D | I | PC+4输入值 |
| 7 | nextPC | O | nextPC计算结果 |

#### (3)功能定义

表12 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 计算PC的下一个值 | 计算PC的下一个值 |

### 5. GRF

#### (1) **模块接口定义**

表13 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| grf.v | module grf(  input clk,  input reset,  input WE,  input [4:0] A1,  input [4:0] A2,  input [4:0] A3,  input [31:0] WD,  input [31:0] PC,  output [31:0] RD1,  output [31:0] RD2); |

#### (2) **接口说明**

表14 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将32个寄存器中的值全部清零  1：复位  0：无效 |
| 3 | A1 | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1 |
| 4 | A2 | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2 |
| 5 | A3 | I | 5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器 |
| 6 | WD | I | 32位写入数据 |
| 7 | WE | I | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| 8 | PC | I | 当前PC值 |
| 9 | RD1 | O | 输出A1指定的寄存器的32位数据 |
| 10 | RD2 | O | 输出A2指定的寄存器的32位数据 |

#### (3)功能定义

表15 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | Reset有效时，将32个寄存器中的值全部清零 |
| 2 | 读数据 | 读出A1, A2地址对应寄存器中所存储数据到RD1, RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中 |

### 6. EXT

#### (1) **模块接口定义**

表16 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ext.v | module ext(  input [15:0] in,  input [1:0] EXTOp,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg [31:0] out  ); |

#### (2) **接口说明**

表17 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | in | I | 16位待扩展数据 |
| 2 | EXTop | I | 扩展操作信号  00：进行无符号扩展  01：进行符号扩展  10：加载至高位 |
| 3 | out | O | 32位输出数据 |

#### (3)功能定义

表18 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 无符号扩展 | 将16位立即数Imm16无符号拓展至32位输出Imm32 |
| 2 | 符号扩展 | 将16位立即数Imm16符号拓展至32位输出Imm32 |
| 3 | 加载至高位 | 将16位立即数Imm16加载至高位后32位输出Imm32 |

### 7. EQUAL

#### (1) **模块接口定义**

表19 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| equal.v | module equal(  input [31:0] RD1,  input [31:0] RD2,  output equal  ); |

#### (2) **接口说明**

表20 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | RD1 | I | 待比较数A |
| 2 | RD2 | I | 待比较数B |
| 3 | equal | O | 比较结果 |

#### (3)功能定义

表21 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 判断是否相等 | A==B则equal=1 |

### 8. ID/EX

#### (1) **模块接口定义**

表22 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IDEX.v | module IDEX(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] Instr\_D,  input [31:0] PC\_D,  input [31:0] PC4\_D,  input [31:0] PC8\_D,  input [31:0] RD1\_D,  input [31:0] RD2\_D,  input [31:0] SignImm\_D,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] Instr\_E,  output reg[31:0] PC\_E,  output reg[31:0] PC4\_E,  output reg[31:0] PC8\_E,  output reg[31:0] RD1\_E,  output reg[31:0] RD2\_E,  output reg[31:0] SignImm\_E  ); |

#### (2)功能定义

表23 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存ID/EX的流水线信号 |

### 9. ALU

#### (1) **模块接口定义**

表24 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| alu.v | module alu(  input [31:0] A,  input [31:0] B,  input [2:0] ALUOp,  output [31:0] Result,  ); |

#### (2) **接口说明**

表25 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | A | I | 参与ALU计算的第一个32位数据 |
| 2 | B | I | 参与ALU计算的第二个32位数据 |
| 3 | ALUOp | I | ALU功能的选择信号：  000：ALU进行与运算  001：ALU进行或运算  010：ALU进行加法运算  011：ALU进行减法运算 |
| 4 | Result | O | ALU的计算结果 |

#### (3)功能定义

表26 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 与运算 |  |
| 2 | 或运算 |  |
| 3 | 加运算 |  |
| 4 | 减运算 |  |

### 10. MDU

#### (1) **模块接口定义**

表27 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| alu.v | module mdu(  input clk,  input reset,  input start,  input [31:0] D1,  input [31:0] D2,  input [5:0] MIPS,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output BUSY,  output [31:0] HI,  output [31:0] LO  ); |

#### (2) **接口说明**

表28 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | D1 | I | 参与MUL/DIV计算的第一个32位数据 |
| 2 | D2 | I | 参与MUL/DIV计算的第二个32位数据 |
| 3 | MIPS | I | 功能的选择信号 |
| 4 | HI | O | 高位计算结果 |
| 5 | LO | O | 低位计算结果 |

#### (3)功能定义

表29 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 符号乘 |  |
| 2 | 无符号乘 |  |
| 3 | 符号除 |  |
| 4 | 无符号除 |  |

