Logisim单周期CPU设计文档

(一)CPU设计要求

32位单周期CPU

支持指令{add，sub，lw，sw，lui，beq，nop}

(二)关键模块设计

ALU

信号名	方向	描述
SrcA[31:0]	I	32位输入信号，第一个操作数A
SrcB[31:0]	I	32位输入信号，第二个操作数B
ALUControl[2:0]	I	3位输入信号，选择ALU的功能 000 : SrcA + SrcB 001 : SrcA - SrcB 010 : A | B 011 : A & B 100 : A>>B 101 : $signed(A)>>>B 110 : A<B 置1
Equal	O	1位输出信号，标志A,B是否相等 A=B置Equal=1
Less	O	1位输出信号，标志A是否小于B A<B置Less=1
ALUResult[31:0]	O	32位输出信号，输出运算结果

GRF

模块定义

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：不能向GRF中写入数据

信号名	方向	描述
A1[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的值读出到RD1
A2[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的值读出到RD2
A3[4:0]	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
WD[31:0]	I	32位数据输入信号
RD1[31:0]	O	输出A1指定的寄存器的32位数据
RD2[31:0]	O	输出A2指定寄存器中的32位数据

功能定义

序号	功能名称	描述
1	复位	reset信号有效是，所有寄存器存储的数值清零
2	读数据	读出A1，A2地址对应寄存器中所存储的数据到RD1，RD2
3	写数据	当WE有效且时钟上升沿来临时，将WD写入A3对应的寄存器中

DM

模块定义

信号名	方向	描述
clk	I	时钟信号
Reset	I	复位信号，将RAM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：不能向GRF中写入数据
WD[31:0]	I	32位数据输入信号，要写入的数据
A[31:0]	I	32位输入信号，指定RAM中的的一个地址
DMop[1:0]	I	2位输入信号，用于lb，lh等特殊指令 00： 正常读写，lw，sw 01： 用于lh和sh，根据WE选择进行哪一条指令 10： 用于lb和sb，根据WE选择进行哪一条指令
RD[31:0]	O	32位输出信号，读出A指定的地址中的数据

功能定义

序号	功能名称	描述

序号	功能名称	描述
1	复位	reset信号有效是，RAM所有地址存储的数值清零
2	读数据	读出A指定的地址的所存储的数据到RD
3	写数据	当WE有效且时钟上升沿来临时，将WD写入A对应的地址中

EXT

信号名	方向	描述
Imm[15:0]	I	15位输入立即数
EXTop	I	功能选择信号 0：Imm无符号拓展到32位 1：Imm符号位拓展到32位
EXTImm[31:0]	O	32位输出信号，输出Imm拓展之后的数

NPC

模块定义

信号名	方向	描述
PC[31:0]	I	32位输入信号，当前指令的地址
NPCop[1:0]	I	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31\cdots28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
instr_index[26:0]	I	26位输入信号，用于PC的计算
offset[16:0]	I	16位输入信号，PC的偏移量
Reg[32:0]	I	32位输入信号，用于寄存器地址的跳转
Branchop[2:0]	I	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal
equal	I	用于指示ALU两端数据是否SrcA=SrcB
n_equal	I	用于指示ALU两端数据是否SrcA!=SrcB
less	I	用于指示ALU两端数据是否SrcA<SrcB

信号名	方向	描述
big	I	用于指示ALU两端数据是否SrcA>rcB
less or equal	I	用于指示ALU两端数据是否SrcA<=SrcB
big or equal	I	用于指示ALU两端数据是否SrcA>=SrcB
NPC[31:0]	O	32位输出，输出下一条指令的地址

功能定义

序号	功能名称	描述
1	计算PC的下一个值	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31…28}$||instr_index||0_2 10： 结合equal,n_equal,less,big,big or equal,less or equal判断是否需要跳转 PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]

IFU

模块定义

信号名	方向	描述
clk	I	时钟信号
Reset	I	异步复位信号
NPCop[1:0]	I	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31…28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
instr_index[26:0]	I	26位输入信号，用于PC的计算
offset[16:0]	I	16位输入信号，PC的偏移量
Reg[32:0]	I	32位输入信号，用于寄存器地址的跳转
Branchop[2:0]	I	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal

