# 计算机组成原理P4实验报告

## 一、CPU设计方案综述

### （一）总体设计概述

本CPU为Verilog实现的单周期MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、jal、jr、nop}。为了实现这些功能，CPU主要包含了PC、NPC、IM、ALU、DM、EXT、GRF、CONTROL等模块，其中前七个模块并列置于DATAPATH模块下，DATAPATH与CONTROL模块并列置于顶层模块下。

### （二）关键模块定义

#### 1. PC

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位至0x00003000） |
| DI[31:0] | I | 32位输入次指令地址 |
| DO[31:0] | O | 32位输出当前指令地址 |

#### NPC

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC[31:0] | I | 32位输入当前指令地址 |
| Imm[25:0] | I | 26位输入用于跳转的立即数 |
| NPCOp[1:0] | I | 2位输入用于选择NPC |
| Zero | I | 1位输入辅助选择NPC |
| RA | I | 32位输入来自寄存器的指令地址 |
| NPC[31:0] | O | 32位输出次指令地址 |
| PC4[31:0] | O | 32位输出返回值 |

#### IM

|  |  |  |
| --- | --- | --- |
| 功能描述 | 指令存储器，内含32\*1024字存储器，根据输入的地址输出指令 | |
| 信号名 | 方向 | 描述 |
| ImAddr[31:0] | I | 32位输入当前指令地址 |
| ImData[31:0] | O | 32位输出指令信号 |

1. ALU

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能 |
| A[32:0] | I | 第一个32位操作数 |
| B[32:0] | I | 第二个32位操作数 |
| ALUOp[3:0] | I | 4位输入ALU功能选择 |
| C[31:0] | O | 32位输出计算结果 |
| Zero | O | 1位输出判断结果 |

1. DM

|  |  |  |
| --- | --- | --- |
| 功能描述 | 数据存储器，内含32\*1024字存储器 | |
| 信号名 | 方向 | 描述 |
| MemAddr[4:0] | I | 5位输入写入内存的地址 |
| MemData[31:0] | I | 32位输入写入内存的数据 |
| WE | I | 写入使能 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位） |
| PC[31:0] | I | 32位输入当前指令地址 |
| RD[31:0] | O | 32位读出数据 |

1. EXT

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能 |
| Imm[25:0] | I | 26位输入立即数 |
| EXTOp[1:0] | I | 2位输入控制EXT功能选择 |
| Ext[31:0] | O | 32位输出扩展结果 |

1. GRF

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能 |
| RegA1[4:0] | I | 第一个读出寄存器编号 |
| RegA2[4:0] | I | 第二个读出寄存器编号 |
| RegA3[4:0] | I | 回写寄存器的编号 |
| RegData[31:0] | I | 回写寄存器的值 |
| WE | I | 写入使能 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| PC[31:0] | I | 32位输入当前指令地址 |
| RD1[31:0] | O | 第一个寄存器编号读出的值 |
| RD2[31:0] | O | 第二个寄存器编号读出的值 |

1. CONTROL

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能 |
| Func[5:0] | I | 6位输入 |
| OpCode[5:0] | I | 6位输入 |
| NPCOp[1:0] | O | 2位输出控制NPC选择 |
| RegWrite | O | 1位输出控制GRF写使能 |
| EXTOp[1:0] | O | 2位输出控制EXT功能选择 |
| ALUOp[3:0] | O | 4位输出控制ALU功能选择 |
| MemWrite | O | 1位输出控制DM写使能 |
| RegA3Sel[1:0] | O | 2位输出用于GRF写入寄存器的选择 |
| RegDataSel[1:0] | O | 2为输出用于GRF写入数据的选择 |
| AluBSel[1:0] | O | 2位输出控制ALU第二个数据的选择 |

