Logisim搭建单周期CPU 设计报告

1. 模块规格

1、IFU（取指令单元）

IFU是取指令单元，包含PC（程序计数器）、IM（指令存储器）和相关逻辑。IFU根据当前指令计算出计算出下一条指令的地址并将其取出，使程序完成顺序执行或跳转

表格 1 IFU端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 说明 |
| nPC\_sel | I | 是否为转移指令的判断信号。（该信号根据当前指令的opcode产生）  0：不是转移指令 1：是转移指令 |
| zero | I | 该转移指令是否需要转移的判断信号（该信号根据ALU判断busA与busB是否相等产生）。  0：不需要转移（busA!=busB）  1：需要转移（busA==busB） |
| imm16[15:0] | I | 转移指令的指令移位量，左移两位并符号扩展后为在IM中的指令偏移量 |
| clr | I | 同步复位信号 |
| Instr[31:0] | O | 当前被取出的指令 |

表格 2 IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条指令的地址 | nPC\_sel&zero=0时，PC←PC+4存入PC。  nPC\_sel&zero=1时，  PC←PC+4+(sign\_ext(imm16)||00)。 |
| 2 | 取指令 | 从IM中取出地址为PC的指令（注：由于PC以字节为单位、位宽为32，但IM中的ROM以字为单位、位宽为5，故只取PC的2-6位作为指令的地址） |
| 3 | 复位 | clr=1时，PC被设置为0x00000000。 |

2、GRF（寄存器堆）

GRF由32个寄存器构成（其中0号寄存器恒为0）。主要包含读和写操作：读操作时，传入RA、RB两个寄存器编号，将对应寄存器的值读出到busA、busB。写操作时，在RegWr信号为1时，将busW写入RW编号对应的寄存器中。

表格 3 GRF端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 说明 |
| RA[4:0] | I | 待读出数据1的寄存器编号($rs的编号)。 |
| RB[4:0] | I | 待读出数据2的寄存器编号($rt的编号)。 |
| RW[4:0] | I | 待写入寄存器的编号($rd或$rt的编号) |
| busW[31:0] | I | 待写入寄存器的数据 |
| RegWr | I | 寄存器写使能信号  0：不写入 1：写入 |
| Clk | I | 时钟信号 |
| busA[31:0] | O | 读出数据1($rs的值) |
| busB[31:0] | O | 读出数据2($rt的值) |

表格 4 GRF功能定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 功能描述 | |
| 1 | 读寄存器 | 将编号为RA的寄存器中的数据输出到busA端口；  将编号为RB的寄存器中的数据输出到busB端口 |
| 2 | 写寄存器 | RegWr=1时，在时钟上升沿将busW写入到编号为RW的寄存器中 |

3、ALU（算术逻辑单元）

ALU由何种算数逻辑组成。根据ALUctr对ALUin1和ALUin2进行加、减、或、相等比较等操作并输出。

表格 5 ALU端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | | 说明 |
| ALUin1[31:0] | I | 第一个待操作数据。 | |
| ALUin2[31:0] | I | 第二个待操作数据。 | |
| ALUctr[1:0] | I | 进行何种运算的选择信号。  00：in1+in2  01：in1-in2  10：in1 | in2  11：in1<<in2 | |
| ALUout[31:0] | O | 输出运算后的结果。 | |
| zero | O | 输出ALUin1与ALUin2的相等判断结果。  0：ALUin1!=ALUin2 1：ALUin1==AlUin2 | |

表格 6 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加（无溢出） | ALUout=ALUin1+ALUin2 |
| 2 | 减（无溢出） | ALUout=ALUin1-ALUin2 |
| 3 | 或 | ALUout=ALUin1 | ALUin2 |
| 4 | 移位 | ALUout=ALUin1<<ALUin2 |
| 5 | 相等判断 | zero=(ALUin1==ALUin2) |

4、DM（数据存储器）

表格 7 DM端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 说明 |
| DMaddr[31:0] | I | DM中的读出/写入地址。 |
| DMdata[31:0] | I | 待写入DM的数据。 |
| MemWr | I | 将DMdata写入DM的写使能信号。  0：不写入 1：写入 |
| DMout[31:0] | O | 输出DM中DMaddr地址中的数据。 |

表格 8 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读出 | MemWr=0时，DMout=DM中DMaddr地址中的数据。 |
| 2 | 写入 | MemWR=1时，将DMdata写入DM的DMaddr地址中。 |

