# Verilog单周期处理器

20373864 谭立德

## 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{ lw、sw、beq、addi、j、ori、lui、add、sub、and、or、addu、subu、lb、sb、lh、sh、jal 、jr、nop }。为了实现这些功能，CPU主要包含了IM、GRF、DM、ALU、PC、CU ，这些模块按照自顶向下的顶层设计逐级展开。

### （二）关键模块定义

#### 1. GRF （通用寄存器组，也称为寄存器文件、寄存器堆）

GRF端口定义：

表0 GRF端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号，将32个寄存器中的值全部清零  1：复位  0：无效 |
| We | I | 写使能信号  1：可向GRF中写入数据  0：不可向GEF中写入数据 |
| 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至RD1 |
| 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至RD2 |
| 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 |
| WD3 | I | 32位数据输入信号 |
| RD1 | O | 输出指定的寄存器中的32位数据 |
| RD2 | O | 输出指定的寄存器中的32位数据 |

GRF模块功能定义：

表1 GRF功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | Reset信号有效时，所有寄存器储存的数值清零 |
| 2 | 读数据 | 读出A1，A2地址对应寄存器中所储存的数据到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 |

1. DM （数据存储器）：

DM端口定义：

表2 DM端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号，将32个寄存器中的值全部清零  1：复位  0：无效 |
| We | I | 写使能信号  1：可向DM中写入数据  0：不可向DM中写入数据 |
| A | I | 5位地址输入信号，指定中储存器上的地址，将其中存储的数据读出至RD1 |
| WD | I | 32位数据输入信号 |
| RD | O | 输出储存器指定地址上的32位数据 |

DM模块功能定义：

表3 DM功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | Reset信号有效时，储存器储存的所有数值清零 |
| 2 | 读数据 | 读出A地址对应储存器中所储存的数据到RD |
| 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 |

#### ALU （算术逻辑运算单元）：

ALU端口定义：

表4 ALU端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| SrcA | I | 32位运算数输入信号 |
| SrcB | I | 32位运算数输入信号 |
| ALU Control | I | 3位逻辑运算选择信号，选择进行哪种逻辑运算 |
| Zero | O | 输出比较两运算数比较的1位输出 |
| ALU Result | O | 输出对两运算数进行指定逻辑运算后的32位结果 |

ALU模块功能定义：

表5 ALU功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 计算 | 根据控制信号进行对应的逻辑计算并输出 |
| 2 | 比较 | 判断两个输入是否相等 |

#### IM （指令存储器）：

IM端口定义：

表6 IM端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC | I | 5位输入地址信号 |
| Instr | 0 | 输出地址所储存32位指令 |

IM模块功能定义：

表7 IM功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 读指令 | 根据输入输出对应32位指令 |

#### Control Unit （指令译码器）：

Control Unit端口定义：

表8 Control Unit端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Opcode[5:0] | I | 指令操作码 |
| Funct[5:0] | I | 指令功能码 |
| Jump | O | 跳转信号 |
| ToHigh16 | O | 高位置位信号 |
| ExtOp | O | 位扩展方式 |
| MemtoReg | O | 读内存信号 |
| MemWrite | O | 内存写使能信号 |
| Branch | O | 分支信号 |
| ALUCtrl[2:0] | O | ALU控制信号 |
| ALUSrc | O | ALU操作数2的来源0：寄存器1：立即数 |
| RegDst | O | 寄存器写地址选择0：Instr[20:16] 1：Instr[15:11] |
| RegWrite | O | 寄存器写使能信号 |
| DMop[1:0] | O | 存储、读取方式控制信号 |