### 数据通路的综合

1. 所有指令的指令级别数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 部件 | PC | NPC | | | IM | RF | | | | EXT | ALU | | DM | |
| 输入信号 | DI | PC | Imm | RS | PC | RegA1 | RegA2 | RegA3 | RegData | Imm | A | B | MemData | MemAddr |
| addu | NPC.NPC | PC.DO |  |  | PC.DO | IM.D[25:21] | IM.D[20:16] | IM.D[15:11] | ALU.C |  | RF.RD1 | RF.RD2 |  |  |
| subu | NPC.NPC | PC.DO |  |  | PC.DO | IM.D[25:21] | IM.D[20:16] | IM.D[15:11] | ALU.C |  | RF.RD1 | RF.RD2 |  |  |
| ori | NPC.NPC | PC.DO |  |  | PC.DO | IM.D[25:21] |  | IM.D[20:16] | ALU.C | IM.D[25:0] | RF.RD1 | EXT.Ext |  |  |
| lw | NPC.NPC | PC.DO |  |  | PC.DO | IM.D[25:21] |  | IM.D[20:16] | DM.RD | IM.D[25:0] | RF.RD1 | EXT.Ext |  | ALU.C |
| sw | NPC.NPC | PC.DO |  |  | PC.DO | IM.D[25:21] | IM.D[20:16] |  |  | IM.D[25:0] | RF.RD1 | EXT.Ext | RF.RD2 | ALU.C |
| beq | NPC.NPC | PC.DO | IM.D[25:0] |  | PC.DO | IM.D[25:21] | IM.D[20:16] |  |  |  | RF.RD1 | RF.RD2 |  |  |
| lui | NPC.NPC | PC.DO |  |  | PC.DO |  |  | IM.D[20:16] | EXT.Ext | IM.D[25:0] |  |  |  |  |
| jal | NPC.NPC | PC.DO | IM.D[25:0] |  | PC.DO |  |  | 0x1f | NPC.PC4 |  |  |  |  |  |
| jr | NPC.NPC | PC.DO |  | RF.RD1 | PC.DO | IM.D[25:21] |  |  |  |  |  |  |  |  |

1. 控制信号真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | NPCOp | RegWrite | EXTOp | ALUOp | MemWrite | RegA3Sel | RegDataSel | AluBSel |
| addu | 00 | 1 |  | 0000 |  | 00 | 00 | 00 |
| subu | 00 | 1 |  | 0001 |  | 00 | 00 | 00 |
| ori | 00 | 1 | 00 | 0010 |  | 01 | 00 | 01 |
| lw | 00 | 1 | 01 | 0000 |  | 01 | 01 | 01 |
| sw | 00 |  | 01 | 0000 | 1 |  |  | 01 |
| beq | 01 |  |  |  |  |  |  | 00 |
| lui | 00 | 1 | 10 |  |  | 01 | 10 |  |
| jal | 10 | 1 |  |  |  | 10 | 11 |  |
| jr | 11 |  |  |  |  |  |  |  |

1. 控制信号状态描述

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NPCOp | | EXTOp | | ALUOp | | RegA3Sel | | RegDataSel | | AluBSel | |
| 控制信号 | 状态描述 | 控制信号 | 状态描述 | 控制信号 | 状态描述 | 控制信号 | 状态描述 | 控制信号 | 状态描述 | 控制信号 | 状态描述 |
| 00 | PC+4 | 00 | 0扩展 | 0000 | 加法运算 | 00 | 存入寄存器为IM.D[15:11] | 00 | 存入寄存器数据为ALU.C | 00 | ALU的第二个操作数为RF.RD2 |
| 01 | beq次指令地址计算 | 01 | 符号扩展 | 0001 | 减法运算 | 01 | 存入寄存器为IM.D[20:16] | 01 | 存入寄存器数据为DM.RD | 01 | ALU的第二个操作数为EXT.Ext |
| 10 | jal次指令地址计算 | 10 | 立即数加载至高位 | 0010 | 按位或 | 10 | 存入31号寄存器 | 10 | 存入寄存器数据为Ext |  |  |
| 11 | 来自寄存器的次地址 |  |  |  |  |  |  | 11 | 存入寄存器数据为PC4 |  |  |

### （四）重要机制实现方法

#### 1. 跳转

NPC模块和ALU模块协同工作支持指令x的跳转机制。

## 二、测试方案

### （一）典型测试样例

#### 1. addu、subu指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $1, $0, 0x1234  ori $2, $0, 0x5678  ori $3, $0, 21  ori $4, $0, 0xffff  ori $31, $0, 5  ori $30, $0, 1  lui $5, 0xaffc  lui $6, 0xaaac  addu $7, $5, $6  addu $8, $5, $1  addu $9, $6, $2  addu $10, $7, $3  addu $11, $3, $4  subu $12, $7, $3  subu $13, $2, $3  subu $14, $3, $31  subu $15, $1, $31  addu $16, $15, $31 | 34011234  34025678  34030015  3404ffff  341f0005  341e0001  3c05affc  3c06aaac  00a63821  00a14021  00c24821  00e35021  00645821  00e36023  00436823  007f7023  003f7823  01ff8021 |