5、EXT（扩展器）

EXT根据ExtOp对16位立即数imm16进行各类扩展。

表格 9 EXT端口说明

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 说明 |
| imm[15:0] | I | 待扩展的数据。 |
| ExtOp[1:0] | I | 进行何种扩展的选择信号。  00：无符号扩展  01：有符号扩展  10：加载到高16位，低16位补0  11：（未定义） |
| ExtOut[31:0] | O | 扩展后的数据。 |

表格 10 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号扩展 | ExtOp=00时，对imm16进行无符号扩展并输出到ExtOut。 |
| 2 | 有符号扩展 | ExtOp=01时，对imm16进行有符号扩展并输出到ExtOut。 |
| 3 | 后补16位0 | ExtOp=10时，将imm16加载到高16位，在低16位补0，并输出到ExtOut。 |

6、电路图总览

计算机生成了可选文字:
BegPstrd
RegWr
rtnPCsel
CLK
Instr
Fetch
Unit
zero
RWRARBbusA
busW
RegFiIe
busB
32
0
1
ALUctr
1
MemtoReg
MemWr
0
CLK
imm16
ExtOp
0
0
32
Data|n
CLK
WrEnAddr
Data
Memory
1
ALUSrc

图 1 CPU主要部件示意图

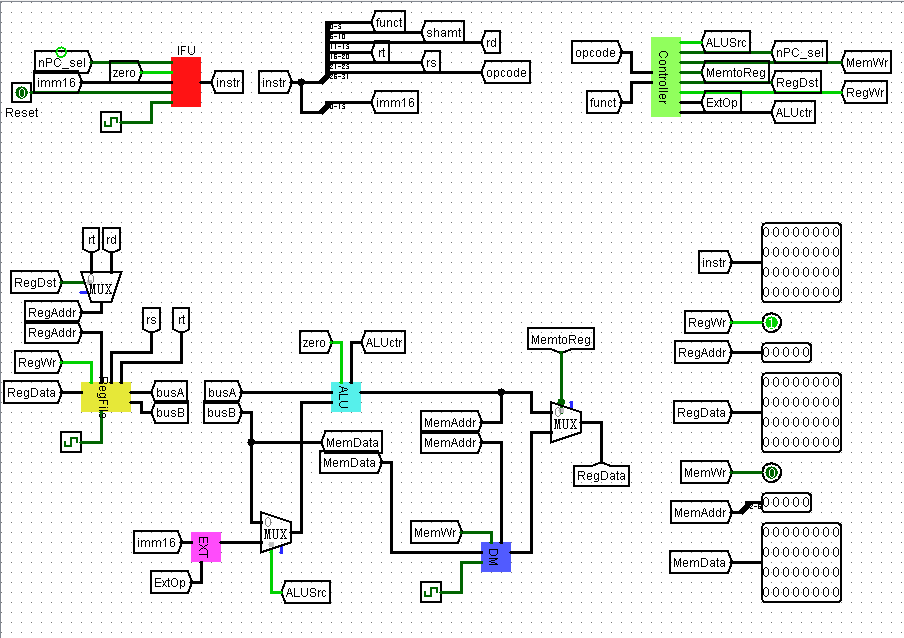


图 2 CPU主要部件示意图（Logisim）

7、思考题

**（1）若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。**

答：

30位PC可理解为以字为单位对指令进行读取，顺序执行时每周期自加1。

优势：①因为每条指令占据4字节，故0-1位恒为00，30位PC可以去掉没有意义的两位，节省寄存器；②跳转指令可以直接将imm16符号扩展后与PC+4相加，而不需要先在低2位补0。

劣势：①由于MIPS中寄存器、内存等都是32位的，30位的PC和这些部件兼容性不好，需要特别在低2位补0。

**（2）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。IM在一个程序中应该是提前写好，写好后固定不变，只需要读出不需要写入的，只读存储器ROM符合IM的要求。DM用于存储数据，对速度要求不高，但需要有较大的存储空间，价格较低的RAM符合DM的要求。GRF用于临时存储数据并承担指令的读出写入工作，需要较快的速度，高速的寄存器符号GRF的要求。