### 11. EX/MEM

#### (1) **模块接口定义**

表30 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| EXMEM.v | module EXMEM(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] Instr\_E,  input [31:0] PC\_E,  input [31:0] PC4\_E,  input [31:0] PC8\_E,  input [31:0] ALUOut\_E,  input [31:0] WD\_E,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] Instr\_M,  output reg[31:0] PC\_M,  output reg[31:0] PC4\_M,  output reg[31:0] PC8\_M,  output reg[31:0] ALUOut\_M,  output reg[31:0] WD\_M  ); |

#### (2)功能定义

表31 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存EX/MEM的流水线信号 |

### 12. DM

#### (1) **模块接口定义**

表32 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| dm.v | module dm(  input clk,  input reset,  input MemWrite,  input MemRead,  input [31:0] addr,  input [31:0] WriteData,  input [31:0] PC,  output [31:0] ReadData); |

#### (2) **接口说明**

表33 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将DM中的值全部清零  1：复位  0：无效 |
| 3 | MemRead | I | 写使能信号  1：可向DM中写入数据  0：无效 |
| 4 | MemWrite | I | 读使能信号  1：可读取DM中数据  0：无效 |
| 5 | addr | I | 32位地址输入信号，对DM指定地址进行读写操作 |
| 6 | WriteData | I | 32位写入数据 |
| 7 | PC | I | 当前PC值 |
| 8 | ReadData | O | 32位输出数据 |

#### (3)功能定义

表34 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | Reset有效时，将DM中的值全部清零 |
| 2 | 读操作 | 读出addr地址对应存储数据到ReadData |
| 3 | 写操作 | 当时钟上升沿来临时，将WriteData写入addr地址对应位置 |

### 13. MEM/WB

#### (1) **模块接口定义**

表35 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MEMWB.v | module MEMWB(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] Instr\_M,  input [31:0] PC\_M,  input [31:0] PC4\_M,  input [31:0] PC8\_M,  input [31:0] ALUOut\_M,  input [31:0] RD\_M,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] Instr\_W,  output reg[31:0] PC\_W,  output reg[31:0] PC4\_W,  output reg[31:0] PC8\_W,  output reg[31:0] ALUOut\_W,  output reg[31:0] RD\_W  ); |

#### (2)功能定义

表36 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存MEM/RB的流水线信号 |

### 14. MUX

#### (1) **模块接口定义**

表37 **模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| mux.v | module mux\_2\_32(  input select,  input [31:0] mi1,  input [31:0] mi2,  output [31:0] mo  );  module mux\_4\_32(  input [1:0] select,  input [31:0] mi1,  input [31:0] mi2,  input [31:0] mi3,  input [31:0] mi4,  output [31:0] mo  );  module mux\_3\_5(  input [1:0] select,  input [4:0] mi1,  input [4:0] mi2,  output [4:0] mo  );  module forward(  input [2:0] select,  input [31:0] mi1,  input [31:0] mi2,  input [31:0] mi3,  input [31:0] mi4,  input [31:0] mi5,  input [31:0] mi6,  input [31:0] mi7,  input [31:0] mi8,  output [31:0] mo  ); |

#### (2) **接口说明**

表38 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | select | I | 选择信号 |
| 2 | mi（多个）x | I | 输入信号 |
| 3 | mo | O | 输出数据 |

#### (3)功能定义

表39 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 选择信号 | 根据select信号选择输入输出 |

## 三、控制器设计与冲突判断

## 

### 1. Control模块**端口与功能说明**

表40 Control模块端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | Op [5:0] | I | 6位控制信号 |
| 2 | Func [5:0] | I | 6位控制信号 |
| 3 | RegDst [1:0] | O | 00：将Rt作为GRF的A3写入地址  01：将Rd作为GRF的A3写入地址  10：将h1f作为GRF的A3写入地址 |
| 4 | ALUSrc | O | 0：将RD2作为参与ALU计算的第二个数据  1：将EXT输出作为参与ALU计算的第二个数据 |
| 5 | MemToReg [1:0] | O | 00：将ALU计算结果写入GRF  01：将DM输出值写入GRF  10：将填充到高位的 16位立即数写入GRF |
| 6 | RegWrite | O | 0：GRF写使能信号无效  1：GRF写使能信号有效 |
| 7 | MemRead | O | 0：DM读使能信号无效  1：DM读使能信号有效 |
| 8 | MemWrite | O | 0：DM写使能信号无效  1：DM写使能信号有效 |
| 9 | Branch [2:0] | O | 000：默认操作  001：当前指令为beq指令  010：当前指令为jar指令  011：当前指令为jr指令  101：当前指令为j指令 |
| 10 | ExtOp [1:0] | O | 00：进行无符号扩展  01：进行符号扩展  10：lui指令计算 |
| 11 | ALUOp [2:0] | O | ALU功能的选择信号：  000：ALU进行与运算  001：ALU进行或运算  010：ALU进行加法运算  011：ALU进行减法运算 |