Verilog运行结果：

@00003000: $ 1 <= 00001234

@00003004: $ 2 <= 00005678

@00003008: $ 3 <= 00000015

@0000300c: $ 4 <= 0000ffff

@00003010: $31 <= 00000005

@00003014: $30 <= 00000001

@00003018: $ 5 <= affc0000

@0000301c: $ 6 <= aaac0000

@00003020: $ 7 <= 5aa80000

@00003024: $ 8 <= affc1234

@00003028: $ 9 <= aaac5678

@0000302c: $10 <= 5aa80015

@00003030: $11 <= 00010014

@00003034: $12 <= 5aa7ffeb

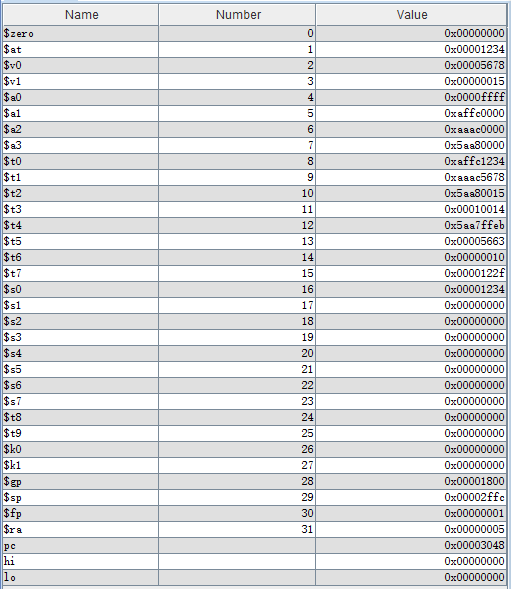
@00003038: $13 <= 00005663

@0000303c: $14 <= 00000010

@00003040: $15 <= 0000122f

@00003044: $16 <= 00001234

Mips运行结果：



#### ori、lui指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $17, $0, 0  ori $18, $0, 256  ori $19, $18, 768  ori $17, $19, 0xffff  ori $28, $0, 1  ori $20, $28, 5  ori $31, $28, 6  ori $14, $20, 512  ori $28, $14, 0xaffc  ori $15, $30, 0xcbac  lui $6, 1  lui $7, 0xffff  lui $8, 0xaffc  lui $9, 256  lui $10, 0x1abc  lui $11, 0xbbca | 34110000  34120100  36530300  3671ffff  341c0001  37940005  379f0006  368e0200  35dcaffc  37cfcbac  3c060001  3c07ffff  3c08affc  3c090100  3c0a1abc  3c0bbbca |

