# CPU设计文档

## 模块规格

### IFU

#### 端口说明

表 1‑1 端口说明

|  |  |  |
| --- | --- | --- |
| 信号 | 方向 | 描述 |
| NPC\_Sel[1:0] | I | 下一指令来源 |
| Z | I | 运算结果为零 |
| Jr[31:0] | I | $ra |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| Instr[31:0] | O | 指令内容 |
| PC[31:0] | O | 顺序执行下一指令地址 |

#### 功能定义

表 1‑2 功能定义

|  |  |  |
| --- | --- | --- |
| 功能 | NPC\_sel[1:0] | 描述 |
| Add\_4 | 00 | 顺序指令 |
| Jump | 01 | 跳转指令 |
| Branch | 10 | 当Z为1时分支 |
| Jr | 11 | 跳转寄存器 |

### GRF

#### 端口说明

表 2‑1 端口说明

|  |  |  |
| --- | --- | --- |
| 信号 | 方向 | 描述 |
| A1[4:0] | I | 第一操作数地址 |
| A2[4:0] | I | 第二操作数地址 |
| A3[4:0] | I | 第三操作数地址 |
| Wd[31:0] | I | 待写入数据 |
| We | I | 写使能信号 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| Rd1[31:0] | O | 第一操作数 |
| Rd2[31:0] | O | 第二操作数 |

### ALU

#### 端口说明

表 3‑1 端口说明

|  |  |  |
| --- | --- | --- |
| 信号 | 方向 | 描述 |
| A[31:0] | I | A |
| B[31:0] | I | B |
| ALUOp[2:0] | I | 运算类型 |
| C[31:0] | O | C |
| N | O | (C < 0) |
| Z | O | (C == 0) |
| V | O | 运算溢出 |
| Carry | O | 运算进位 |

#### 功能定义

表 3‑2 功能定义

|  |  |  |
| --- | --- | --- |
| 功能 | ALUOp[2:0] | 描述 |
| Nop | 000 | 0 |
| Add | 001 | A+B |
| Sub | 010 | A-B |
| And | 011 | A&B |
| Or | 100 | A|B |
| Xor | 101 | A^B |
| Le | 110 | A<B |
| B | 111 | B |

### DM

#### 端口说明

表 4‑1 端口说明

|  |  |  |
| --- | --- | --- |
| 信号 | 方向 | 描述 |
| Addr[31:0] | I | 数据地址 |
| Din[31:0] | I | 待写入数据 |
| We | I | 写使能信号 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| Dout[31:0] | O | 数据 |

### EXT

#### 端口说明

表 5‑1 端口说明

|  |  |  |
| --- | --- | --- |
| 信号 | 方向 | 描述 |
| ExtOp[1:0] | I | 扩展功能 |
| Imm[15:0] | I | 待扩展立即数 |
| Ext[31:0] | O | 扩展结果 |

#### 功能定义

表 5‑2 功能定义

|  |  |  |
| --- | --- | --- |
| 功能 | ExtOp[1:0] | 描述 |
| {16’b0, Imm} | 00 | 无符号扩展 |
| {{16{Imm[15]}}, Imm} | 01 | 有符号扩展 |
| {Imm, 16’b0} | 10 | 扩展至低位 |
| 32’b0 | 11 | 复位 |

### MUX

#### RegDst

表 6‑1 RegDst功能定义

|  |  |  |
| --- | --- | --- |
| 信号 | RegDst[1:0] | 数据来源 |
| Instr[15:11] | 00 | R型指令 |
| Instr[20:16] | 01 | I型指令 |
| 31 | 10 | J型指令 |

#### ALUSrc

表 6‑2 ALUSrc功能定义

|  |  |  |
| --- | --- | --- |
| 信号 | ALUSrc | 数据来源 |
| Rd2 | 0 | 寄存器数据 |
| Ext | 1 | 已扩展立即数 |

#### DataSrc

表 6‑3 DataSrc功能定义

|  |  |  |
| --- | --- | --- |
| 信号 | DataSrc[1:0] | 数据来源 |
| C | 00 | ALU |
| Dout | 01 | DM |
| PC | 10 | IFU |

## 控制器设计

### 端口说明

表 1‑1 端口说明

|  |  |  |
| --- | --- | --- |
| 信号 | 方向 | 描述 |
| OpCode[5:0] | I |  |
| Funct[5:0] | I |  |
| RegDst[1:0] | O | 待回写寄存器地址来源 |
| ALUSrc | O | ALU第二操作数来源 |
| DataSrc[1:0] | O | 待回写数据来源 |
| RegWrite | O | 寄存器写使能信号 |
| MemWrite | O | 数据存储器写使能信号 |
| NPC\_Sel[1:0] | O | 下一指令地址来源 |
| ExtOp[1:0] | O | 立即数扩展方式 |
| ALUOp[2:0] | O | ALU运算方式 |

### 功能定义

表 2‑1 R型指令功能定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Funct | 001000 | 100001 | 100011 | 101010 |
| OpCode | 000000 | 000000 | 000000 | 000000 |
|  | jr | addu | subu | slt |
| RegDst[1:0] | xx | 00 | 00 | 00 |
| ALUSrc | 0 | 0 | 0 | 0 |
| DataSrc[1:0] | xx | 00 | 00 | 00 |
| RegWrite | 0 | 1 | 1 | 1 |
| MemWrite | 0 | 0 | 0 | 0 |
| NPC\_Sel[1:0] | 11 | 00 | 00 | 00 |
| ExtOp[1:0] | xx | xx | xx | xx |
| ALUOp[2:0] | 100 | 001 | 010 | 110 |