### （三）重要机制实现方法

#### 1. J类型指令

根据输入判断和ALU模块协同工作算出跳转地址后跳转。

#### 2. R类型指令

根据输入判断和ALU模块协同工作算出结果后存储回寄存器堆中以实现指令R类型指令。

#### 3. I类型指令

根据输入判断和ALU模块和DM模块协同工作支持I类型指令。

## 测试方案

测试程序:（同logisim测试程序）

lui $t0,0x0004 # lui：立即数0x0004加载至t0寄存器的高位

lui $t1,0x0008 # lui：立即数0x0008加载至t1寄存器的高位

ori $t3,$zero,0x00002000 # ori：zero寄存器中的内容与立即数0x00002000进行或运算，储存在t3寄存器中

sw $t0,4($t3) # sw：把t0寄存器中值（1Word），存储到t3的值再加上偏移量4,所指向的RAM中

sw $t0,8($t3) # sw：把t0寄存器中值（1Word），存储到t3的值再加上偏移量8,所指向的RAM中

loop:add$t2,$t2,$t1 # add：t1寄存器中的值加上t2寄存器中的值后存到t2寄存器中

lw $t4,4($t3) # lw：把t3寄存器的值+4当作地址读取存储器中的值存入t4

lui $t5,0x0004 # lui：立即数0x0004加载至t5寄存器的高位

sub $t7,$t6,$t5 # sub：t6寄存器中的值减去t5寄存器中的值后存到t7寄存器中

sub $t0,$t0,$t5 # sub：t0寄存器中的值减去t5寄存器中的值后存到t0寄存器中

add $t6,$t6,$t0 # add：t6寄存器中的值加上t0后存到t6寄存器中

beq $t0,$t1,loop # beq：判断t0的值和t1的值是否相等，相等转

loop

add $t0,$t0,$t5 # add：t0寄存器中的值加上t5后存到t0寄存器中

lui $v0,0x0001 # lui：立即数0x0001加载至v0寄存器的高位lui$v1,0x0002 # lui：立即数0x0002加载至v1寄存器的高位add $v0,$v0,$v1 # add：v0寄存器中的值加上v1后存到v0寄存器中