Verilog运行结果：

@00003000: $17 <= 00000000

@00003004: $18 <= 00000100

@00003008: $19 <= 00000300

@0000300c: $17 <= 0000ffff

@00003010: $28 <= 00000001

@00003014: $20 <= 00000005

@00003018: $31 <= 00000007

@0000301c: $14 <= 00000205

@00003020: $28 <= 0000affd

@00003024: $15 <= 0000cbac

@00003028: $ 6 <= 00010000

@0000302c: $ 7 <= ffff0000

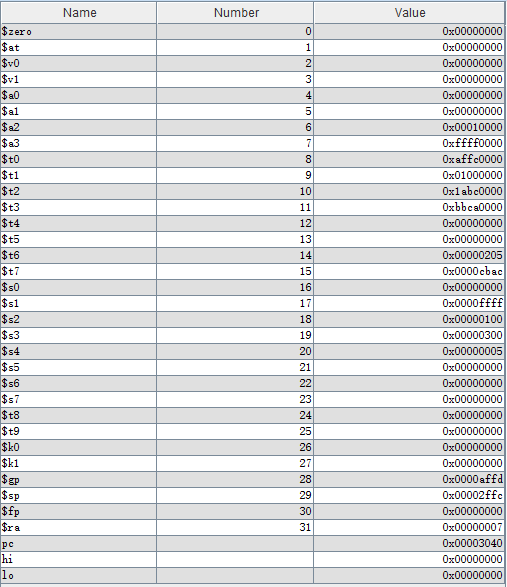
@00003030: $ 8 <= affc0000

@00003034: $ 9 <= 01000000

@00003038: $10 <= 1abc0000

@0000303c: $11 <= bbca0000

MIPS运行结果：



#### lw、sw指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $1, $0, 12  ori $2, $0, 16  ori $3, $0, 24  ori $4, $0, 0x0100  ori $5, $0, 0x0104  ori $6, $0, 0x0108  ori $17, $0, 0x1234  ori $18, $0, 0x5678  ori $19, $0, 0xbbbb  ori $20, $0, 0xaffc  lui $21, 0x4321  lui $22, 0x9876  lui $23, 0xcbbc  lui $24, 0xddab  sw $17, 0($1)  sw $18, 4($3)  sw $19, 0($2)  sw $20, 8($3)  sw $21, 0($1)  sw $22, 12($3)  sw $23, 0($2)  sw $24, 16($3)  lw $9, 0($1)  lw $10, 0($2)  lw $11, 0($3)  lw $12, 4($1)  lw $13, 8($3)  lw $14, 12($3)  lw $15, 4($2)  lw $16, 16($3)  sw $17, 0($4)  sw $18, 4($6)  sw $19, 0($5)  sw $20, 8($6)  sw $21, 0($4)  sw $22, 12($6)  sw $23, 0($5)  sw $24, 16($6)  lw $9, 0($4)  lw $10, 0($5)  lw $11, 0($6)  lw $12, 4($4)  lw $13, 8($6)  lw $14, 12($6)  lw $15, 4($5)  lw $16, 16($6) | 3401000c  34020010  34030018  34040100  34050104  34060108  34111234  34125678  3413bbbb  3414affc  3c154321  3c169876  3c17cbbc  3c18ddab  ac310000  ac720004  ac530000  ac740008  ac350000  ac76000c  ac570000  ac780010  8c290000  8c4a0000  8c6b0000  8c2c0004  8c6d0008  8c6e000c  8c4f0004  8c700010  ac910000  acd20004  acb30000  acd40008  ac950000  acd6000c  acb70000  acd80010  8c890000  8caa0000  8ccb0000  8c8c0004  8ccd0008  8cce000c  8caf0004  8cd00010 |