### 2. Control模块**真值表**

表41 Control模块**真值表**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | addu | subu | ori | lw | sw | beq | lui | jal | j | jr |
| Op字段 | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 | 000010 | 000000 |
| Func字段 | 100001 | 100011 | xxxxxx | | | | | | | 001000 |
| RegDst | 01 | 01 | 00 | 00 | 00 | 00 | 00 | 10 | 00 | 01 |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| MemToReg | 00 | 00 | 00 | 01 | 00 | 00 | 10 | 11 | 00 | 00 |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| MemRead | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Branch | 000 | 000 | 000 | 000 | 000 | 001 | 000 | 010 | 101 | 011 |
| ExtOp | 00 | 00 | 01 | 00 | 00 | 00 | 10 | 00 | 00 | 00 |
| ALUOp | 010 | 011 | 001 | 010 | 010 | 011 | 111 | 111 | 111 | 010 |

### 3. 分布式Control模块

#### (1) **模块接口定义**

表42 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| mips.v | controller C\_ID (  .op(Instr\_D[`i\_op]),  .Fu(Instr\_D[`i\_func]),  .Branch(con\_o\_Branch\_3),  .EXTOp(con\_o\_EXTOp\_2)  );  controller C\_EX (  .op(Instr\_E[`i\_op]),  .Fu(Instr\_E[`i\_func]),  .ALUOp(con\_o\_ALUOp\_3),  .ALUSrcA(con\_o\_ALUSrcA),  .ALUSrcB(con\_o\_ALUSrcB)  );  controller C\_MEM (  .op(Instr\_M[`i\_op]),  .Fu(Instr\_M[`i\_func]),  .MemRead(con\_o\_MemRead),  .MemWrite(con\_o\_MemWrite)  );  controller C\_WB (  .op(Instr\_W[`i\_op]),  .Fu(Instr\_W[`i\_func]),  .RegDst(con\_o\_RegDst\_2),  .RegWrite(con\_o\_RegWrite),  .MemtoReg(con\_o\_MemtoReg\_2)  ); |

#### (2) **总模块接口定义**

表43 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| controller.v | module controller(  input [5:0] op,  input [5:0] Fu,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output [2:0] ALUOp,  output [1:0] RegDst,  output ALUSrcA,  output ALUSrcB,  output RegWrite,  output MemRead,  output MemWrite,  output [1:0] MemtoReg,  output [2:0] Branch,  output [1:0] EXTOp  ); |

### 4. 暂停与转发机制

见附表

## 四、测试程序

### 1. test branch代码

module *tb*;

*// Inputs*

    reg clk;

    reg reset;

*// Instantiate the Unit Under Test (UUT)*

    mips uut (

        .clk(clk),

        .reset(reset)

    );

    initial begin

*// Initialize Inputs*

        clk = 0;

        reset = 1;

*// Wait 100 ns for global reset to finish*

        #100;

        reset = 0;

*// Add stimulus here*

    end

    always #10 clk = ~clk;