信号名	方向	描述
equal	I	用于指示ALU两端数据是否SrcA=SrcB
less	I	用于指示ALU两端数据是否SrcA<SrcB

信号名	方向	描述
Instr[31:0]	O	32为输出信号，输出当前要执行的指令
PC[31:0]	O	32位输出信号，当前PC的地址

CTRL

模块定义

信号名	方向	描述
OP[5:0]	I	6位输入信号，指令操作码
Func[6:0]	I	6位输入信号，指令的func段
RegDst[1:0]	O	GRFA3输入端控制信号 00 : A3←$Instr_{20...16}$ 01 : A3←$Instr_{15...11}$ 11 : A3←0x1f
RegWrite	O	寄存器写入控制信号 0 : 不能向GRF写入 1 : 可以向GRF写入
EXTop	O	功能选择信号 0：Imm无符号拓展到32位 1：Imm符号位拓展到32位
ALUsrc[1:0]	O	ALUSrcB输入控制信号 00 : SrcB←RD2 01 : SrcB←EXTImm 10 : SrcB←sll指令的s SrcA←RD2
ALUctrl[2:0]	O	3位输出信号，选择ALU的功能 000 : SrcA + SrcB 001 : SrcA - SrcB 010 : A |B 011 : A & B 100 : A>>B 101 : $signed(A)>>>B 110 : A<B 置1 111 : A<<B

信号名	方向	描述
Menwrite	O	内存写入控制信号 0 : 不能向DM写入 1 : 可以向DM写入
MemtoReg[1:0]	O	控制向寄存器的写入数据 00 : WD←ALUResult 01 : WD←RD 10 : WD←$NPC_{31...0}$ 11 : WD←[$ALUResult_{15...0}$||0_{16}]

信号名	方向	描述
NPCop[1:0]	O	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31...28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
Branchop[2:0]	O	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal
DMop[1:0]	O	用于lb,sb,lh,sh等操作的拓展 00 : 正常读取，以字为单位 01 : 用于lh或sh Menwrite = 1--->sh Menwrite = 0--->lh 11 : 用于lb或sb Menwrite = 1--->sb Menwrite = 0--->lb

相关数据通路信号说明

RegDst

控制对于A3的输入信号，RegDst=00，A3←$Instr_{16...20}$

，RegDst=01，A3←$Instr_{11...15}$

，RegDst=10，A3←0x1f，用于jal指令

ALUsrc

控制对于SrcA的输入信号，ALUsrc=10，SrcA←RD2，用于sll指令

，ALUsrc=other，SrcA←RD1

控制对于SrcB的输入信号，ALUsrc=00，SrcB←RD2

，ALUsrc=01，SrcB←EXTImm

，ALUsrc=10，SrcB←$Instr_{6...10}$(shamt)

MemtoReg

控制对于寄存器的写入信号，MemtoReg=00，WD←ALUResult

，MemtoReg=01，WD←RD

，MemtoReg=10，WD←PC+4

，MemtoReg=11，WD←$ALUResult_{15...0}||0_{16}$，用于lui指令

指令控制信号

Instr	RegDst[1:0]	Regwrite	EXTop	ALUsrc[1:0]	ALUctrl[2:0]	Memwrite	MemtoReg[1:0]	NPCop[1:0]	Branchop[2:0]	DMop[1:0]
add	01	1	x	00	000	0	xx	00	xxx	00
sub	01	1	x	00	001	0	xx	00	xxx	00
ori	00	1	0	01	010	0	xx	00	xxx	00
lw	00	1	1	01	000	0	01	00	xxx	00
sw	xx	0	1	01	000	1	xx	00	xxx	00
beq	xx	0	x	00	xxx	x	xx	10	000	00
lui	00	1	0	01	000	x	11	00	xxx	00

(三)思考题

1.上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能。

解答 单周期CPU中，IFU，GRF，DM模块起着状态存储的功能，它们存储了每一个状态的指令以及一些数据CTRL，ALU是组合逻辑电路，起着状态转移的功能。

2.现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用 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。

解答 我认为是合理的，对于IM所存储的指令，由于在程序运行中不需要对指令进行修改，使用只读的ROM可以保证指令的读但不会被修改。DM是可读也可写的所以要使用可以读写的RAM，而且由于内存的空间比较大，使用寄存器会导致成本急剧增大，使用满足功能需求同时控制成本的RAM是合理的。GRF由于需要高速的读取或者存储，并且由于寄存器的数目较少，使用寄存器阵列也是合理的。