add $v1,$v0,$v1 # add：v0寄存器中的值加上v1后存到v1寄存器中

ori $a0,$v0,0xffff # ori：v0寄存器中的内容与立即数0xffff进行或运算，储存在a0寄存器中

sub $a1,$a0,0x0000ffff # sub：a0寄存器中的值减去立即数0x0000ffff后存到a1寄存器中

loop2: sub $a2,$v1,$v0 # sub：v1寄存器中的值减去v0中的值后存到a2寄存器中

add $a1,$a2,$a1 # add：a2寄存器中的值加上a1后存到a1寄存器中

beq $a1,$v1,loop2 # beq：判断a1的值和v1的值是否相等，相等转loop2

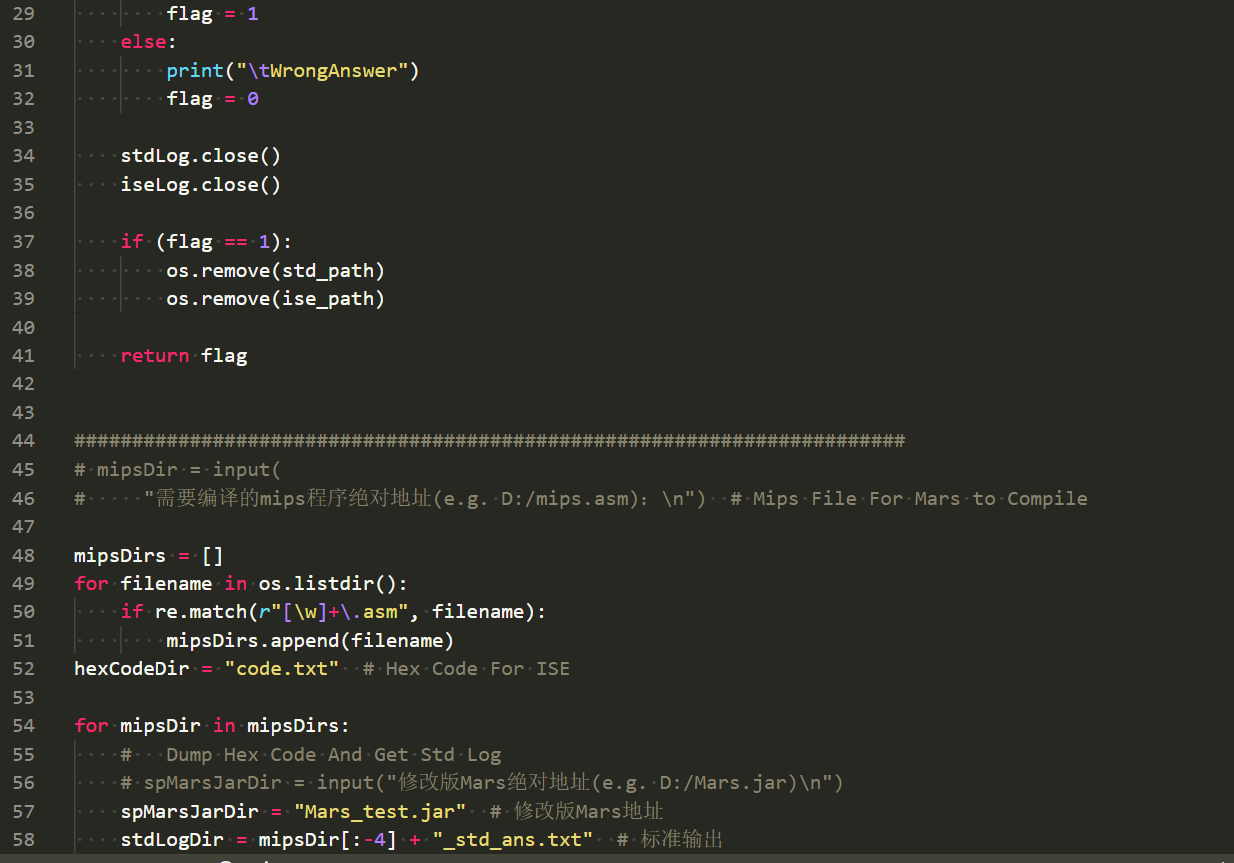
对应机器码：3c080004 3c090008 340b2000 ad680004 01495020 8d6c0004

3c0d0004 01cd7822 010d4020 01c87020 1109fff9 010d4020 3c020001

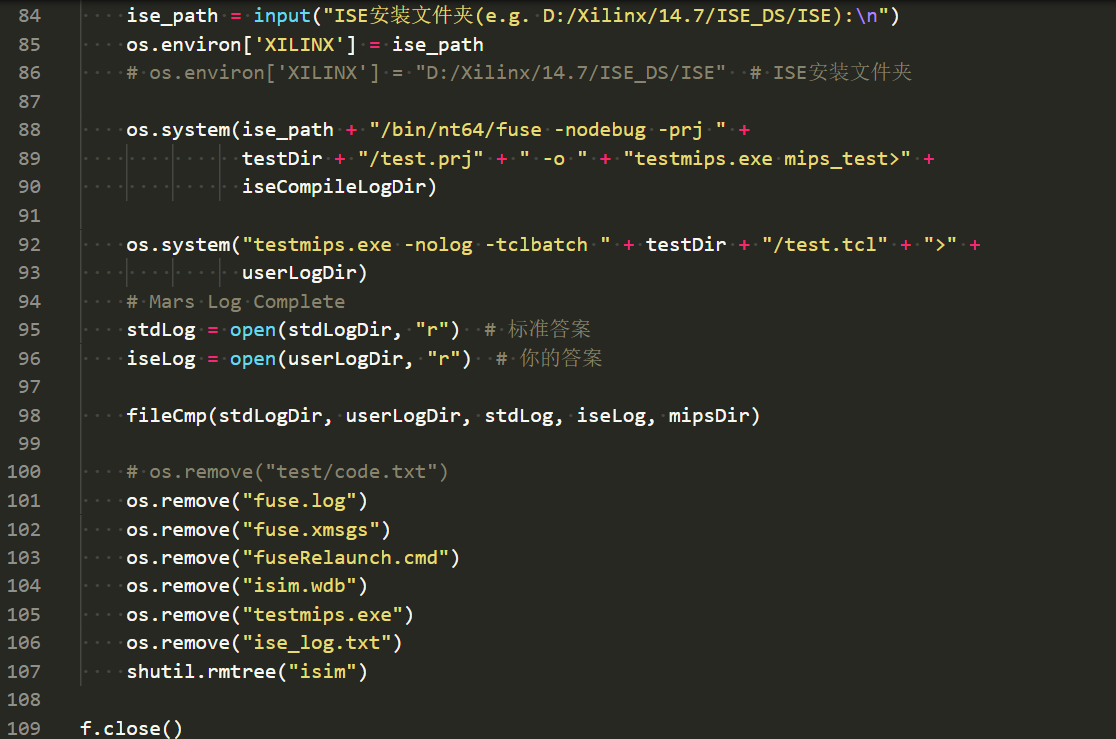
3c030002 00431020 00431820 3444ffff 3c010000 3421ffff 00812822