endmodule

### 2. mars测试程序与结果

#### (1) **暂停测试**

表9 EXT端口说明

|  |  |  |
| --- | --- | --- |
| 冲突类型 | MIPS代码 | 预期结果 |
| Beq 与 cal\_r/ cal\_i/ load冲突 | .text  ori $3,$0,123  addu $4,$0,$3  beq $4,$3,next1  nop  ori $9,$0,99  next1:  addu $5,$4,$0  ori $6,$0,123  beq $6,$5,next2  nop  ori $9,$0,88  next2:  sw $6,0($0)  lw $7,0($0)  lw $8,0($0)  beq $7,$8,next3  nop  ori $9,$0,77  next3:  ori $9,$0,100  3403007b  00032021  10830002  00000000  34090063  00802821  3406007b  10c50002  00000000  34090058  ac060000  8c070000  8c080000  10e80002  00000000  3409004d  34090064 | @00003000: $ 3 <= 0000007b  @00003004: $ 4 <= 0000007b  @00003014: $ 5 <= 0000007b  @00003018: $ 6 <= 0000007b  @00003028: \*00000000 <= 0000007b  @0000302c: $ 7 <= 0000007b  @00003030: $ 8 <= 0000007b  @00003040: $ 9 <= 00000064 |
| Cal\_r/ cal\_i 与load冲突 | .text  ori $8,$0,123  sw $8,0($0)  lw $9,0($0)  addu $10,$9,$9  lw $10,0($0)  ori $11,$10,0  sw $11,1024($0)  3408007b  ac080000  8c090000  01295021  8c0a0000  354b0000  ac0b0400 | @00003000: $ 8 <= 0000007b  @00003004: \*00000000 <= 0000007b  @00003008: $ 9 <= 0000007b  @0000300c: $10 <= 000000f6  @00003010: $10 <= 0000007b  @00003014: $11 <= 0000007b  @00003018: \*00000400 <= 0000007b |
| load\_rs 与load\_rt冲突 | .text  ori $3,$0,12  sw $3,0($0)  lw $4,0($0)  sw $3,0($4)  lw $5,0($0)  lw $6,0($4)  3403000c  ac030000  8c040000  ac830000  8c050000  8c860000 | @00003000: $ 3 <= 0000000c  @00003004: \*00000000 <= 0000000c  @00003008: $ 4 <= 0000000c  @0000300c: \*0000000c <= 0000000c  @00003010: $ 5 <= 0000000c  @00003014: $ 6 <= 0000000c |
| store\_rs 与load rt冲突 | .text  ori $3,$0,16  sw $3,0($0)  lw $4,0($0)  sw $4,4($0)  lw $6,4($0)  sw $6,8($0)  lw $7,8($0)  sw $7,12($0)  lw $8,12($0)  sw $8,0($8)  lw $9,0($8)  sw $9,4($9)  lw $10,4($9)  addu $11,$10,$10  34030010  ac030000  8c040000  ac040004  8c060004  ac060008  8c070008  ac07000c  8c08000c  ad080000  8d090000  ad290004  8d2a0004  014a5821 | @00003000: $ 3 <= 00000010  @00003004: \*00000000 <= 00000010  @00003008: $ 4 <= 00000010  @0000300c: \*00000004 <= 00000010  @00003010: $ 6 <= 00000010  @00003014: \*00000008 <= 00000010  @00003018: $ 7 <= 00000010  @0000301c: \*0000000c <= 00000010  @00003020: $ 8 <= 00000010  @00003024: \*00000010 <= 00000010  @00003028: $ 9 <= 00000010  @0000302c: \*00000014 <= 00000010  @00003030: $10 <= 00000010  @00003034: $11 <= 00000020 |
| jr\_rs 与cal\_r/ cal\_i / load冲突 | .text  ori $3,$0,0x00003010 #16  jr $3 #4  nop#8  ori $4,$0,123 #12  ori $5,$0,0x00003010 #16  ori $6,$0,24#24  addu $7,$5,$6 #24  jr $7 #28  nop  ori $1,$0,123  addu $9,$7,$7  34033010  00600008  00000000  3404007b  34053010  34060018  00a63821  00e00008  00000000  3401007b  00e74821 | @00003000: $ 3 <= 00003010  @00003010: $ 5 <= 00003010  @00003014: $ 6 <= 00000018  @00003018: $ 7 <= 00003028  @00003028: $ 9 <= 00006050 |

#### (2) **转发测试**

表9 EXT端口说明

|  |  |  |
| --- | --- | --- |
| 转发类型 | MIPS代码 | 预期结果 |
| E->D | .text  lui $3, 203  lui $4, 203  beq $3,$4 next  ori $5,123  ori $6,123  next:  jal end  ori $8,$31,0  ori $9,$31,0  end:  ori $10,$31,0  3c0300cb  3c0400cb  10640002  34a5007b  34c6007b  0c000c08  37e80000  37e90000  37ea0000 | @00003000: $ 3 <= 00cb0000  @00003004: $ 4 <= 00cb0000  @0000300c: $ 5 <= 0000007b  @00003014: $31 <= 0000301c  @00003018: $ 8 <= 0000301c  @00003020: $10 <= 0000301c |
| M->D | .text  ori  $3, $0,203  addu $4,$0, 203 #要暂停  beq $3,$4 next  ori $5,123  ori $6,123  next:  jal end  ori $8,$31,0  ori $9,$31,0  end:  ori $10,$31,0  340300cb  3c010000  342100cb  00012021  10640002  34a5007b  34c6007b  0c000c0a  37e80000  37e90000  37ea0000 | @00003000: $ 3 <= 000000cb  @00003004: $ 1 <= 00000000  @00003008: $ 1 <= 000000cb  @0000300c: $ 4 <= 000000cb  @00003014: $ 5 <= 0000007b  @0000301c: $31 <= 00003024  @00003020: $ 8 <= 00003024  @00003028: $10 <= 00003024 |
| W->D | .text  ori $12, $0,321  ori $13, $0,321  ori $3, $0,4  sw $12, 0($3)  sw $13, 4($3)  lw $4, 4($3)  lw $5, 0($3)  beq $4,$5,end  ori $8,$0,123  ori $9,$0,123  end:  ori $31,$0,0  340c0141  340d0141  34030004  ac6c0000  ac6d0004  8c640004  8c650000  10850002  3408007b  3409007b  341f0000 | @00003000: $12 <= 00000141  @00003004: $13 <= 00000141  @00003008: $ 3 <= 00000004  @0000300c: \*00000004 <= 00000141  @00003010: \*00000008 <= 00000141  @00003014: $ 4 <= 00000141  @00003018: $ 5 <= 00000141  @00003020: $ 8 <= 0000007b  @00003028: $31 <= 00000000 |
| M->E/ W->E | .text  ori $3,$0,123  ori $4,$3,0  addu $5,$4,$3  addu $6,$5,$4  addu $7,$5,$6  subu $8,$7,$6  3403007b  34640000  00832821  00a43021  00a63821  00e64023 | @00003000: $ 3 <= 0000007b  @00003004: $ 4 <= 0000007b  @00003008: $ 5 <= 000000f6  @0000300c: $ 6 <= 00000171  @00003010: $ 7 <= 00000267  @00003014: $ 8 <= 000000f6 |
| W->M | .text  ori $3, $0,12  ori $4,$0,123  sw $4,0($3)  addu $5,$4,$0  sw $5,4($3)  3403000c  3404007b  ac640000  00802821  ac650004 | @00003000: $ 3 <= 0000000c  @00003004: $ 4 <= 0000007b  @00003008: \*0000000c <= 0000007b  @0000300c: $ 5 <= 0000007b  @00003010: \*00000010 <= 0000007b |

