MIPS单周期处理器设计文档

MIPS单周期处理器设计文档由顶层模块、数据通路、控制器、测试程序组成。

1. 顶层模块mips.v

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 功能描述 |
| 1 | clk | 1 | I | pc、grf、dm的时钟信号 |
| 2 | reset | 1 | I | 当reset=1且时钟为上跳沿时，pc被设置为0x00003000，gpr所有寄存器置零，dm所有数据置零 |

顶层模块将数据通路模块datapath.v与控制器模块control,v相连。

1. 数据通路datapath.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | reset | 1 | I | 复位信号 |
| 2 | clk | 1 | I | 时钟信号 |
| 3 | PCOP | 2 | I | muxPCOP模块控制信号 |
| 4 | RegDst | 2 | I | muxRegDst模块控制信号 |
| 5 | ExtOP | 1 | I | Ext模块控制信号 |
| 6 | RegWrite | 1 | I | 寄存器文件写使能 |
| 7 | RegWData | 2 | I | muxRegWData模块控制信号 |
| 8 | ALUSrc | 1 | I | muxALUSrc模块控制信号 |
| 9 | ALUOP | 4 | I | 选择ALU运算类型的信号 |
| 10 | MemWrite | 1 | I | dm写入控制信号 |
| 11 | instruction | 32 | O | 指令 |

datapath.v将pc.v, dm.v , im.v , alu.v , gpr.v , ext.v , muxPCOP.v , muxRegWData.v , muxRegDst.v , muxALUSrc.v 连接在一起。下面将介绍这些模块。

1. gpr.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | RA1 | 5 | I | 读寄存器文件时第一个寄存器的下标 |
| 2 | RA2 | 5 | I | 读寄存器文件时第二个寄存器的下标 |
| 3 | Waddr | 5 | I | 写入文件的寄存器下标 |
| 4 | WData | 32 | I | 寄存器文件写入数据 |
| 5 | clk | 1 | I | 时钟信号 |
| 6 | reset | 1 | I | 复位信号 |
| 7 | RegWrite | 1 | I | 寄存器文件写使能 |
| 8 | RData1 | 32 | O | 读寄存器文件时第一个寄存器的输出 |
| 9 | RData2 | 32 | O | 读寄存器文件时第二个寄存器的输出 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器 | RData1输出RA1所寻址的寄存器的文件数据，RData2输出RA2所寻址的寄存器的文件数据 |
| 2 | 写寄存器 | 当时钟上升沿到来时，并且reset=0且RegWrite有效时，WD被写入WA所寻址的寄存器 |
| 3 | 复位 | reset=1 且 clk为上升沿时gpr所有寄存器置零 |

1. alu.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | ALU\_A | 32 | I | ALU的第一个操作数 |
| 2 | ALU\_B | 32 | I | ALU的第二个操作数 |
| 3 | ALUOP | 4 | I | 选择ALU运算类型的信号 |
| 4 | ALU\_C | 32 | O | ALU的运算结果 |
| 5 | equal | 1 | O | 判断ALU\_A与ALU\_B是否相等，若相等输出1，否则输出0 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法 | 输出端ALU\_C=ALU\_A+ALU\_B |
| 2 | 减法 | 输出端ALU\_C=ALU\_A-ALU\_B |
| 3 | 或 | 输出端ALU\_C=ALU\_A|ALU\_B |
| 4 | 判断相等 | 若ALU\_A=ALU\_B，equal信号输出1，否则输出0 |

1. dm.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | addr | 32 | I | 写入数据储存器的数据的地址/读取数据储存器的数据的地址 |
| 2 | din | 32 | I | 写入数据储存器的数据 |
| 3 | MemWrite | 1 | I | 写入控制信号 |
| 4 | clk | 1 | I | 时钟信号 |
| 5 | reseet | 1 | I | 复位信号 |
| 6 | dout | 32 | O | 从数据存储器读取的数据 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据储存器 | dout输出数据储存器在addr[11:2]地址的数据 |
| 2 | 写数据储存器 | 当MemWrite有效,reset=0,并且时钟为上升沿时，din被写入数据存储器地址为addr[11:2]的区域 |
| 3 | 复位 | reset=1并且时钟为上升沿时,DM所有数据置零 |

