Logisim 单周期CPU设计文档

1. CPU设计方案综述

本CPU为logisim实现的单周期CPU（32位），支持的指令集包含**{addu, subu, ori, lw, sw, beq, lui, nop}，并进行了适当扩展。**

**该CPU采用层次化，模块化的设计，主要包含IFU，GRF，EXT，ALU，DM，Controller等模块。**

1. 关键模块定义
2. IFU
3. 基本描述

IFU内部主要包括PC，NPC，IM（容量32bit\*32，起始地址为0x00000000）以及相关逻辑。NPC中产生下一条指令的地址，当时钟上升沿到来时，PC更新指令地址并将其输出，IM根据地址输出对应指令

1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| Reset | I | 复位信号 |
| NPCOp[1:0] | I | 控制NPC进行相应的操作：  00：当前为顺序执行指令，NPC输出PC+4  01：当前指令为beq，作为决定是否跳转的条件之一  10：当前指令为jal，NPC输出PC31..28||instr\_index||02  11：当前指令为jr，NPC输出GRF[rs] |
| RA[31:0] | I | 将GRF[rs]的值输入IFU |
| Zero | I | 相等标志信号，判断ALU两操作数是否相等 |
| Instru[31:0] | O | 根据地址取到的当前指令 |
| PC4[31:0] | O | 输出PC+4作为地址 |
| imm[15:0] | O | 输出Instru[15:0] |

1. 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 | 根据当前PC的值从IM中取出相应的指令并通过Instru端口输出 |
| 3 | 输出PC+4 | 在PC4输出端口输出PC |
| 4 | 计算NPC | NPC根据NPCOp取值确定：  00：PC+4  01：PC+4+sign\_extend(offset||00)  （Zero为真时）  10：PC||instr\_index||00  11：GRF[rs] |
| 5 | 更新PC | 当时钟上升沿到来时，更新PC为NPC |

1. GRF
   1. 基本描述

GRF模块内部具有32个具有写使能和复位功能的寄存器，0号寄存器内的值始终为0。GRF支持同时读取两个寄存器的值以及写入一个寄存器的操作。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| Reset | I | 复位信号 |
| A1[4:0] | I | 地址输入信号，将对应地址寄存器的值输出至RD1 |
| A2[4:0] | I | 地址输入信号，将对应地址寄存器的值输出至RD2 |
| A3[4:0] | I | 地址输入信号，指定要进行写入的寄存器 |
| RFWr | I | 写使能信号 |
| WD[31:0] | I | 要写入寄存器的值 |
| RD1 | O | 数据输出信号，输出A1地址对应的寄存器的值 |
| RD2 | O | 数据输出信号，输出A2地址对应的寄存器的值 |

* 1. 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，GRF中的寄存器全部复位（初值设置为0x00000000） |
| 2 | 读取数据 | 读取A1和A2地址所对应寄存器的数据至RD1和RD2输出端口 |
| 3 | 写入数据 | 当时钟上升沿到来时，如果RFWr信号有效，则将WD输入端口的数据写入A3地址所对应的寄存器中 |

1. EXT
   1. 基本描述

EXT用于将16位立即数进行符号（无符号）扩展成32位并输出。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm16[15:0] | I | 数据输入信号，输入要进行扩展的16位立即数 |
| EXTOp | I | 控制信号：  0：无符号扩展  1：符号扩展 |
| Ext[31:0] | O | 数据输出信号，输出扩展完毕的数据 |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号扩展 | 将16位立即数无符号扩展为32位 |
| 2 | 符号扩展 | 将16位立即数符号扩展为32位 |

1. ALU
   1. 基本描述

ALU对输入的两个操作数（32bit）进行加、减、或、大小比较功能，输出运算的结果以及比较结果。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 数据输入信号，输入ALU的第一个操作数 |
| B[31:0] | I | 数据输入信号，输入ALU的第二个操作数 |
| ALUOp[1:0] | I | 控制信号：  00：A+B  01：A-B  10：A|B |
| C[31:0] | O | 数据输出信号，输出ALU的计算结果 |
| Zero | O | 数据输出信号，输出两操作数进行相等比较的结果 |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 加法 | 将两操作数相加 |
| 2 | 减法 | 将两操作数相减 |
| 3 | 或运算 | 将两操作数按位或 |
| 4 | 相等比较 | 判断两操作数是否相等，相等则Zero为真，反之为假 |