## 五、思考题

### 1)乘除模块

#### 1.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？

ALU输出为32位，乘除法输出为64位（HI、LO寄存器），如果要在ALU中运算改动过大；乘除法计算延迟大，硬件上也应该独立。

#### 2.参照你对延迟槽的理解，试解释“乘除槽”。

类似延迟槽，当D级控制器检测到当前指令为乘除法操作时，会产生start信号，并且在下一周期产生busy信号，此时间为乘除槽，如果不发生冲突而不导致暂停。延迟槽可理解为一条空指令。

### 2)扩展DM

#### 1.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语言中字符串每个字符占用一个字节。此时如果为按字节访问可以直接取出字符，而按照字访问则需要先取出字再取字符。此时按字节访问内存相对于按字访问内存性能上更有优势。

### 3)复杂性控制与设计风格

#### 1.如何概括你所设计的CPU的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？

大杂烩式。（侦测者）

在阅读完Ausar的《ball ball你们不要再集中式写控制器了》一文之后，对于控制信号我用了6位解码，降低了耦合度。

#### 2.你对流水线CPU设计风格有何见解？

我认为侦测者（DETECTOR）型更具优势，在第一次考P5的时候我使用的是**规划者（PLANNER）型，由于时间紧张没有Debug成功。**

### 4) 在线测试相关说明

#### 1.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？

assign stall\_rs = (D\_need\_rs & (D\_Tuse\_rs < Tnew\_E) & (GRF\_A3\_E != 5'h0) & (GRF\_A3\_E == Instr\_D[`i\_rs])) |

                            (D\_need\_rs & (D\_Tuse\_rs < Tnew\_M) & (GRF\_A3\_M != 5'h0) & (GRF\_A3\_M == Instr\_D[`i\_rs]));

    assign stall\_rt = (D\_need\_rt & (D\_Tuse\_rt < Tnew\_E) & (GRF\_A3\_E != 5'h0) & (GRF\_A3\_E == Instr\_D[`i\_rt])) |

                            (D\_need\_rt & (D\_Tuse\_rt < Tnew\_M) & (GRF\_A3\_M != 5'h0) & (GRF\_A3\_M == Instr\_D[`i\_rt]));

    assign stall\_hl =   (D\_need\_hl & (busy\_E|start\_E));

