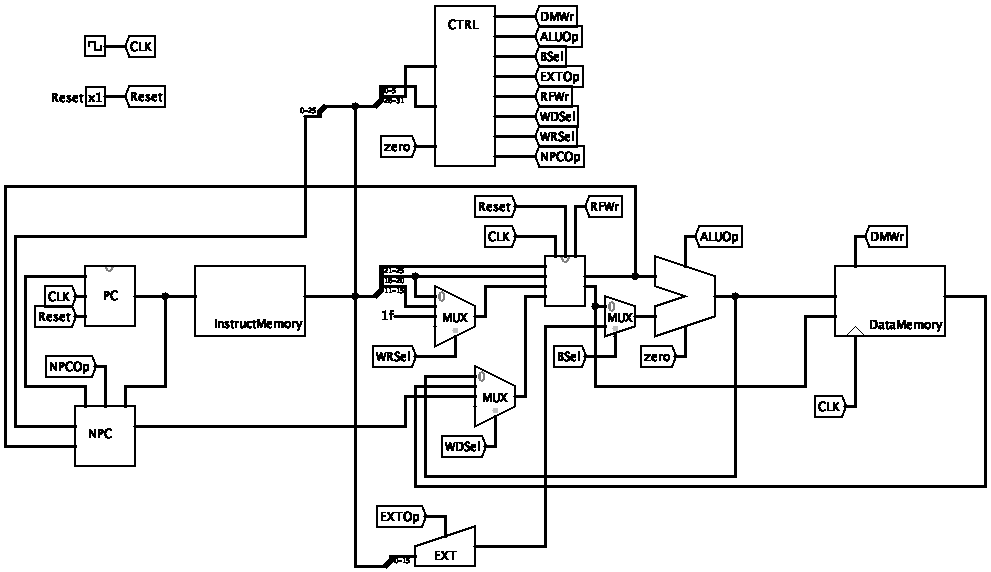
单周期CPU设计说明文档

一、数据通路设计

1、数据通路架构图



2、数据通路具体实现

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 部件 | PC | NPC | | | IM | GRF | | | |
| 输入信号 | npc | pc | Imm | rs | addr | A1 | A2 | A3 | WD |
| addu | NPC.npc | PC.pc |  |  | PC.pc | IM.Instr[25:21] | IM.Instr[20:16] | IM.Instr[15:11] | ALU.C |
| subu | NPC.npc | PC.pc |  |  | PC.pc | IM.Instr[25:21] | IM.Instr[20:16] | IM.Instr[15:11] | ALU.C |
| ori | NPC.npc | PC.pc |  |  | PC.pc | IM.Instr[25:21] |  | IM.Instr[20:16] | ALU.C |
| lw | NPC.npc | PC.pc |  |  | PC.pc | IM.Instr[25:21] |  | IM.Instr[20:16] | DM.RD |
| sw | NPC.npc | PC.pc |  |  | PC.pc | IM.Instr[25:21] |  | IM.Instr[20:16] |  |
| beq | NPC.npc | PC.pc | IM.Instr[15:0] |  | PC.pc | IM.Instr[25:21] | IM.Instr[20:16] |  |  |
| lui | NPC.npc | PC.pc |  |  | PC.pc | IM.Instr[25:21] |  | IM.Instr[20:16] | ALU.C |
| jal | NPC.npc | PC.pc | IM.Instr[25:0] |  | PC.pc |  |  | 0x1f | NPC.pc4 |
| jr | NPC.npc | PC.pc |  | GRF.RD1 | PC.pc | IM.Instr[25:21] |  |  |  |
|  | NPC.npc | PC.pc | IM.Instr[25:0] | GRF.RD1 | PC.pc | IM.Instr[25:21] | IM.Instr[20:16] | IM.Instr[20:16]  IM.Instr[15:11]  0x1f | ALU.C  DM.RD  NPC.pc4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 部件 | EXT | ALU | | DM | |
| 输入信号 | Imm | A | B | addr | WD |
| addu |  | GRF.RD1 | GRF.RD2 |  |  |
| subu |  | GRF.RD1 | GRF.RD2 |  |  |
| ori | IM.Instr[15:0] | GRF.RD1 | EXT.out |  |  |
| lw | IM.Instr[15:0] | GRF.RD1 | EXT.out | ALU.C |  |
| sw | IM.Instr[15:0] | GRF.RD1 | EXT.out | ALU.C | GRF.RD2 |
| beq |  | GRF.RD1 | GRF.RD2 |  |  |
| lui | IM.Instr[15:0] | GRF.RD1 | EXT.out |  |  |
| jal |  |  |  |  |  |
| jr |  |  |  |  |  |
|  | IM.Instr[15:0] | GRF.RD1 | GRF.RD2  EXT.out | ALU.C | GRF.RD2 |

