利用Logisim开发MIPS单周期处理器

一、整体结构：

GRF（通用寄存器组，Registers）、控制器（Control）、ALU（算术逻辑单元）、DataMemory（数据存储器）、signEXT（位扩展器）、PC和InstructionMemory组成（取指令单元）和ALU Control（ALU控制器）组成

处理器为32位处理器

处理器应支持的指令集为：{addu,subu,ori,lw,sw,beq,lui,nop}。

二、模块规格：

（一）模块规格撰写

1．PC（程序计数器）

器件：32bit寄存器

加法器 由于通常情况下nPC=PC+4也为了美观，直接把加法集成进去了。

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| CLR | I | 复位信号  1：有效  0：无效 |
| PCin[31:0] | I | nPC |
| PCnow[31:0] | O | 当前PC值 |
| PC+4[31:0] | O | PC值加4 |

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被设置为起始地址0x00000000 |
| 2 | 更新PC | 根据PCin在时钟上升沿时候更新为下一周期正确的PC |

2．IM（指令存储器）

器件：32\*32bitROM

因为ROM中储存了32个地址，且IM实际地址宽度仅为5位，

而且我发现PC总是4个4个加0000下一位是0100下一位是1000后两位是不变的零，而且Logism里面的ROM却是1个1个加0000下一位是0001下一位是0010 教程中有使用30位pc的办法，我后来发现不太直观，于是就采用的是将PC[31：0]中[6:2] 6 5 4 3 2 位作为ROM的地址，取指令。然后分了一下

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| program counter address[31：0] | I | program counter address |
| Op[31:26] | O | 6bitOp |
| rs[25:21] | O | 5bitrs |
| Rt[20:16] | O | 5bit rt |
| Rd[15:11] | O | 5bit rd |
| Shamet[10:6] | O | 5bit shamet |
| Funct[5:0] | O | 6bit funct |
| Imm[15:0] | O | 16bit [immediate](javascript:;) [operand](javascript:;) or offset |
| Instr[31:0] | O | 32位MIPS指令 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取指令 | 根据PC从IM中取出指令 |

2.GRF（通用寄存器组）：内部包括32个寄存器

具有写使能的寄存器实现，寄存器总数为 32 个

**0 号寄存器**的值始终保持为 0。其他寄存器初始值均为 0

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| WD | I | 读写控制信号  1：写操作  0：不可写 |
| CLK | I | 时钟信号 |
| CLR | I | 复位信号  1：有效  0：无效 |
| A1 | I | 读寄存器1的地址 |
| A2 | I | 读寄存器2的地址 |
| A3 | I | 写寄存器的地址 |
| RD1 | O | 32位输出1 |
| RD2 | O | 32位输出2 |

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，所有寄存器的值被设置为0x00000000 |
| 2 | 写寄存器 | 根据输入的写寄存器地址，把输入的数据写入写寄存器中 |
| 3 | 读寄存器 | 根据输入的读寄存器地址，将数据读出 |

3.ALU（算术逻辑单元）

提供32位加、减、或运算

可以不支持溢出

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | ALU32位输入数据A |
| B[31:0] | I | ALU32位输入数据B |
| ALUOp[1:0] | I | ALU功能选择信号  00:加法  01:减法  10:或运算 |
| Out | O | 32位数据输出 |
| zero | O | Beq的时候用到1代表相等 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 或 | A|B |
| 2 | 减 | A-B |
| 3 | 加 | A+B |
| 4 | 等于 | Beq的时候比较两个reg相等输出1 |

4.DM（数据存储器）

使用RAM实现，容量为32bit\*32,采用双端口模式

起始地址：0x00000000

RAM 应使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| CLR | I | 复位信号  1：有效  0：无效 |
| Address[31:0] | I | 需访问的32位RAM地址 按照IM同理的方法连接MemAddr |
| MemAddr[4:0] | O | 没有什么用只是为了与需要检测的5位Memaddr连接 |
| WData[31:0] | I | 写入数据的输入 |
| RData | O | 读取数据的输出 |
| ReadEn | I | 可能有用还未开发 |
| WriteEn | I | 读写控制信号  1：可以进行写操作 |