解决：Tnew和Tuse：

|  |  |  |  |
| --- | --- | --- | --- |
| 指令名 | D级Tuse-Rs | D级Tuse-Rt | E级Tnew |
| ADD | 1 | 1 | 1 |
| ADDI | 1 | null | 1 |
| ADDIU | 1 | null | 1 |
| ADDU | 1 | 1 | 1 |
| AND | 1 | 1 | 1 |
| ANDI | 1 | null | 1 |
| BEQ | 0 | 0 | null |
| BGEZ | 0 | null | null |
| BGTZ | 0 | null | null |
| BLEZ | 0 | null | null |
| BLTZ | 0 | null | null |
| BNE | 0 | 0 | null |
| DIV | 1 | 1 | null |
| DIVU | 1 | 1 | null |
| J | null | null | null |
| JAL | null | null | 0 |
| JALR | 0 | null | 0 |
| JR | 0 | null | null |
| LB | 1 | null | 2 |
| LBU | 1 | null | 2 |
| LH | 1 | null | 2 |
| LHU | 1 | null | 2 |
| LUI | null | null | 1 |
| LW | 1 | null | 2 |
| MFHI | null | null | 1 |
| MFLO | null | null | 1 |
| MTHI | 1 | null | null |
| MTLO | 1 | null | null |
| MULT | 1 | 1 | null |
| MULTU | 1 | 1 | null |
| NOR | 1 | 1 | 1 |
| OR | 1 | 1 | 1 |
| ORI | 1 | null | 1 |
| SB | 1 | 2 | null |
| SH | 1 | 2 | null |
| SLL | null | 1 | 1 |
| SLLV | 1 | 1 | 1 |
| SLT | 1 | 1 | 1 |
| SLTI | 1 | null | 1 |
| SLTIU | 1 | null | 1 |
| SLTU | 1 | 1 | 1 |
| SRA | null | 1 | 1 |
| SRAV | 1 | 1 | 1 |
| SRL | null | 1 | 1 |
| SRLV | 1 | 1 | 1 |
| SUB | 1 | 1 | 1 |
| SUBU | 1 | 1 | 1 |
| SW | 1 | 2 | null |
| XOR | 1 | 1 | 1 |
| XORI | 1 | null | 1 |

测试样例:使用hgg的随机生成代码

示例：

.data

.space 64

arr1: .space 64

arr2: .space 64

.space 64

.text

N0: srl $t2, $t2, 28

addi $t2, $t2, 64

andi $t2, $t2, 0xfffffffe

sh $t2, 10($t2)

N1: sra $t0, $t0, 28

lbu $t3, 76($t0)

N2: xor $t1, $t2, $t3

N3: srl $t2, $t2, 28

addi $t2, $t2, 64

andi $t2, $t2, 0xfffffffc

sw $t3, 4($t2)

N4: nor $zero, $t2, $t2

N5: ori $t1, $t0, 37763

N6: srl $t1, $t1, 28

addi $t1, $t1, 64

sb $t3, 0($t1)

N7: ori $t0, $t0, 1

div $t2, $t0

N8: addiu $zero, $t0, 12364

N9: sra $t2, $t2, 28

lbu $t1, 72($t2)

N10: srl $t1, $t1, 28

addi $t1, $t1, 64

lb $t3, 8($t1)

N11: sra $zero, $t3, 0

N12: sra $t3, $t3, 28

sb $t1, 73($t3)

N13: sllv $zero, $t3, $t3

N14: srl $t0, $t0, 28

addi $t0, $t0, 64

andi $t0, $t0, 0xfffffffe

lhu $zero, 4($t0)

N15: subu $t2, $t2, $t0

N16: slt $t3, $zero, $t0

N17: nop

N18: srl $t0, $t0, 28

addi $t0, $t0, 64

andi $t0, $t0, 0xfffffffc

sw $t0, 0($t0)

N19: mthi $t2

N20: srl $t1, $t3, 2

N21: addiu $t3, $zero, 37229

N22: addiu $t2, $t3, 3747

N23: mtlo $t2

N24: or $t2, $zero, $zero

N25: slt $zero, $t3, $zero

N26: or $t0, $t1, $zero

N27: slt $t0, $t1, $t0

N28: sra $t2, $t2, 28

lb $t0, 68($t2)

N29: slt $t0, $zero, $zero

N30: sltu $t0, $t3, $zero

N31: srl $t0, $t1, 15

N32: srl $t2, $t2, 28

addi $t2, $t2, 64

andi $t2, $t2, 0xfffffffc

sw $zero, 12($t2)

N33: sra $t2, $t2, 28

andi $t2, $t2, 0xfffffffc

lw $t3, 72($t2)

N34: sra $t0, $t0, 28

andi $t0, $t0, 0xfffffffc

lw $zero, 64($t0)

N35: mult $t3, $t1

N36: sra $t0, $t0, 28

andi $t0, $t0, 0xfffffffe

lhu $zero, 74($t0)

N37: srl $t3, $t3, 28

addi $t3, $t3, 64

andi $t3, $t3, 0xfffffffe

lhu $t1, 4($t3)

N38: ori $zero, $t1, 8431

N39: lui $t0, 8265

N40: xor $t3, $t3, $zero

N41: and $t0, $t2, $t2

N42: slti $t0, $t2, -31436

N43: srav $t0, $t2, $zero

N44: addiu $t1, $t2, 65117

N45: slt $t2, $zero, $t3

N46: srav $t0, $t3, $zero

N47: srl $t3, $t3, 28

addi $t3, $t3, 64

andi $t3, $t3, 0xfffffffe

lhu $t0, 10($t3)

N48: srl $t2, $t2, 28

addi $t2, $t2, 64

sb $t3, 6($t2)