二、模块规格

1、PC

（1）端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号 |
| npc | I | 32位存储数据 |
| pc | O | 输出32位PC存储的数据 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当reset为高电平时PC同步复位为0x0000\_3000 |
| 2 | 写入npc | 时钟上升沿写入npc端口传入的值 |

2、NPC

（1）端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| pc[31:0] | I | 当前指令PC值 |
| NPCOp[1:0] | I | 计算NPC功能选择 |
| Imm[25:0] | I | 26位立即数 |
| rs[31:0] | I | 32位寄存器值 |
| npc[31:0] | O | 输出32位NPC的值 |
| pc4[31:0] | O | 输出32位PC+4的值 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 次地址计算 | NPCOp=00，计算输出顺序地址（PC+4）  NPCOp=01，计算输出beq地址  NPCOp=10，计算输出jal地址  NPCOp=11，计算输出jr地址 |

3、IM

（1）端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| addr[31:0] | I | 输入32位地址 |
| Instr[31:0] | O | 输出32位指令 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取出指令 | 根据addr的值从IM中取出指令 |

4、GRF

（1）端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| A1[4:0] | I | 第1个读出寄存器的编号 |
| A2[4:0] | I | 第2个读出寄存器的编号 |
| A3[4:0] | I | 写入寄存器的编号 |
| RD1[31:0] | O | A1指向寄存器的值 |
| RD2[31:0] | O | A2指向寄存器的值 |
| WD[31:0] | I | 写入寄存器的值 |
| WE | I | 写入使能 |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | reset=1时在时钟上升沿所有寄存器复位为0x0000\_0000 |
| 2 | 读出寄存器 | A1和A2对应的32位寄存器值分别通过RD1和RD2输出 |
| 3 | 写入寄存器 | WE=1时，WD的值写入A3所指的寄存器 |

5、ALU

（1）端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| A[31:0] | I | 第1个32位操作数 |
| B[31:0] | I | 第2个32位操作数 |
| C[31:0] | O | 32位计算结果 |
| ALUOp[1:0] | I | ALU功能选择 |
| zero | O | A和B相等比较结果 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | ALU数学逻辑计算 | ALUOp=00，C=A+B  ALUOp=01，C=A-B  ALUOp=10，C=A|B  ALUOp=11，保留 |
| 2 | 相等比较 | zero=1，A=B  zero=0，AB |

6、DM

（1）端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| addr[31:0] | I | 地址输入 |
| WD[31:0] | I | 写入数据 |
| RD[31:0] | O | 读出数据 |
| WE | I | 写入使能 |
| clk | I | 时钟信号 |
| reset | I | 异步复位信号 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | RST=1时，在时钟上升沿RAM复位为零 |
| 2 | 读出数据 | 将addr所表示的内存地址中的值读出 |
| 3 | 写入数据 | WE=1时，将WD的值写入addr所表示的内存地址中 |

7、EXT

（1）端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| Imm[15:0] | I | 16位立即数输入 |
| out[31:0] | O | 32位扩展结果输出 |
| EXTOp[1:0] | I | 扩展功能信号 |

（2）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 扩展立即数 | EXTOp=00，进行零扩展  EXTOp=01，进行符号扩展  EXTOp=10，进行LUI扩展  EXTOp=11，保留 |

8、Controller

（见第二部分：控制器设计）

三、控制器设计

1、端口定义及功能说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| opcode[5:0] | I | Instr[31:26] |
| funct[5:0] | I | Instr[5:0] |
| zero | I | 相等比较信号 |
| DMWr | O | DM写使能信号 |
| ALUOp[1:0] | O | ALU运算信号 |
| BSel | O | ALU的B端MUX选择信号：  0：寄存器RD2值  1：EXT扩展数据 |
| EXTOp[1:0] | O | EXT扩展信号 |
| RFWr | O | GRF写使能信号 |
| WDSel[1:0] | O | 寄存器的WD端MUX选择信号：  00：ALU运算结果  01：DM输出数据  10：PC + 4  11：保留 |
| WRSel[1:0] | O | 寄存器的A3端MUX选择信号：  00：Instr[20:16]  01：Instr[15:11]  10：0x1f  11：保留 |
| NPCOp[1:0] | O | IFU中NPC运算信号 |