1. DM
   1. 基本描述

DM用于数据存储（容量为32bit\*32，起始地址为0x00000000）。DM支持复位功能，采用单向双端口设计。每当时钟上升沿到来时，如果写使能有效则能将数据写入对应地址，其余时间根据地址信号读出相应数据。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| Reset | I | 复位信号 |
| DMWr | I | 写使能信号 |
| A[31:0] | I | 地址信号，指定要进行操作的存储单元的地址 |
| WD[31:0] | I | 数据输入信号，输入要写入存储单元的数据 |
| RD[31:0] | O | 数据输出信号，输出地址对应的存储单元的数据 |

* 1. 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，每一个存储单元都被复位为0x00000000 |
| 2 | 读取 | 根据A地址信号输出对应存储单元的数据至RD输出端口 |
| 3 | 写入 | 当时钟上升沿到来时，如果DMWr有效，则根据A地址信号将WD输入端口中的数据写入对应的存储单元 |

1. 数据通路设计

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 部件 | IFU | GRF | | | | EXT | ALU | | DM | |
| 输入信号 | RA | A1 | A2 | A3 | WD | imm | A | B | A | WD |
| addu |  | IFU.Instr[25:21] | IFU.Instr[20:16] | IFU.Instr[15:11] | ALU.C |  | GRF.RD1 | GRF.RD2 |  |  |
| subu |  | IFU.Instr[25:21] | IFU.Instr[20:16] | IFU.Instr[15:11] | ALU.C |  | GRF.RD1 | GRF.RD2 |  |  |
| ori |  | IFU.Instr[25:21] |  | IFU.Instr[20:16] | ALU.C | IFU.Instr[15:0] | GRF.RD1 | GRF.RD2 |  |  |
| lw |  | IFU.Instr[25:21] |  | IFU.Instr[20:16] | DM.RD | IFU.Instr[15:0] | GRF.RD1 | EXT.Ext | ALU.C |  |
| sw |  | IFU.Instr[25:21] | IFU.Instr[20:16] |  |  | IFU.Instr[15:0] | GRF.RD1 | EXT.Ext | ALU.C | GRF.RD2 |
| beq |  | IFU.Instr[25:21] | IFU.Instr[20:16] |  |  |  | GRF.RD1 | GRF.RD2 |  |  |
| lui |  |  |  | IFU.Instr[20:16] | IFU.Instr[15:0]|| |  |  |  |  |  |
| jal |  |  |  | 0x1f | IFU.PC4 |  |  |  |  |  |
| jr | GRF.RD1 | IFU.Instr[25:21] |  |  |  |  |  |  |  |  |
| nop |  | | | | | | | | | |
| 综合 | RA | IFU.Instr[25:21] | IFU.Instr[20:16] | IFU.Instr[15:11]  IFU.Instr[20:16]  0x1f | ALU.C,DM.RD,IFU.PC4,  IFU.Instr[15:0]|| | IFU.Instr[15:0] | GRF.RD1 | GRF.RD2  EXT.Ext | ALU.C | GRF.RD2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输出 | 0端口 | 1端口 | 2端口 | 3端口 |
| GRF.A3 | IFU.Instr[15:11] | IFU.Instr[20:16] | 0x1f |  |
| GRF.WD | ALU.C | DM.D | IFU.PC4 | IFU.Imm32 |
| ALU.B | GRF.RD2 | EXT.Ext |  |  |

1. 控制器设计
   1. 基本思路

通过指令的opcode和funct产生数据通路所需要的控制信号，具体操作为先通过与阵列得到指令变量，再通过或阵列得到各控制信号的取值。

* 1. 真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | NPCOp[1:0] | GRFWr | EXTOp | ALUOp[1:0] | DMWr | A3Sel[1:0] | WDSel[1:0] | BSel |
| addu  (000000/100001) | 00 | 1 | x | 00 | 0 | 00 | 00 | 0 |
| subu  (000000/100011) | 00 | 1 | x | 01 | 0 | 00 | 00 | 0 |
| ori  (001101) | 00 | 1 | 0 | 10 | 0 | 01 | 00 | 1 |
| lw  (100011) | 00 | 1 | 1 | 00 | 0 | 01 | 01 | 1 |
| sw  (101011) | 00 | 0 | 1 | 00 | 1 | xx | xx | 1 |
| beq  (000100) | 01 | 0 | x | 01 | 0 | xx | xx | 0 |
| jal  (000011) | 10 | 1 | x | xx | 0 | 10 | 10 | x |
| jr  (000000/001000) | 11 | 0 | x | xx | 0 | xx | xx | x |
| lui  (001111) | 00 | 1 | x | xx | 0 | 01 | 11 | x |