Verilog运行结果：

@00003000: $ 1 <= 0000000c

@00003004: $ 2 <= 00000010

@00003008: $ 3 <= 00000018

@0000300c: $ 4 <= 00000100

@00003010: $ 5 <= 00000104

@00003014: $ 6 <= 00000108

@00003018: $17 <= 00001234

@0000301c: $18 <= 00005678

@00003020: $19 <= 0000bbbb

@00003024: $20 <= 0000affc

@00003028: $21 <= 43210000

@0000302c: $22 <= 98760000

@00003030: $23 <= cbbc0000

@00003034: $24 <= ddab0000

@00003038: \*0000000c <= 00001234

@0000303c: \*0000001c <= 00005678

@00003040: \*00000010 <= 0000bbbb

@00003044: \*00000020 <= 0000affc

@00003048: \*0000000c <= 43210000

@0000304c: \*00000024 <= 98760000

@00003050: \*00000010 <= cbbc0000

@00003054: \*00000028 <= ddab0000

@00003058: $ 9 <= 43210000

@0000305c: $10 <= cbbc0000

@00003060: $11 <= 00000000

@00003064: $12 <= cbbc0000

@00003068: $13 <= 0000affc

@0000306c: $14 <= 98760000

@00003070: $15 <= 00000000

@00003074: $16 <= ddab0000

@00003078: \*00000100 <= 00001234

@0000307c: \*0000010c <= 00005678

@00003080: \*00000104 <= 0000bbbb

@00003084: \*00000110 <= 0000affc

@00003088: \*00000100 <= 43210000

@0000308c: \*00000114 <= 98760000

@00003090: \*00000104 <= cbbc0000

@00003094: \*00000118 <= ddab0000

@00003098: $ 9 <= 43210000

@0000309c: $10 <= cbbc0000

@000030a0: $11 <= 00000000

@000030a4: $12 <= cbbc0000

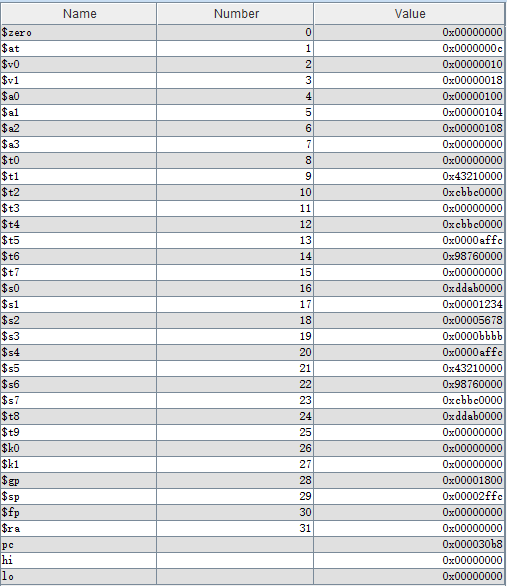
@000030a8: $13 <= 0000affc

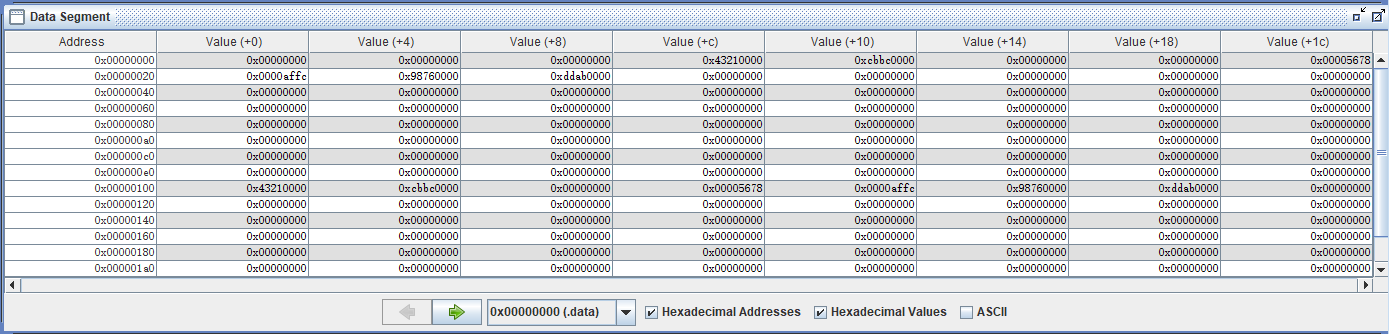
@000030ac: $14 <= 98760000

@000030b0: $15 <= 00000000

@000030b4: $16 <= ddab0000

MIPS运行结果：





1. beq、nop指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $1, $0, 0x1111  ori $2, $0, 0xaffc  ori $31, $0, 4  lui $3, 0x1234  lui $4, 0xaffc  addu $8, $1, $2  nop  addu $10, $2, $1  beq $3, $4, label1  subu $20, $8, $1  sw $20, 0($31)  nop  beq $8, $10, label2  label1:  addu $11, $1, $3  label2:  lw $30, 4($0)  ori $25, $0, 0xaffc  beq $25, $2, label3  nop  addu $24, $25, $31  label3:  lui $23, 0x1234  beq $23, $2, label4  nop  lui $16, 0x1123  label4:  nop  lui $27, 0x1000 | 34011111  3402affc  341f0004  3c031234  3c04affc  00224021  00000000  00415021  10640004  0101a023  aff40000  00000000  110a0001  00235821  8c1e0004  3419affc  13220002  00000000  033fc021  3c171234  12e20002  00000000  3c101123  00000000  3c1b1000 |

