Verilog单周期CPU设计文档

一、模块

1.IFU

（1）介绍

取指令单元，内部包括 PC（程序计数器）、IM(指令存储器)及相关逻辑。

（2）端口定义

表格 1 IFU端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| ifJ | I | 1 | 当前指令是否为J |
| ifBcom | I | 1 | 当前指令是否满足跳转要求 |
| reset | I | 1 | 异步复位信号 1: 复位 0: 无效 |
| clk | I | 1 | 时钟信号 |
| Instr | O | 32 | 当前指令 |

（3）功能定义

表格 2 IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，PC被设置为0x00000000 |
| 2 | 取指令 | 当时钟上升沿到来，根据PC[6:2]从IM取指令 |
| 3 | 计算下一个PC | 如果当前指令满足跳转要求，PC=PC+4+EXT(imm||02)  如果当前指令为J指令，PC=PC[31:28]||instr\_index||02  否则，PC=PC+4 |

2.GRF

（1）介绍

通用寄存器组，也称为寄存器文件、寄存器堆。可以存取32位数据。

（2）端口定义

表格 3 GRF端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| A1 | I | 5 | 指定32个寄存器中的一个，输出其中数据到RD1 |
| A2 | I | 5 | 指定32个寄存器中的一个，输出其中数据到RD2 |
| A3 | I | 5 | 指定32个寄存器中的一个，写入Data数据 |
| Data | I | 32 | 输入数据 |
| WE | I | 1 | 写入使能信号 1: 可写入 0: 不可写入 |
| reset | I | 1 | 异步复位信号 1: 复位 0: 无效 |
| clk | I | 1 | 时钟信号 |
| RD1 | O | 32 | A1指定寄存器中的数据 |
| RD2 | O | 32 | A2指定寄存器中的数据 |

（3）功能定义

表格 4 GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，所有寄存器的数据清零 |
| 2 | 读数据 | 读出A1,A2指定寄存器中数据到RD1,RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿时，将Data写入A3指定寄存器 |

3.ALU

（1）介绍

算术逻辑单元，提供32位加、减、与、或运算，不检测溢出。

（2）端口定义

表格 5 ALU端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| A | I | 32 | ALU的输入1 |
| B | I | 32 | ALU的输入2 |
| sel | I | 2 | 选择信号，00: A+B 01: A-B 10: A&B 11: A|B |
| C | O | 32 | 运算结果 |
| notzero | O | 1 | 判断C是否为零1: C=0 0: C!=0 |

（3）功能定义

表格 6 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 运算 | 按照sel信号选择C为A和B做什么运算得到的结果 |
| 2 | 判零 | 判断C是否为0 |

4.DM

（1）介绍

存储数据。

（2）端口定义

表格 7 DM端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| address | I | 5 | 待操作地址 |
| data | I | 32 | 待输入数据 |
| WE | I | 1 | 写入使能信号1: 可写入 0: 不可写入 |
| clk | I | 1 | 时钟信号 |
| reset | I | 1 | 异步复位信号 1: 复位 0: 无效 |
| out | O | 32 | 读出的数据 |

（3）功能定义

表格 8 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 写数据 | 当WE有效且时钟上升沿时，将data写入address地址 |
| 2 | 读数据 | 从address中读取数据，输出至out |
| 3 | 复位 | 当复位信号有效时，所有ROM的数据清零 |

5.EXT

（1）介绍

将16位立即数扩展为32位。

（2）端口定义

表格 9 EXT端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| A | I | 16 | 待扩展的16位立即数 |
| sel | I | 2 | 扩展方式选择信号  00: 无符号扩展  01: 有符号扩展  10: 后面拼接两个0后符号扩展  11: 加载至高位 |
| B | O | 32 | 扩展后的数 |

（3）功能定义

表格 10 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 扩展 | 按照sel信号选择B为A做什么扩展得到的结果 |