N49: or $t3, $t3, $t1

N50: sra $zero, $zero, 28

andi $zero, $zero, 0xfffffffe

sh $t2, 76($zero)

N51: srl $t2, $t2, 28

addi $t2, $t2, 64

andi $t2, $t2, 0xfffffffc

sw $t0, 8($t2)

N52: lui $t1, 48528

N53: srl $zero, $zero, 28

addi $zero, $zero, 64

andi $zero, $zero, 0xfffffffe

lh $zero, 14($zero)

N54: sra $t0, $t0, 28

andi $t0, $t0, 0xfffffffe

sh $t0, 64($t0)

N55: sra $zero, $zero, 28

andi $zero, $zero, 0xfffffffc

lw $t3, 72($zero)

N56: sra $t3, $t3, 28

andi $t3, $t3, 0xfffffffc

sw $t3, 72($t3)

N57: srl $t0, $t0, 28

addi $t0, $t0, 64

lbu $t0, 8($t0)

N58: nop

N59: nop

N60: sra $t2, $t2, 28

lbu $t1, 64($t2)

N61: nor $t3, $t2, $t0

N62: ori $t2, $t2, 1

div $t1, $t2

N63: and $t2, $zero, $t1

N64: slt $t2, $t0, $t3

N65: lui $zero, 44879

N66: sra $t3, $t3, 28

sb $t0, 76($t3)

N67: addiu $t2, $t3, 31936

N68: sra $t0, $t0, 31

N69: sltu $zero, $t0, $zero

N70: sra $t0, $t0, 28

andi $t0, $t0, 0xfffffffc

lw $t1, 76($t0)

N71: srl $t1, $t1, 28

addi $t1, $t1, 64

lbu $t1, 7($t1)

N72: mfhi $zero

N73: addu $t2, $t2, $t1

N74: slti $t0, $t2, 16742

N75: andi $t0, $zero, 10814

N76: ori $t1, $t1, 1

divu $zero, $t1

N77: mfhi $t1

N78: nor $t2, $t3, $t2

N79: srav $t0, $t0, $zero

N80: ori $t2, $t2, 1

div $t1, $t2

N81: ori $t3, $zero, 41382

N82: sll $t1, $t0, 28

N83: mthi $t0

N84: or $t2, $t2, $t2

N85: mfhi $zero

N86: ori $t3, $t3, 1

divu $t2, $t3

N87: sra $t0, $t0, 28

lb $zero, 73($t0)

N88: ori $t2, $t2, 1

divu $t2, $t2

N89: sltu $t3, $t2, $zero

N90: sra $t2, $t2, 1

N91: sra $zero, $zero, 28

andi $zero, $zero, 0xfffffffc

lw $t2, 76($zero)

N92: slt $t0, $t3, $t1

N93: srl $t3, $t3, 28

addi $t3, $t3, 64

andi $t3, $t3, 0xfffffffe

sh $t0, 8($t3)

N94: sra $t2, $t2, 28

andi $t2, $t2, 0xfffffffe

sh $t1, 76($t2)

N95: sra $t1, $t1, 28

sb $t3, 67($t1)

N96: slt $t0, $t2, $zero

N97: srl $t3, $t3, 28

addi $t3, $t3, 64

lbu $t0, 1($t3)

N98: mflo $t0

N99: sra $t2, $t2, 28

lbu $zero, 65($t2)

N100: nop

N101: sra $t0, $t0, 28

lbu $zero, 76($t0)

N102: srl $t1, $t1, 28

addi $t1, $t1, 64

lb $t0, 4($t1)

N103: mtlo $zero

N104: slti $t2, $t2, 27571

N105: lui $t2, 31619

N106: ori $t2, $t2, 3236

N107: sra $t3, $t3, 28

andi $t3, $t3, 0xfffffffe

sh $t0, 74($t3)

N108: mflo $t1

N109: sltiu $zero, $t3, -28826

N110: sra $t2, $t2, 28

andi $t2, $t2, 0xfffffffc

sw $t1, 72($t2)

N111: slti $t2, $t1, -9649

N112: andi $t3, $t1, 32390

N113: ori $t2, $t2, 1

divu $t2, $t2

N114: and $t0, $t1, $t3

N115: mtlo $zero

N116: sra $zero, $zero, 28

andi $zero, $zero, 0xfffffffe

lh $t0, 64($zero)

N117: srl $t0, $t0, 28

addi $t0, $t0, 64

lbu $t3, 6($t0)

N118: sllv $t2, $t2, $zero

N119: and $zero, $t2, $t3

N120: srl $t2, $t2, 28

addi $t2, $t2, 64

sb $t0, 5($t2)