Verilog运行结果：

@00003000: $ 1 <= 00001111

@00003004: $ 2 <= 0000affc

@00003008: $31 <= 00000004

@0000300c: $ 3 <= 12340000

@00003010: $ 4 <= affc0000

@00003014: $ 8 <= 0000c10d

@0000301c: $10 <= 0000c10d

@00003024: $20 <= 0000affc

@00003028: \*00000004 <= 0000affc

@00003038: $30 <= 0000affc

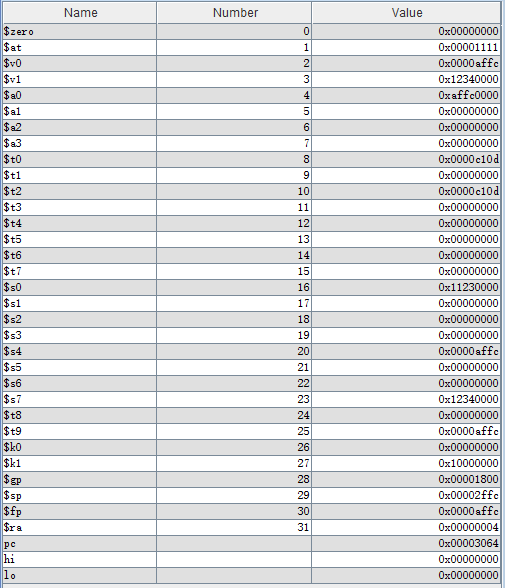
@0000303c: $25 <= 0000affc

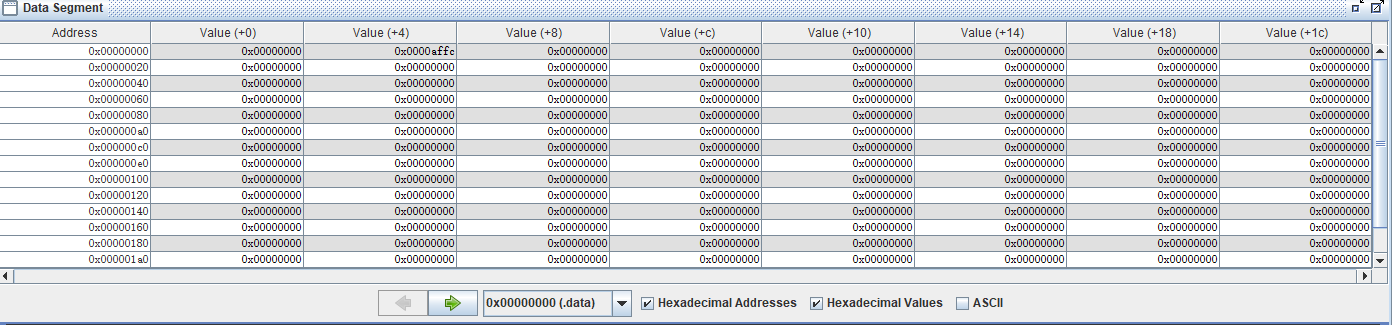
@0000304c: $23 <= 12340000

@00003058: $16 <= 11230000

@00003060: $27 <= 10000000

MIPS运行结果：





1. jal、jr指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $1, $0, 0x3048  ori $2, $0, 256  ori $3, $0, 0xffff  ori $4, $0, 1  addu $8, $2, $4  addu $10, $8, $8  subu $11, $10, $2  addu $12, $8, $4  jal label  beq $11, $12, label2  ori $21, $0, 145  label2:  lui $25, 0x1853  jr $1  label:  ori $15, $0, 123  lui $16, 0x1234  jr $ra  ori $17, 0x5678  addu $19, $16, $17  subu $26, $25, $4  subu $27, $2, $4  addu $28, $4, $4 | 34013048  34020100  3403ffff  34040001  00444021  01085021  01425823  01046021  0c000c0d  116c0001  34150091  3c191853  00200008  340f007b  3c101234  03e00008  36315678  02119821  0324d023  0044d823  0084e021 |