3.在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路。

解答 我设计了NPC模块，用于计算在各种指令中对于下一个状态PC的计算。我使用了NPCop控制PC在j，jr，beq，bne，ble，blt，bgt等指令中对于PC的控制，我添加了判断ALU两端数据大小的信号以此控制bne等需判断指令的跳转条件判断

NPC

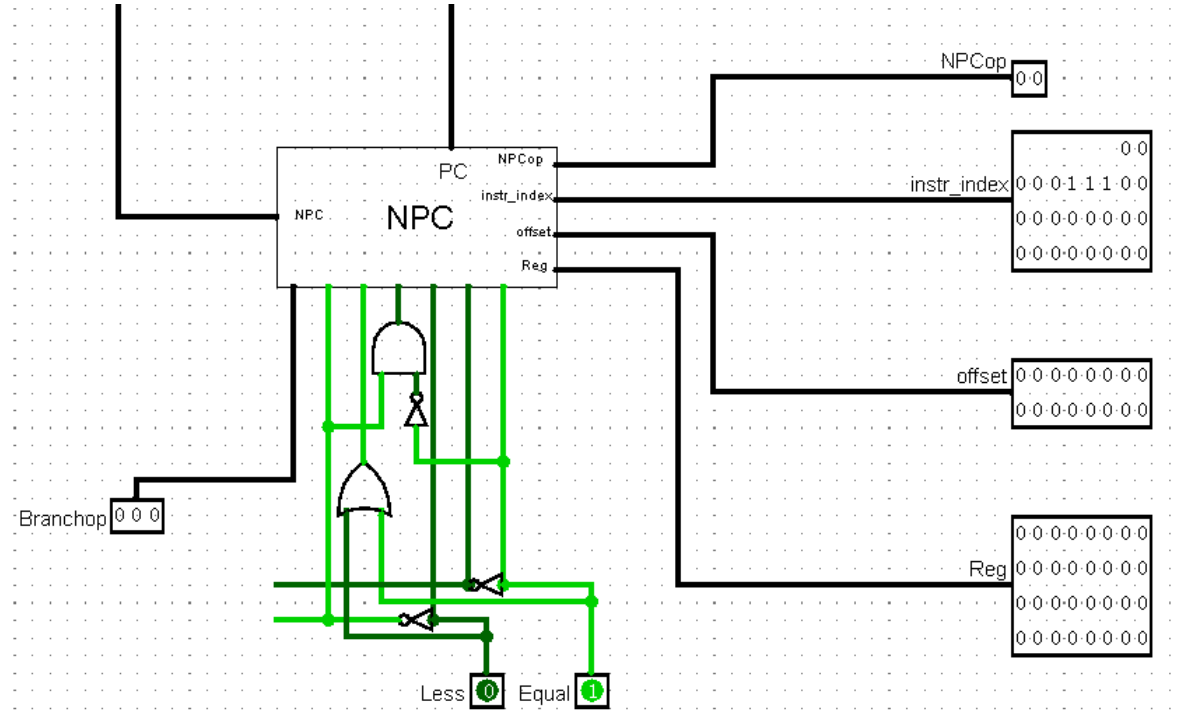
模块定义

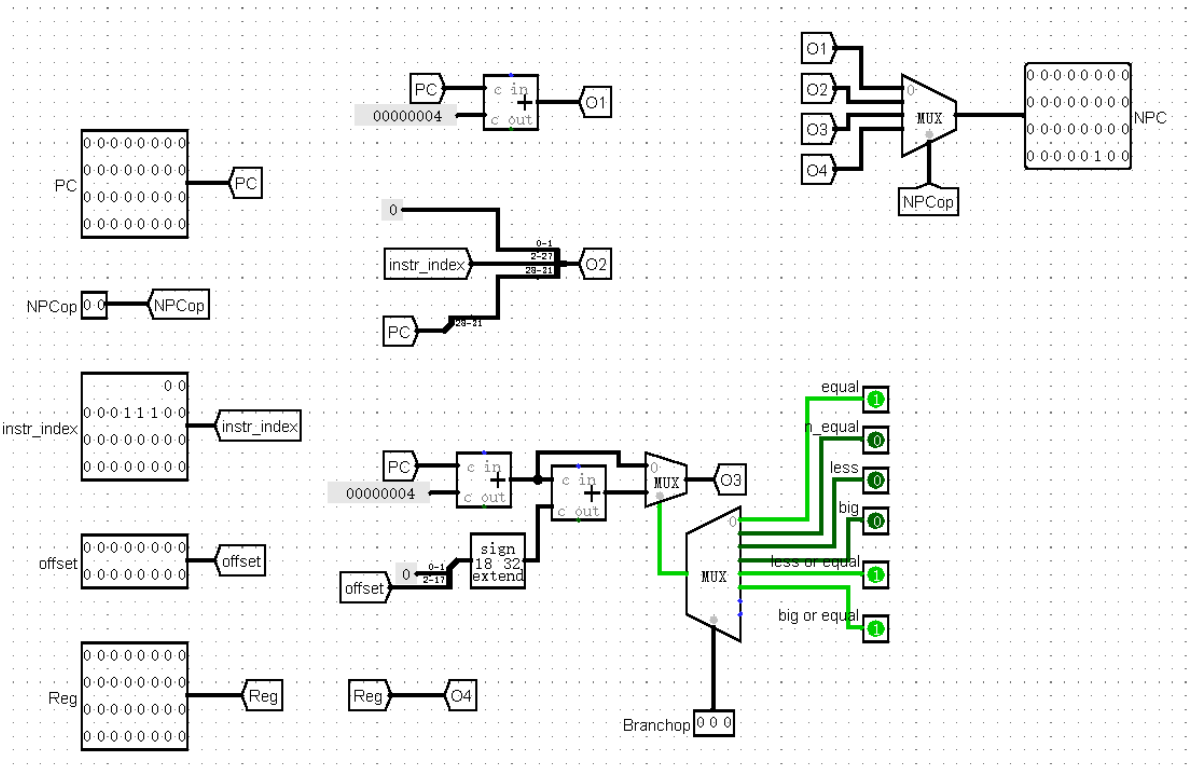
信号名	方向	描述
PC[31:0]	I	32位输入信号，当前指令的地址
NPCop[1:0]	I	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31\cdots28}$||instr_index||0_2 10：PC←PC + 4 + sign_extend(offset||0_2) 11：PC←GPR[rs]
instr_index[26:0]	I	26位输入信号，用于PC的计算
offset[16:0]	I	16位输入信号，PC的偏移量
Reg[32:0]	I	32位输入信号，用于寄存器地址的跳转
Branchop[2:0]	I	用于指示进行何种跳转判断 000 : 判断equal 001 : 判断n_equal 010 : 判断less 011 : 判断big 100 : 判断less or equal 101 : 判断big or equal
equal	I	用于指示ALU两端数据是否SrcA=SrcB
n_equal	I	用于指示ALU两端数据是否SrcA!=SrcB
less	I	用于指示ALU两端数据是否SrcA<SrcB

信号名	方向	描述
big	I	用于指示ALU两端数据是否SrcA>rcB