2、真值表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | addu | subu | ori | lw | sw | beq | lui | jal | jr |
| Op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 | 000000 |
| Func | 100001 | 100011 | NA | NA | NA | NA | NA | NA | 001000 |
| DMWr | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ALUOp | 00 | 01 | 10 | 00 | 00 | XX | X0 | XX | XX |
| BSel | 0 | 0 | 1 | 1 | 1 | X | 1 | X | X |
| EXTOp | XX | XX | 00 | 01 | 01 | XX | 10 | XX | XX |
| RFWr | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| WDSel | 00 | 00 | 00 | 01 | XX | XX | 00 | 10 | XX |
| WRSel | 01 | 01 | 00 | 00 | XX | XX | 00 | 10 | XX |
| NPCOp | 00 | 00 | 00 | 00 | 00 | 0 Zero | 00 | 10 | 11 |

四、测试模块

1、测试A

测试程序：

ori $s0, $0, 0x5555

ori $a0, $s0, 0xaaaa

ori $v0, $s0, 0x0000

lui $s0, 0x4321

lui $a0, 0x8765

lui $v0, 0x0000

lui $t0, 65535

ori $t0, $t0, 65535

ori $t1, $t0, 65535

lui $t1, 65535

期望结果：

@00003000: $16 <= 00005555

@00003004: $ 4 <= 0000ffff

@00003008: $ 2 <= 00005555

@0000300c: $16 <= 43210000

@00003010: $ 4 <= 87650000

@00003014: $ 2 <= 00000000

@00003018: $ 8 <= ffff0000

@0000301c: $ 8 <= ffffffff

@00003020: $ 9 <= ffffffff

@00003024: $ 9 <= ffff0000

2、测试B

测试程序：

ori $0, 0xffff

ori $1, 0xeeee

ori $2, 0xdddd

ori $4, 0xcccc

ori $8, 0xbbbb

ori $16, 0xaaaa

ori $26, 0x9999

ori $28, 0x8888

ori $29, 0xabcd

ori $31, 0xefef

addu $9, $0, $0

addu $10, $0, $1

addu $11, $2, $0

addu $12, $4, $0

addu $13, $0, $8

addu $14, $0, $16

addu $15, $0, $26

addu $17, $0, $28

addu $18, $29, $0

addu $19, $31, $0

期望结果：

@00003000: $ 0 <= 0000ffff

@00003004: $ 1 <= 0000eeee

@00003008: $ 2 <= 0000dddd

@0000300c: $ 4 <= 0000cccc

@00003010: $ 8 <= 0000bbbb

@00003014: $16 <= 0000aaaa

@00003018: $26 <= 00009999

@0000301c: $28 <= 00008888

@00003020: $29 <= 0000abcd

@00003024: $31 <= 0000efef

@00003028: $ 9 <= 00000000

@0000302c: $10 <= 0000eeee

@00003030: $11 <= 0000dddd

@00003034: $12 <= 0000cccc

@00003038: $13 <= 0000bbbb

@0000303c: $14 <= 0000aaaa

@00003040: $15 <= 00009999

@00003044: $17 <= 00008888

@00003048: $18 <= 0000abcd

@0000304c: $19 <= 0000efef

3、测试C

测试程序：

lui $s0, 65535

ori $s0, 65535

ori $s1, $0, 1

addu $t0, $0, $s1

subu $t1, $0, $s1

addu $s0, $s0, $s1

subu $s1, $s1, $s1

lui $a0, 0xeded

ori $a0, $a0, 0xeded

lui $a1, 0xcccc

ori $a1, $a1, 0xcccc

addu $v0, $a0, $a1

addu $v0, $v0, $a1

subu $v0, $a0, $a1

subu $v0, $a1, $a0

期望结果：

@00003000: $16 <= ffff0000

@00003004: $16 <= ffffffff

@00003008: $17 <= 00000001

@0000300c: $ 8 <= 00000001

@00003010: $ 9 <= ffffffff

@00003014: $16 <= 00000000

@00003018: $17 <= 00000000

@0000301c: $ 4 <= eded0000

@00003020: $ 4 <= edededed

@00003024: $ 5 <= cccc0000

@00003028: $ 5 <= cccccccc

@0000302c: $ 2 <= bababab9

@00003030: $ 2 <= 87878785

@00003034: $ 2 <= 21212121

@00003038: $ 2 <= dedededf

4、测试D

测试程序：

.data

number: .space 12

.text

ori $t0, $0, 1

ori $t1, $0, 2

ori $t2, $0, 3

ori $s0, $0, 0

sw $t0, 0($s0)

sw $t1, 4($s0)

sw $t2, 8($s0)

ori $s0, $0, 8

lw $t0, 0($s0)

lw $t1, -4($s0)

lw $t2, -8($s0)