00623022 00c52820 10a3fffd









## 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？IMG_256

### 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。

解：

地址为[11:2]时，可以截取传入地址的第11位到第2位信号。由于DM传入地址位字节地址，而存储器是按字存储，相差4倍，因此后两位必然为0，要取第11到2位作为字地址。addr信号位寻址结果，因此来自ALU运算结果。

### （二）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？

解：

数据需要初始加载或者在运算过程中负责储存。

### （三）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 。

解：

有符号溢出，就是计算结果符号位被进位（或借位）覆盖导致符号错误。在add、addi指令中，发生这种情况将会产生IntegerOverflow异常信号（错误算术运算）导致程序终止。但是若忽略溢出，二进制补码的加法方式决定有符号加法和无符号加法是等价的，不会发生异常就等价于无符号加法。

### （四）根据自己的设计说明单周期处理器的优缺点。

解：

优点：数据通路简单，没有延时槽，没有数据冲突和控制冲突。

缺点：和实际硬件不同，数据存储器和指令存储器分开，而在实际的体系结构中不可能。而且，寻址需要另立加法器，而算术器件是比较耗费晶体管和时间的。此外，由于不同指令由于关键路径不同而延迟时间不同，导致时钟周期由最慢的指令决定，这严重降低执行效率。

附1：

主译码器真值表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | Opcode | Reg  Write | ExtOp | RegDst | ToHigh16 | ALUSrc | Branch | Mem  Write | MemtoReg | R | Jump | DM  op | ALU  Control | … |
| R | 000000 | 1 | x | 01 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | xx | … |  |
| lw | 100011 | 1 | 1 | 00 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 00 | 010 |  |
| sw | 101011 | 0 | 1 | x | 0 | 1 | 0 | 1 | x | 0 | 0 | 00 | 010 |  |
| beq | 000100 | 0 | 1 | x | 0 | 0 | 1 | 0 | x | 0 | 0 | xx | 110 |  |
| addi | 001000 | 1 | 1 | 00 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | xx | 010 |  |
| j | 000010 | 0 | x | x | 0 | x | x | 0 | x | 0 | 1 | xx | xxx |  |
| ori | 001101 | 1 | 0 | 00 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | xx | 001 |  |
| lui | 001111 | 1 | 0 | 00 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | xx | xxx |  |
| lb | 100000 | 1 | 1 | 00 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 01 | 010 |  |
| sb | 101000 | 0 | 1 | x | 0 | 1 | 0 | 1 | x | 0 | 0 | 01 | 010 |  |
| lh | 100001 | 1 | 1 | 00 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 10 | 010 |  |
| sh | 101001 | 0 | 1 | x | 0 | 1 | 0 | 1 | x | 0 | 0 | 10 | 010 |  |
| jal | 000011 | 1 | x | 10 | 0 | x | x | 0 | x | 0 | 1 | xx | xxx |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

附2：

ALU译码器真值表：R类

|  |  |
| --- | --- |
| Funct | ALUControl |
| 100000(add) | 010（加） |
| 100010(sub) | 110（减） |
| 100100(and) | 000（与） |
| 100101(or) | 001（或） |
| 101010(slt) | 111（小于置位） |
| 100001(addu) | 010 |
| 100011(subu) | 110 |
| 001001(jr) | xxx |

附3：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 与 | 或 | 加 |  |  |  | 减 |  |