less or equal	I	用于指示ALU两端数据是否SrcA<=SrcB
big or equal	I	用于指示ALU两端数据是否SrcA>=SrcB
NPC[31:0]	O	32位输出，输出下一条指令的地址

功能定义

序号	功能名称	描述
1	计算PC的下一个值	两位控制信号，控制NPC的的值 00：PC←PC+4 01：PC←$PC_{31\cdots28}$||instr_index||0_2 10： 结合equal,n_equal,less,big,big or equal,less or equal判断是否需要跳转 11：PC←GPR[rs]





4.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?

解答 对于`nop`指令，只是执行了一个PC=PC+4的行为，其他地方没有改变，将指令加进去和不加进去没有什么改变。同时`sll $0,$0,0`的机器码就是`0x00000000`可以作为一个空循环的指令使用。

5.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。

解答 如果寄存器中存储的DM的地址被映射在`0x3000_0000`到`0x3fff_ffff`间，而我们的DM起始地址是0，那么，我们可以将输入地址用一个减法器直接减去0x3000_0000，再作为DM的地址输入。

6.阅读 Pre 的 “MIPS 指令集及汇编语言” 一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处。

对于ori指令，还可以增加一些对于$0的赋值，检查是否会对$0造成修改。

对于add指令，还可以增加一些正+零，负+零的指令。

对于sw，最好对每一个内存空间都进行赋值，以保证正确性。

可以对于每一个寄存器都进行赋值，以检查寄存器的连接正确情况。