期望结果：

@00003000: $ 8 <= 00000001

@00003004: $ 9 <= 00000002

@00003008: $10 <= 00000003

@0000300c: $16 <= 00000000

@00003010: \*00000000 <= 00000001

@00003014: \*00000004 <= 00000002

@00003018: \*00000008 <= 00000003

@0000301c: $16 <= 00000008

@00003020: $ 8 <= 00000003

@00003024: $ 9 <= 00000002

@00003028: $10 <= 00000001

4、测试E

测试程序：

ori $s0, 20

ori $s1, 10

addu $t0, $s0, $s1

jal a

subu $t1, $s1, $s0

sw $t0, 0($0)

sw $t1, 4($0)

lw $t0, 4($0)

lw $t1, 0($0)

beq $0, $0, end

*a:*

lui $a0, 0xffff

ori $a0, 0xffff

jr $ra

*end:*

nop

nop

nop

期望结果：

@00003000: $16 <= 00000014

@00003004: $17 <= 0000000a

@00003008: $ 8 <= 0000001e

@0000300c: $31 <= 00003010

@00003028: $ 4 <= ffff0000

@0000302c: $ 4 <= ffffffff

@00003010: $ 9 <= fffffff6

@00003014: \*00000000 <= 0000001e

@00003018: \*00000004 <= fffffff6

@0000301c: $ 8 <= fffffff6

@00003020: $ 9 <= 0000001e

五、思考题

1、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？

lw与sw都是按字寻址的指令，且addr信号来源于ALU计算的结果，即ALU.C，而对DM中RAM的寻址一般使用ALU.C[11:2]的数据，故用[11:2]定义addr的位数。

2、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是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

针对PC、DM、GRF。

PC清零为0x00003000，使PC值所指向的IM中所取出的指令为我们所要的第一条指令。

DM和GRF清零为0，将DM和GRF中失效的数据和未初始化的数据都置为零，提供一个干净的环境。

3、列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

（1）if\_else型：

always @(\*) begin

if(opcode==6’b000100) begin

RFWr = 1’b1;

……

end

else if(opcode==6’b001010) begin

RFWr = 1’b1;

……

end

……

end

（2）case型：

always @(\*) begin

case(opcode)

6’b000000: ……

6’b000100: ……

6’b001010: ……

……

default: ……

endcase

end

（3）assign型：

assign jal = ~opcode[5] & ~opcode[4] & ~opcode[3]

& ~opcode[2] & opcode[1] & opcode[0];

……

assign RFWr = jal | lw | addu | …… ;

……

（4）define型：

`define JAL ~opcode[5] & ~opcode[4] & ~opcode[3]

& ~opcode[2] & opcode[1] & opcode[0]

……

assign RFWr = (`JAL) | (`LW) | …… ;

`define JAL 6’b000011

……

always @(\*) begin

case(opcode)

`JAL: ……

`LW: ……

……

default: ……

endcase

end

（5）门级原语：

wire jal, jr, …;

and (jal, ~opcode[5] , ~opcode[4] , ~opcode[3] , ~opcode[2] , opcode[1] , opcode[0]);

……

or (RFWr, jal, lw, …);

4、根据你所列举的编码方式，说明他们的优缺点。

使用宏定义的方法可以使得代码更加直观和易读，使开发者修改指令时轻松方便，让冗长的表达式变得清晰简单，但相较于使用中间变量保存指令有效信号，在观察波形时，因为宏只是文本的替换，不能很好观察到相关信号的变化。

使用assign和or、and原语，让控制器的与、或逻辑十分清晰，而if\_else和case让每个指令分别对应相应的信号改变，能够准确的定位每个指令的不同信号。这两大类方式各有千秋，应根据构建者的偏好进行选择。

5、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。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分 。

addi将rs符号扩展为33位，将Imm符号扩展为33位，最后进行相加，若不溢出，取低32位存入rt中；addiu将Imm符号扩展为32位同rs进行相加并存入rt中，若不考虑溢出，则addi和addiu都是将rs 与符号扩展的Imm相加并存入rt，其结果并没有差别。

同理，在不考虑溢出时，add与addu也只是将rs与rt相加存入rd的过程。

6、根据自己的设计说明单周期处理器的优缺点。

优点：CPU设计简单，体积小，成本低，不需要设计大量寄存器和控制信号逻辑电路。

缺点：与流水线CPU相比，时钟频率低，指令吞吐量小，不能充分利用内部模块。

7、简要说明jal、jr和堆栈的关系。

在调用子函数时，使用jal跳转至新的地址处，并将原函数中需要执行的下一个指令的地址存入$ra中，接着，在子函数将$ra中的值入栈进行保存，在执行完函数功能后，通过出栈来恢复保存的$ra的值，并使用jr跳回原函数继续原函数的功能实现。