Verilog运行结果：

@00003000: $ 1 <= 00003048

@00003004: $ 2 <= 00000100

@00003008: $ 3 <= 0000ffff

@0000300c: $ 4 <= 00000001

@00003010: $ 8 <= 00000101

@00003014: $10 <= 00000202

@00003018: $11 <= 00000102

@0000301c: $12 <= 00000102

@00003020: $31 <= 00003024

@00003034: $15 <= 0000007b

@00003038: $16 <= 12340000

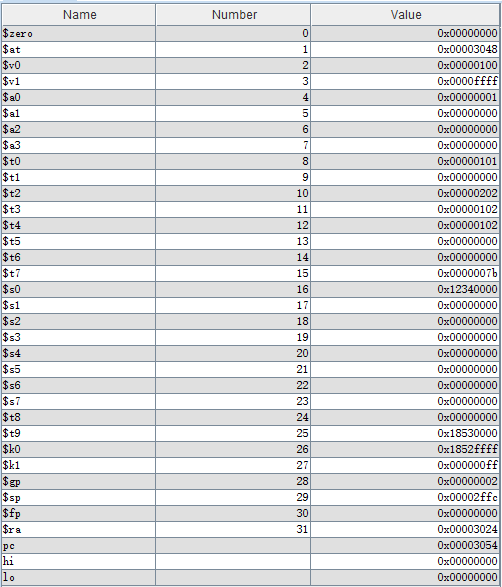
@0000302c: $25 <= 18530000

@00003048: $26 <= 1852ffff

@0000304c: $27 <= 000000ff

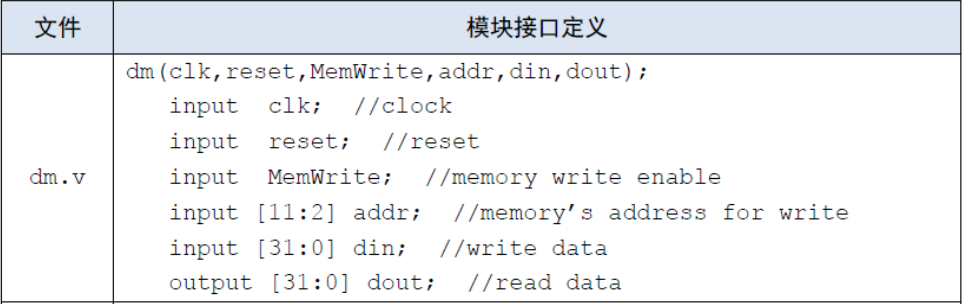
@00003050: $28 <= 00000002

MIPS运行结果：



## 三、思考题

**（一）根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？**

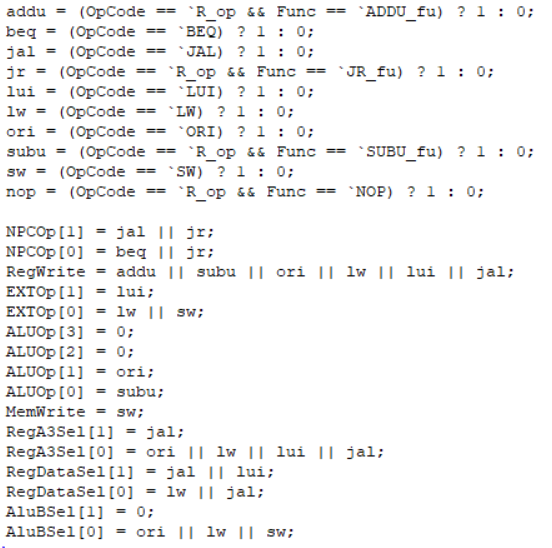
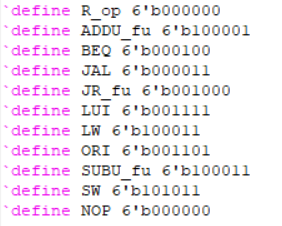


答：在指令中，地址的偏移单位是4，即每个单位存储一个字节，而dm的偏移单位是1，即每个单位存储一个字，因此指令地址偏移是dm中地址的四倍，与此同时，在二进制表示中，四倍相当于左移两位，即dm中地址选取[11:2]位。

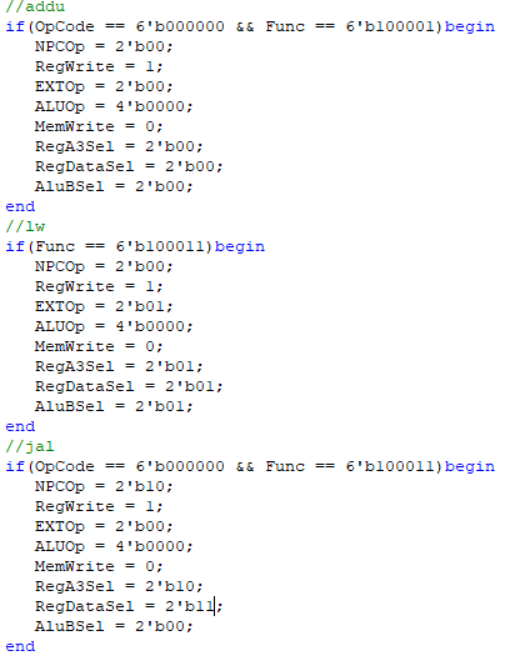
**（二）思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。**