表 2‑2 I型指令功能定义

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| OpCode | 000100 | 001001 | 001101 | 001111 | 100011 | 101011 |
|  | beq | addiu | ori | lui | lw | sw |
| RegDst[1:0] | 01 | 01 | 01 | 01 | 01 | 01 |
| ALUSrc | 0 | 1 | 1 | 1 | 1 | 1 |
| DataSrc[1:0] | xx | 00 | 00 | 00 | 01 | 01 |
| RegWrite | 0 | 1 | 1 | 1 | 1 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 1 |
| NPC\_Sel[1:0] | 10 | 00 | 00 | 00 | 00 | 00 |
| ExtOp[1:0] | xx | 00 | 00 | 10 | 00 | 00 |
| ALUOp[2:0] | 010 | 001 | 100 | 111 | 001 | 001 |

表 2‑3 J型指令功能定义

|  |  |  |
| --- | --- | --- |
| OpCode | 000010 | 000011 |
|  | j | jal |
| RegDst[1:0] | xx | 10 |
| ALUSrc | x | x |
| DataSrc[1:0] | xx | 10 |
| RegWrite | 0 | 1 |
| MemWrite | 0 | 0 |
| NPC\_Sel[1:0] | 01 | 01 |
| ExtOp[1:0] | xx | xx |
| ALUOp[2:0] | xxx | xxx |

## 测试程序

由于IM容量限制，测试时先将IM扩容。测试了全部指令后，又将IM容量恢复。将规定指令集以外的部分指令测试段改为注释后又进行了一次测试。两次测试均正常。

测试程序中将需要检查的寄存器的值保存到了DM，在程序最后附有DM由0x00002000地址开始的连续几个寄存器的期望值。

.text

# addiu

addiu $s0, $zero, 0x00002000

addiu $t0, $zero, 10

sw $s0, 0($s0)

sw $t0, 4($s0)

addiu $t1, $s0, 8

sw $t1, 0($t1)

# addu

addu $t1, $t0, $t0

sw $t1, 0x0000200c

# beq

beq $t0, $t0, if\_1\_else

addiu $t0, $zero, 1

if\_1\_else:

addiu $t0, $t0, 1

sw $t0, 0x00002010

beq $t1, $s0, if\_2\_else

addiu $t0, $zero, 1

if\_2\_else:

addiu $t0, $t0, 1

sw $t0, 0x00002014

# subu

subu $t0, $t0, 1

sw $t0, 0x00002018

# slt

# slt $t1, $t0, $s0

# sw $t1, 0x0000201c

# slt $t1, $s0, $t0

# sw $t1, 0x00002020

# ori

addiu $t1, $zero, 6

ori $t2, $t1, 1

sw $t2, 0x00002024

# lui

lui $t0, 2

sw $t0, 0x0000202c

# jal, jr

# jal test

# addiu $t0, $zero, 1

# sw $t0, 0x00002034

# j

j jump

addiu $t0, $zero, 8

jump:

sw $t0, 0x00002038

# lw

lw $t0, 0x00002004

sw $t0, 0x0000203c

# test:

# sw $ra, 0x00002030

# jr $ra

#################

# check

# 0x00002000

# 0x0000000a

# 0x00002008

# 0x00000014

# 0x0000000b

# 0x00000002

# 0x00000001

# 0x00000000

# 0x00000000

# 0x00000007

# 0x00000000

# 0x00020000

# 0x00000000

# 0x00000000

# 0x00020000

# 0x0000000a

# 思考题

## 模块规格

程序计数器为30位寄存器时，可以节约硬件，执行beq或j指令时可以直接相加。但在与外部交换PC值时，如使用jal指令或jr指令时，就需要扩展低两位。

合理。到目前为止，指令寄存器不会被写入，只需读取。而且用RAM实现指令寄存器可能导致指令被改写，并不安全。数据存储器需要读写，而且容量大，用寄存器实现不现实，故选用RAM。GRF需要同时读取两个寄存器，写入一个寄存器，因此用memory实现比较繁琐，只能选用寄存器。

改进：数据存储器的数据写入和读取可以用同一根总线实现。

## 控制器设计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Funct | 100000 | 100010 |  |  | |
| OpCode | 000000 | 000000 | 001101 | 100011 | 101011 | | 000100 |
|  | add | sub | ori | lw | sw | | beq |
| RegDst | 1 | 1 | 0 | 0 | x | | x |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | | 0 |
| MemtoReg | 0 | 0 | 0 | 1 | x | | x |
| RegWrite | 1 | 1 | 1 | 1 | 0 | | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | | 0 |
| NPC\_Sel | 0 | 0 | 0 | 0 | 0 | | 1 |
| ExtOp | x | x | 0 | 1 | 1 | | x |
| ALUctr[2:0] | 001 | 010 | 011 | 001 | 001 | | 010 |

上述表达式即为最简。

不把指令加入真值表时，Controller输出信号全为0。此时，GRF和DM都不能写入，而NPC指向PC+4。符合nop指令的要求，所以nop指令不需要加入真值表。

## 测试CPU

可以将DM地址的高位经过与门判断是否符合数据段的要求，然后将与门的输出与原本的写使能信号进行与运算作为新的写使能信号。这样可以将不属于数据段的输入过滤掉。

形式验证的优点在于完全性。经过了形式验证且通过的电路不会存在错误。但缺点在于太过繁琐，运算量可能很大。形式验证可以看作设计的逆过程，对于形式验证中可以想到的情况，设计时也都应有考虑。因此由设计人员进行形式验证的意义不大。

编程测试虽然不能测试出全部输入情况，但可以针对有代表性的情况进行验证。用验证的可靠性换取了效率。