1. 测试方案
   1. 测试代码

.data

a: .word 1:32

.text

ori $t0,$t0,7 #$t0=7

ori $t1,$t1,15 #t1=15

ori $t3,$t3,4 #t3=2

ori $t5,$t5,1 #t5=1

nop

ori $t6,$t6,1

beq $t5,$t6,next

ori $t0,$t0,6

ori $t1,$t1,6

ori $t3,$t3,6

next:

addu $t1,$t0,$t0 #t1=14

subu $t2,$t1,$t0 #t2=7

lw $t4,a($t3)

sw $t1,a($t3)

jal test

addu $s0,$s0,$t5

subu $s0,$s0,$t5

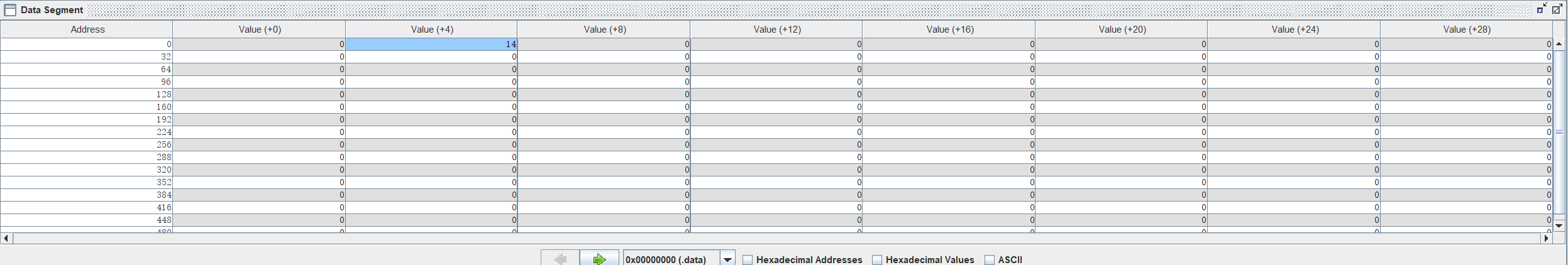
test:

lui $s0,1

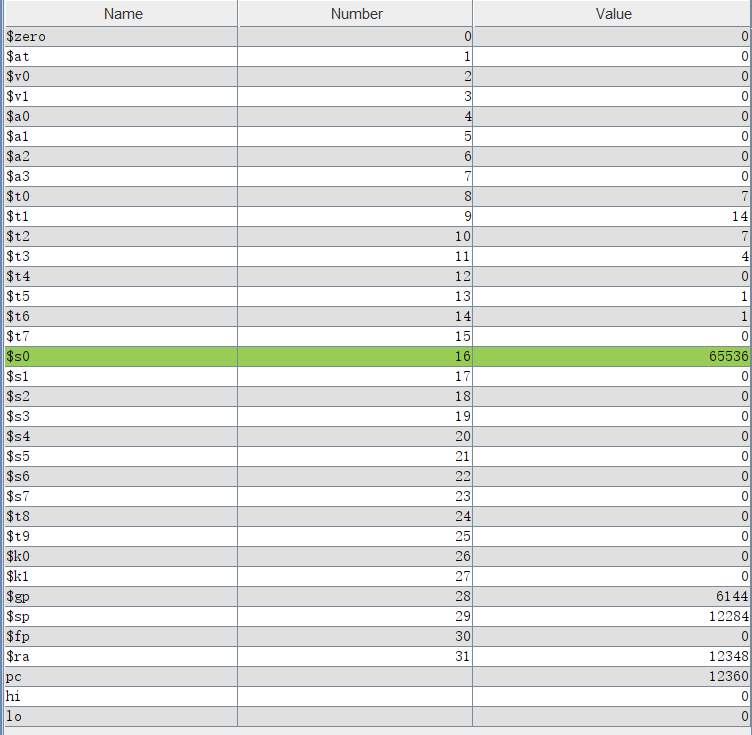
jr $ra

* 1. 期望结果

DM:

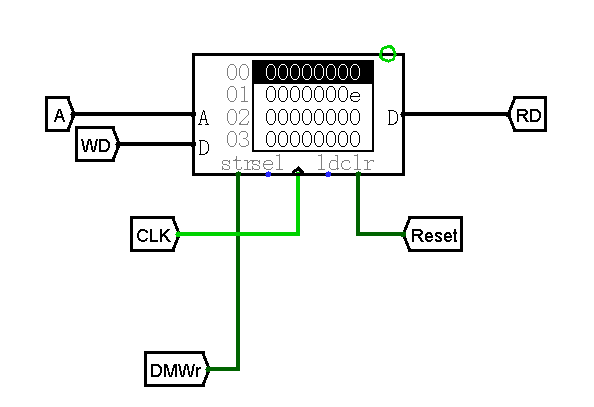


GRF:

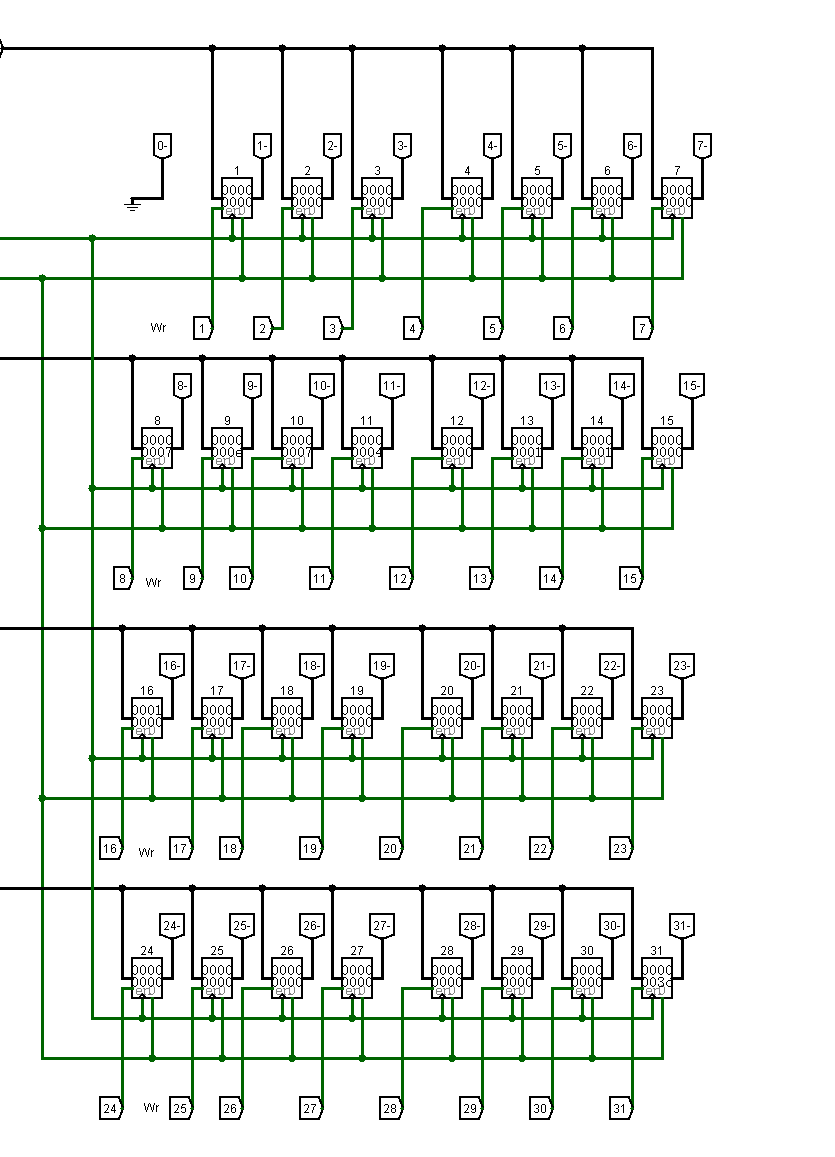


本CPU中$gp,$sp中应为0

* 1. 测试结果

DM：

GRF：



1. 思考题
   1. 现在我们的模块中IM使用ROM， DM使用RAM， GRF用Register，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

合理。

IM需要导入指令并且无需修改，只会同时读取一个存储单元；DM需要对存储单元进行读写的操作，同时也只会写或者读一个存储单元，因此IM用ROM，DM用RAM（读写分离、单向双端口）是合理的；GRF要搭建一个寄存器堆，用寄存器来完成也是合理的。

* 1. 事实上，实现nop空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

nop的机械码为0x00000000，在与阵列中不对应任何一个指令变量，所有控制信号都无效，实际上只执行了PC的更新。

* 1. 上文提到，MARS不能导出PC与DM起始地址均为0的机器码。实际上，可以通过为DM增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

可以将高位地址信号与相应位数的起始地址比较大小，将大于等于的真值作为DM的片选信号，并将低位地址接入DM的地址输入信号。

由于本CPU中的DM实际上仅使用7位地址，故地址从0开始和从0x00003000开始实际上没有区别，故无需进行改造。

* 1. 除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式验证。形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)”了解相关内容后，简要阐述相比于测试，形式验证的优劣之处。

优点：速度快，能够覆盖所有情况，无需开发测试模块；

缺点：无法模拟出元件的物理特性和延迟。