答：

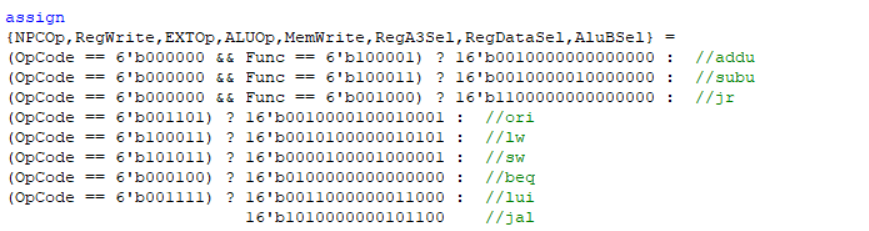
（1） 宏定义、三位运算符、或运算相结合（本CPU设计所用方法）



（2）if-else句式



（3）assign语句（跟学长学的，想不到想不到）



比较：第一种方式便于添加指令但相对混乱（或许可以通过增添空格等方式手动对齐），第二种方式逻辑很清晰但代码过长可读性差，第三种方式代码相对简洁但可读性更差（16位的编码debug直接眼瞎），总的来说宏定义能节省大量时间、不易出错且便于修改在三中译码方式中都建议使用，相比之下我更偏好第一种方式，一定程度上可能也是因为其与logisim的逻辑是一脉相承的。

**（三）题目：在相应的部件中，**reset的优先级**比其他控制信号（不包括clk信号）都要**高**，且相应的设计都是**同步复位**。清零信号reset所驱动的部件具有什么共同特点？**

答：reset驱动的部件本质上都是记忆性部件（logisim里对应的Memory Library），且都支持写入和读取，reset信号为1时，在时钟上升沿部件内所有数据返回到初始值（对于PC来说是0x00003000，对于其余部件来说是0）。

（四**）题目：C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。**

答：从相关指令的具体操作来看，add和addi判断溢出的具体方法是在32位数前再接一位符号位并将其与加法执行后的第31位比较（从0位计），若不溢出，计算结果仍取0-31位。我的理解是判断溢出的功能只是增添了一位用于判断的数，而并没有对原数据计算结果的0-31造成影响，因此在不考虑溢出即程序员可以保证程序不会有溢出时add和addi均可正常产生结果且结果分别与addu和addiu相同。即可以认为在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。

**（五）题目：根据自己的设计说明单周期处理器的优缺点。**

答：优点：设计简单，各模块内聚程度高，模块间耦合程度小。大部分模块内部是简单的组合逻辑，只需要建立好逻辑清晰的数据通路，进而完成各模块内部的逻辑即可保证较低的错误率，增添指令时只需考虑各模块内部真值表的补充，实现简单。缺点：时钟频率受各部分组合逻辑的制约，为保证在一个时钟周期内完成任一条指令的执行，需设置时钟周期大于等于执行时间最长的一条指令的延迟时间（本设计中该指令是lw），这将导致在执行其他指令时大部分时间被浪费，CPU速度较慢，性能不高。

四、debug记录

（一）display放置问题

最开始我懒于在GRF和DM模块中添加PC端口，只是将display代码放置在了DATAPATH模块，这导致在reset高位时本不应有写入功能（但此时可能出现写使能为1的情况），自然不应有display的执行，但在我的设计中display的执行与否只依赖于写使能的判断，这导致在reset为1的时候出现了一些多余的输出。

之后我将display功能分别放入了GRF与DM模块内，在模块内部reset具有最高优先级，display是在reset为0与写使能为1的双重前提下执行的，因此问题得到解决。

（二）控制信号设置问题

在解决了display的问题后，我又遇到了三个bug，一个是因为设计文档中控制信号打错，两个是因为verilog代码中控制信号敲错，不得不说，仔细真的很重要。

（三）借鉴了同学写的Python自动文件比对工具

standard = open("standard.txt", "r")

test = open("new.txt", "r")

error = 0

for i in range(1024):

std = standard.readline().rstrip('\n')

tt = test.readline().rstrip('\n')

if (std != tt):

print("ERROR")

error = 1

print(std)

print(tt)

if (error == 0):

print("OK")

standard.close()

test.close()