1. ext.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | ext\_in | 16 | I | 位扩展输入信号 |
| 2 | EXTOP | 1 | I | 信号为1时，进行符号扩展，信号为0时，进行0扩展 |
| 3 | ext\_out | 32 | O | 位扩展输出信号 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 符号扩展 | 当EXTOP信号为1时，ext\_in符号扩展后由ext\_out输出 |
| 2 | 0扩展 | 当EXTOP信号为0时，在ext\_in前置16个0后由ext\_out输出 |

1. im.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | address | 32 | I | 指令地址 |
| 2 | instr | 32 | O | 指令 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取指令 | 取出位于(address-0x00003000)[11:2]处指令 |

1. pc.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | PC\_in | 32 | I | 下一条指令PC值 |
| 2 | clk | 1 | I | 时钟信号 |
| 3 | reset | 1 | I | 复位信号 |
| 4 | PC\_out | 32 | O | 当前PC值 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出指令地址 | 时钟上升沿且reset=0是PC输出指令地址 |
| 2 | 复位 | 时钟上升沿且reset=1，将PC置为0x00003000 |

1. muxPCOP.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | PCOP | 2 | I | muxPCOP控制信号 |
| 2 | equal | 1 | I | ALU\_A与ALU\_B是否相等 |
| 3 | PC | 32 | I | 当前指令PC值 |
| 4 | RD1 | 32 | I | 读寄存器文件时第一个寄存器的输出 |
| 5 | EXT\_out | 32 | I | ext输出端 |
| 6 | instr | 32 | I | 当前指令 |
| 7 | newPC | 32 | O | 下一条指令PC值 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出下条指令地址 | PCOP=2’b00时：  newPC=PC+4  PCOP=2’b01时：若equal=1  newPC={{EXT\_out[29:0]},2'b00}+PC+4  否则 newPC=PC+4  PCOP=2’b10时：  newPC=RD1  PCOP=2’b11时：  newPC={{PC[31:28]},{{instr[25:0]},2'b00}} |

1. muxRegDst.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | RegDst | 2 | I | muxRegDst控制信号 |
| 2 | instr | 32 | I | 当前指令 |
| 3 | WAddr | 5 | O | 写入文件的寄存器下标 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出写入文件的寄存器下标 | RegDst=2’b00，WAddr=rd  RegDst=2’b01，WAddr=rt  RegDst=2’b10，WAddr=31 |

1. muxRegWData.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | ALU\_C | 32 | I | ALU结果 |
| 2 | DM\_Data | 32 | I | 从数据存储器读取的数据 |
| 3 | PC | 32 | I | 当前指令PC值 |
| 4 | RegWData | 2 | I | muxRegWData的控制信号 |
| 5 | WData | 32 | O | 寄存器文件写入数据 |
| 6 | instr | 32 | I | 当前指令 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出寄存器文件写入数据 | RegWData=2’b00时：WData=ALU\_C  RegWData=2’b01时：WData=DM\_Data  RegWData=2’b10时：  WData={instr[15:0],{16{1'b0}}}  RegWData=2’b11时：WData=PC+4 |

1. muxALUSrc.v

信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | ALUSrc | 1 | I | muxAlUSrc的控制信号 |
| 2 | RD2 | 32 | I | 读寄存器文件时第二个寄存器的下标 |
| 3 | EXT\_out | 32 | I | 位扩展输出信号 |
| 4 | ALU\_B | 32 | O | ALU的第二个操作数 |

模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择ALU的第二个操作数 | ALUSrc=0时：ALU\_B=RD2  ALUSrc=1时：ALU\_B=EXT\_out |