6.Controller

（1）介绍

根据指令有关信息（opcode，func）判断指令类型，进而得到各个选择器、使能信号等的数据，决定各组件控制信号。

（2）端口定义

表格 11 Controller端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| Instr | I | 32 | 指令内容 |
| regSlt | O | 1 | 选择进入GRF的B的来源 0: rt 1: rd |
| regWE | O | 1 | GRF的写入使能信号 |
| dmWE | O | 1 | DM的写入使能信号 |
| extOp | O | 2 | EXT的选择信号 |
| aluOp | O | 2 | ALU的选择信号 |
| aluB | O | 1 | ALU的B端口来源 0: rt 1: 扩展后的立即数 |
| toReg | O | 1 | 存入寄存器数据来源 0: DM数据 1: ALU结果 |
| ifBeq | O | 1 | 判断是否为Beq指令 |
| rs | O | 5 | 读rs寄存器序号 |
| rt | O | 5 | 读rt寄存器序号 |
| rd | O | 5 | 读rd寄存器序号 |
| imm | O | 16 | 读立即数 |
| jcom | O | 1 | 判断是否为J指令 |

（3）真值表（instr略）（非严格 无x）

表格 12 Controller真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 端口 | addu | subu | ori | lw | sw | lui | beq | j |
| Instr  （略） | 000000  100001 | 000000  100011 | 001101 | 100011 | 101011 | 001111 | 000100 | 000010 |
| regSlt | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| regWE | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| dmWE | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| extOp0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| extOp1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| aluOp0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| aluOp1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| aluB | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| toReg | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| ifBeq | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| jcom | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

二、测试程序

（1）代码

test1：

.data

arr: .space 32

.text

ori $s0,10

ori $s7,$s0,10

loop:

beq $t0,$s0,loopout

lw $s1,arr($t1)

addu $t1,$t1,4

lw $s2,arr($t1)

addu $s3,$s1,$s7

addu $s2,$s2,$s3

sw $s2,arr($t1)

addu $t0,$t0,1

jal loop

loopout:

beq $t0,$t0,loopout

test2：

beq $t0,$t0,funcout

subu $31,$31,$31#清零

func:

lui $t0,1

jr $31

funcout:

addu $t1,$t1,$t0

ori $1,$1,1

jal func

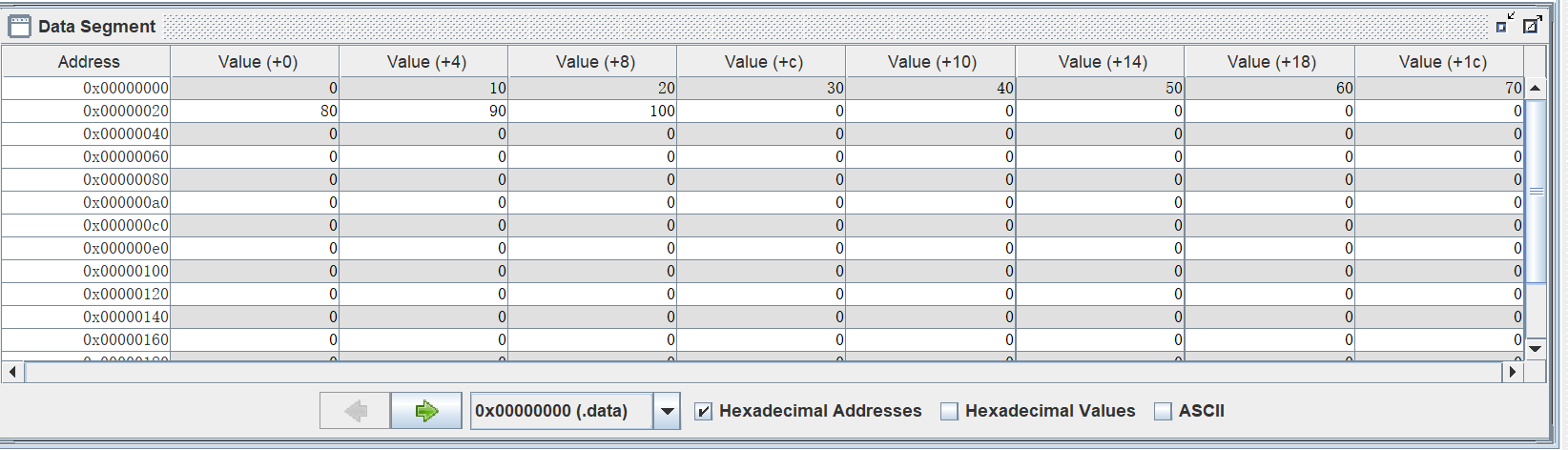
addu $t1,$t0,$t1

now:

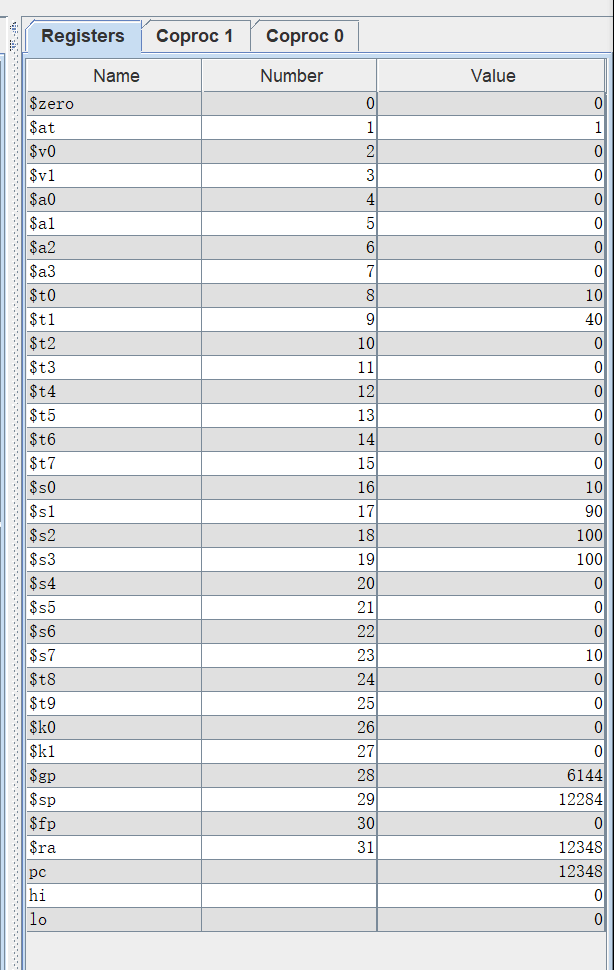
beq $t0,$t0,now

其中加了大量nop指令，已经全部删除，在此不全部展示。

（2）test1 MARS输出结果：



图表 1 MARS数据输出结果



图表 2 MARS寄存器运行结果

（2）test1 CPU运行结果：

@00003000: $16 <= 0000000a

@00003004: $23 <= 0000000a

@0000300c: $17 <= 00000000

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 00000004

@0000301c: $18 <= 00000000

@00003020: $19 <= 0000000a

@00003024: $18 <= 0000000a

@00003028: \*00000004 <= 0000000a

@0000302c: $ 1 <= 00000000

@00003030: $ 1 <= 00000001

@00003034: $ 8 <= 00000001

@00003038: $31 <= 0000303c

@0000300c: $17 <= 0000000a

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 00000008

@0000301c: $18 <= 00000000

@00003020: $19 <= 00000014

@00003024: $18 <= 00000014

@00003028: \*00000008 <= 00000014

@0000302c: $ 1 <= 00000000

@00003030: $ 1 <= 00000001

@00003034: $ 8 <= 00000002

@00003038: $31 <= 0000303c

@0000300c: $17 <= 00000014

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 0000000c

@0000301c: $18 <= 00000000

@00003020: $19 <= 0000001e

@00003024: $18 <= 0000001e

@00003028: \*0000000c <= 0000001e

@0000302c: $ 1 <= 00000000

@00003030: $ 1 <= 00000001

@00003034: $ 8 <= 00000003

@00003038: $31 <= 0000303c

@0000300c: $17 <= 0000001e

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 00000010

@0000301c: $18 <= 00000000

@00003020: $19 <= 00000028

@00003024: $18 <= 00000028

@00003028: \*00000010 <= 00000028

@0000302c: $ 1 <= 00000000

@00003030: $ 1 <= 00000001

@00003034: $ 8 <= 00000004

@00003038: $31 <= 0000303c

@0000300c: $17 <= 00000028

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 00000014

@0000301c: $18 <= 00000000

@00003020: $19 <= 00000032

@00003024: $18 <= 00000032

@00003028: \*00000014 <= 00000032

@0000302c: $ 1 <= 00000000

@00003030: $ 1 <= 00000001