N121: slt $zero, $t0, $zero

N122: addu $t3, $t2, $t3

N123: ori $t1, $t1, 1

div $t1, $t1

N124: slti $t2, $t2, 19042

N125: ori $t1, $t1, 1

div $t2, $t1

N126: addiu $t2, $t1, 20224

N127: slt $t3, $t0, $t3

N128: mflo $t3

N129: ori $t2, $t2, 1

div $zero, $t2

N130: multu $t3, $t2

N131: xori $t1, $t0, 51745

N132: srlv $zero, $t3, $zero

N133: sra $zero, $zero, 28

andi $zero, $zero, 0xfffffffc

sw $t0, 72($zero)

N134: ori $t1, $t1, 1

divu $zero, $t1

N135: mflo $zero

N136: or $zero, $zero, $t3

N137: xor $zero, $t1, $zero

N138: sllv $zero, $t0, $t0

N139: ori $t3, $t2, 39088

N140: srl $zero, $zero, 28

addi $zero, $zero, 64

andi $zero, $zero, 0xfffffffc

lw $t0, 12($zero)

N141: sllv $t0, $t1, $t0

N142: sra $t1, $t1, 28

lbu $zero, 64($t1)

N143: addiu $t3, $t1, 10438

N144: sllv $zero, $t0, $t0

N145: ori $t3, $t3, 1

divu $t3, $t3

N146: srav $t1, $t1, $t0

N147: srlv $t0, $zero, $t1

N148: srlv $t0, $t3, $zero

N149: srl $t1, $t1, 28

addi $t1, $t1, 64

sb $t1, 3($t1)

N150: sra $t1, $t0, 12

N151: srav $t3, $t1, $t1

N152: mult $t3, $t1

N153: ori $t1, $t1, 56805

N154: ori $t0, $t0, 1

divu $t0, $t0

N155: andi $t3, $t1, 43985

N156: srl $t0, $t0, 28

addi $t0, $t0, 64

andi $t0, $t0, 0xfffffffc

sw $t0, 0($t0)

N157: xor $zero, $t0, $t3

N158: or $zero, $t3, $t1

N159: sra $zero, $zero, 28

lb $t1, 69($zero)

N160: or $t0, $t2, $t3

N161: srl $t2, $zero, 18

N162: srl $zero, $zero, 28

addi $zero, $zero, 64

andi $zero, $zero, 0xfffffffe

lhu $t2, 8($zero)

N163: srl $zero, $zero, 28

addi $zero, $zero, 64

andi $zero, $zero, 0xfffffffe

lhu $t2, 6($zero)

N164: sra $t3, $zero, 6

N165: nor $t0, $t2, $t1

N166: sll $t2, $t2, 15

N167: srl $t2, $t2, 28

addi $t2, $t2, 64

sb $t0, 4($t2)

N168: xori $t0, $t3, 43781

N169: addiu $t2, $zero, 52292

N170: sltiu $t1, $t3, -20723

N171: ori $t2, $t2, 1

div $t3, $t2

N172: nor $t3, $t2, $zero

N173: xori $zero, $zero, 55973

N174: srl $t3, $t3, 28

addi $t3, $t3, 64

andi $t3, $t3, 0xfffffffc

sw $t1, 12($t3)

N175: srl $zero, $zero, 28

addi $zero, $zero, 64

andi $zero, $zero, 0xfffffffe

lhu $t1, 2($zero)

N176: xor $t0, $t2, $t3

N177: sll $zero, $t0, 31

N178: slt $t0, $zero, $t2

N179: srl $t0, $t2, 22

N180: or $t0, $t3, $t1

N181: and $t1, $t2, $zero

N182: srl $t2, $t2, 28

addi $t2, $t2, 64

andi $t2, $t2, 0xfffffffe

lhu $t3, 2($t2)

N183: slt $t1, $t1, $t2

N184: mfhi $t0

N185: srl $t2, $t2, 28

addi $t2, $t2, 64

sb $t0, 13($t2)

N186: sllv $t1, $zero, $zero

N187: srav $t1, $t3, $zero

N188: nop

N189: and $t2, $t3, $t1

N190: srl $t2, $t2, 28

addi $t2, $t2, 64

andi $t2, $t2, 0xfffffffc

lw $t3, 8($t2)

N191: sra $t1, $t1, 28

sb $zero, 75($t1)

N192: andi $t0, $zero, 31199

N193: sra $t2, $t3, 26

N194: srl $t3, $t3, 28

addi $t3, $t3, 64

andi $t3, $t3, 0xfffffffe

lh $zero, 10($t3)

N195: srl $t1, $t3, 25

N196: srl $zero, $zero, 28

addi $zero, $zero, 64

andi $zero, $zero, 0xfffffffc

lw $t1, 0($zero)

N197: nop

N198: ori $t0, $t2, 33370

N199: mfhi $t0

N200: nop

EXIT:

beq $zero, $zero, EXIT

nop