1. 控制器control.v

以下为控制信号产生的真值表：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 | 无 | | | | | | 001000 | 无 |
| opcode | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 | 000000 |  |
| 控制信号 | addu | subu | ori | lw | sw | beq | lui | jal | jr | nop |
| PCOP | 00 | 00 | 00 | 00 | 00 | 01 | 00 | 11 | 10 | 0 |
| EXTOP | x | x | 0 | 1 | 1 | 1 | x | x | x | x |
| RegDst[1:0] | 00 | 00 | 01 | 01 | x | x | 01 | 10 | x | x |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| RegWData[1:0] | 00 | 00 | 00 | 01 | x | x | 10 | 11 | x | x |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | x | x | x | x | x |
| ALUOP[3:0] | 0000 | 0001 | 0010 | 0000 | 0000 | x | x | x | x | x |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

control.v信号端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号 | 位数 | 方向 | 描述 |
| 1 | OP | 6 | I | OPcode |
| 2 | FUNC | 6 | I | Function code |
| 3 | PCOP | 2 | O | muxPCOP控制信号 |
| 4 | RegDst | 2 | O | muxRegDst控制信号 |
| 5 | ExtOP | 1 | O | ext控制信号 |
| 6 | RegWrite | 1 | O | gpr写入控制信号 |
| 7 | RegWData | 2 | O | muxRegWData的控制信号 |
| 8 | ALUSrc | 1 | O | muxAlUSrc的控制信号 |
| 9 | ALUOP | 4 | O | alu控制信号 |
| 10 | MemWrite | 1 | O | dm写入控制信号 |

1. 测试程序

Nop 机器码：00000000

lui $2,0x0101 3c020101

sw $2,4($0) ac020004

Nop 00000000

lw $3,4($0) 8c030004

addu $2,$2,$3 00431021

subu $4,$2,$4 00442023

beq $4,$2,label 10820004

label1:

lw $1,4($0) 8c010004

sw $31,12($0) ac1f000c

ori $7,$31,1 37e70001

jr $31 03e00008

label:

ori $5,$2,1 34450001

subu $5,$5,$2 00a22823

lui $6,100 3c060064

sw $5,8($0) ac050008

jal label1 0c000c08

addu $8,$7,$2 00e24021

Nop 00000000

测试期望：ISM依次输出：

$ 2 <= 01010000

\*00000004 <= 01010000

$ 3 <= 01010000

$ 2 <= 02020000

$ 4 <= 02020000

$ 5 <= 02020001

$ 5 <= 00000001

$ 6 <= 00640000

\*00000008 <= 00000001

$31 <= 00003044

$ 1 <= 01010000

\*0000000c <= 00003044

$ 7 <= 00003045

$ 8 <= 02023045

测试结束：$1=$3=0x01010000

$2=$4=0x02020000

$5=0x00000001

$6=0x00640000

$7=0x00003045

$8=0x02023045

$31=0x00003044

dm地址为4处：0x01010000

地址为8处：0x00000001

地址为12处：0x00003044

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？

因为ise中相邻地址单元地址相差1，而实际上应该相差4，所以取[11:2]，addr信号从ALU结果处来。

1. 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,GPR,因为这些部件会被进行读写操作，不复位可能会有残留之前写入的值，影响后续操作。

1. 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。

利用case语句：case(OP)

6’b101011:MemWrite<=1;

.........

利用assign语句：assign sw =(OP==6'b101011)?1:0;

assign MemWrite=sw;

.........

1. 根据你所列举的编码方式，说明他们的优缺点。

第一种：优点在于一目了然，缺点在于运用了时序逻辑。

第二种：优点在于简单，但要实现必须简化。

1. 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与add运用双符号位，32号位与31号位为符号位，若不考虑溢出，则相加后，30号位不会向31号位进位，最后32号位与31号位相同，所以add与addu，addi与addiu等价。

1. 根据自己的设计说明单周期处理器的优缺点。

优点：元器件少，结构相对简单。

缺点：所有指令执行周期都和最慢指令相同，浪费时间，IM与DM分离，不现实。

1. 简要说明jal、jr和堆栈的关系。

每次使用jal都相当于向$31压入PC+4相当于压栈，但是类似于栈，jr只跳转到最后一个压入寄存器的值，像出栈一样，后入先出。