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，所有数据被设置为0x00000000 |
| 2 | 写操作 | 根据输入的寄存器地址，把输入的数据写入 |
| 3 | 读操作 | 根据输入的寄存器地址，将其中的数据读出 |
| 4 | 32-5的转换 | 为了正确输出MemAddr |

1. signEXT（位扩展器）：

使用logisim内置的Bit Extender

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Immin[15:0] | I | 16位imm数据输入 |
| Result[31:0] | O | 位扩展后的32位输出 |
| ExtOp[1:0] | I | 位扩展选择信号  00:高位补0（0扩展）  01：高位符号扩展  10：低位补0（0扩展左移16位） |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 高位补0 | 高16位补0 |
| 2 | 低位补0 | 低16位补0 |
| 3 | 符号扩展 | 若符号位为0，则高位补0  若符号位为1，则高位补1 |

6.控制器（Control）

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| op[5:0] | I | 6位op |
| RegDst | O | 写寄存器地址 0是rt 1是rd |
| ALUsrc | O | ALU的B操作数 0是grf中值 1是立即数 |
| Memtoreg | O | 写寄存器的数据来源0是alu结果1是dm |
| RegWrite | O | GRF中的writeEn |
| MemWrite | O | DM中的writeEn |
| PCsrc | O | nextPC选择信号 |
| Extop | O | 控制扩展方式 |
| ALUOp[1:0] | O | 控制ALU进行相应运算 |
| Need\_ALUC | O | 是R型指令的话需要ALUcontrol进一步判断 |
| Needshift | O | 还在开发中，目前没用到是否需要4移位 |

8.ALU控制器（ALUControl）

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| func[5:0] | I | 6位function |
| ALUOp[1:0] | O | 控制ALU进行相应运算 |

（二）思考题

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

答：合理，无改进意见；

IM为指令寄存器，以取指令为主，指令一旦写入便不再改变；

DM为数据寄存器，既存又取，需要能够随时写入与读出；

GRF为通用寄存器组，应该由32个寄存器组成，可存可取。

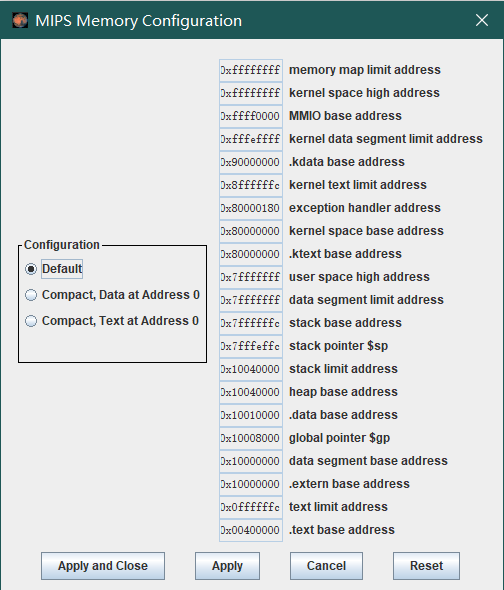
1. 事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？

因为所有位都为0时，我发现搭出来所有的输出都为零，输了PC+4外没有任何状态改变。

3. 上文提到，MARS 不能导出 PC 与 DM 起始地址均为 00 的机器码。实际上，可以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。

只要PC DM 起始位置岔开的很大 后几位都是0 尤其是[6:0]中都是0 那跟0x00000000没啥区别的。

而且可以修改.text base address 和.data base address

4.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 | 缺点 |
| 仿真测试 | 操作速度快 | 初始设置时间长，debug较为困难，验证周期长 |
| 形式验证 | 易于发现细节处的bug | 所需数学知识多，分布小规模，仅在验证小模块时较为实用 |

仿真有时候是瞎猫碰死耗子 容易测，难debug全，形式验证需要整体思考把问题想明白。