@00003034: $ 8 <= 00000005

@00003038: $31 <= 0000303c

@0000300c: $17 <= 00000032

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 00000018

@0000301c: $18 <= 00000000

@00003020: $19 <= 0000003c

@00003024: $18 <= 0000003c

@00003028: \*00000018 <= 0000003c

@0000302c: $ 1 <= 00000000

@00003030: $ 1 <= 00000001

@00003034: $ 8 <= 00000006

@00003038: $31 <= 0000303c

@0000300c: $17 <= 0000003c

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 0000001c

@0000301c: $18 <= 00000000

@00003020: $19 <= 00000046

@00003024: $18 <= 00000046

@00003028: \*0000001c <= 00000046

@0000302c: $ 1 <= 00000000

@00003030: $ 1 <= 00000001

@00003034: $ 8 <= 00000007

@00003038: $31 <= 0000303c

@0000300c: $17 <= 00000046

@00003010: $ 1 <= 00000000

@00003014: $ 1 <= 00000004

@00003018: $ 9 <= 00000020

@0000301c: $18 <= 00000000

@00003020: $19 <= 00000050

符合预期。

三、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？  
   只输入了lw、sw存取地址所需的地址范围（扩展性不强，不能做lb、sb等指令）。来源是外面计算的address的11:2位。
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？  
   dm，grf，ifu。



因为要求清零。（含有存储PC、寄存器堆、数据的寄存器）

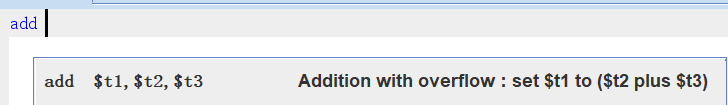
1. 列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

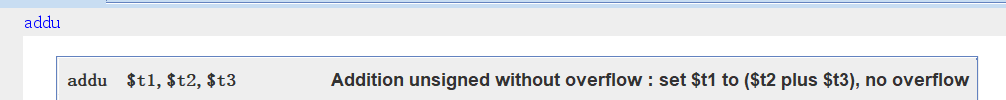
利用if-else完成操作码和控制信号的值之间的对应；利用assign语句完成操作码和控制信号的值之间的对应；用case。if(a==1)state=1;else if(a==2)state=2;else state=3;assign state=a==1?1:a==2?2:3; state=3;case(a)1:state=1;2:state=2;endcase

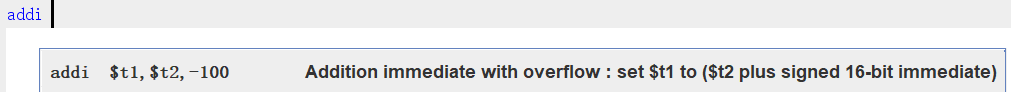
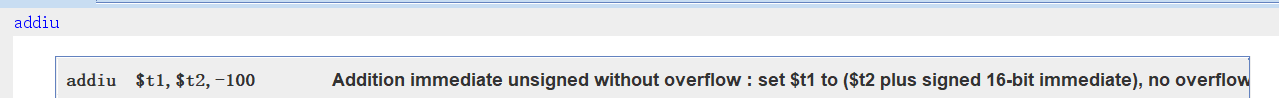
1. 根据你所列举的编码方式，说明他们的优缺点。

if/else case较清晰，三目运算符较简单，但遇到signed类型容易出错。

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。  
   这不是很显然的吗？





就差一个判断是否overflow。

1. 根据自己的设计说明单周期处理器的优缺点。

优点：易于完成；缺点：速度慢

1. 简要说明jal、jr和堆栈的关系。

jal将下一个pc存到寄存器中，jr通过访问寄存器中的值得到跳转地址，其中可以用堆栈对这些数据以及其他所需数据进行存储。