1. Controller（控制器）设计

1、基本描述

Controller根据指令中的opcode段和funct段，先利用与门确定该指令类型，再利用或门确定各控制信号。

2、控制信号真值表

表格 11 控制信号真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 | \ | | | | | |
| opcode | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000000 |
|  | addu | subu | ori | lw | sw | beq | lui | nop |
| ExtOp[1:0] | xx | xx | 00 | 01 | 01 | xx | 10 | xx |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 | x |
| ALUCtr[1:0] | 00 | 01 | 10 | 00 | 00 | xx | 00 | xx |
| nPC\_sel | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| MemWr | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| MemtoReg | 0 | 0 | 0 | 1 | x | x | 0 | x |
| RegDst | 1 | 1 | 0 | 0 | x | x | 0 | x |
| RegWr | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |

3、控制信号含义

表格 12 控制信号含义1

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| ALUSrc | ALUin2=busB | ALUin2=ExtOut |
| nPC\_sel | 当前指令不是跳转指令 | 当前指令是跳转指令 |
| MemWr | 不可以向DM中写入数据 | 可以向DM中写入数据 |
| MemtoReg | busW=ALUOut | busW=DMOut |
| RegDst | RW=rt | RW=rd |
| RegWr | 不可以向GRF中写入数据 | 可以向GRF中写入数据 |

表格 13 控制信号含义2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 10 | 11 |
| ExtOp | 无符号扩展 | 有符号扩展 | 低位补16位0 | （未定义） |
| ALUCtr | 加（无符号） | 减（无符号） | 或 | 逻辑左移 |

4、思考题

**（1）结合上文给出的样例真值表，给出RegDst， ALUSrc， MemtoReg，RegWrite, nPC\_Sel, ExtOp与op和func有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）**

答：

ExtOp[1]=lui

ExtOp[0]=lw+sw

ALUSrc=ori+lw+sw+lui

ALUCtr[1]=ori

ALUCtr[0]=subu

nPC\_sel=beq

MemWr=sw

MemtoReg=lw

RegDst=addu+subu

RegWr=addu+subu+ori+lw+lui

其中：

addu=op[5]’\*op[4]’\*op[3]’\*op[2]’\*op[1]’\*op[0]’\*func[5]\*func[4]’\*func[3]’\*func[2]’\*func[1]’\*func[0]

subu=op[5]’\*op[4]’\*op[3]’\*op[2]’\*op[1]’\*op[0]’\*func[5]\*func[4]’\*func[3]’\*func[2]’\*func[1]\*func[0]

ori= op[5]’\*op[4]’\*op[3]\*op[2]\*op[1]’\*op[0]

lw= op[5]\*op[4]’\*op[3]’\*op[2]’\*op[1]\*op[0]

sw= op[5]\*op[4]’\*op[3]\*op[2]’\*op[1]\*op[0]

beq= op[5]’\*op[4]’\*op[3]’\*op[2]\*op[1]’\*op[0]’

lui= op[5]’\*op[4]’\*op[3]\*op[2]\*op[1]\*op[0]

**（2）充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式， 请给出化简后的形式。**

答：

ExtOp[1]= op[5]’\*op[4]’\*op[3]\*op[2]\*op[1]\*op[0]

ExtOp[0]= op[5]\*op[4]’ \*op[2]’\*op[1]\*op[0] (lw+sw)

ALUSrc= op[5]’\*op[4]’\*op[3]\*op[2] \*op[0] (ori+lui)

+ op[5]\*op[4]’ \*op[2]’\*op[1]\*op[0]

ALUCtr[1]= op[5]’\*op[4]’\*op[3]\*op[2]\*op[1]’\*op[0]

ALUCtr[0]=op[5]’\*op[4]’\*op[3]’\*op[2]’\*op[1]’\*op[0]’\*func[5]\*func[4]’\*func[3]’\*func[2]’\*func[1]\*func[0]

nPC\_sel= op[5]’\*op[4]’\*op[3]’\*op[2]\*op[1]’\*op[0]’

MemWr= op[5]\*op[4]’\*op[3]\*op[2]’\*op[1]\*op[0]

MemtoReg= op[5]\*op[4]’\*op[3]’\*op[2]’\*op[1]\*op[0]

RegDst=op[5]’\*op[4]’\*op[3]’\*op[2]’\*op[1]’\*op[0]’\*func[5]\*func[4]’\*func[3]’\*func[2]’ \*func[0]

RegWr=op[5]’\*op[4]’\*op[3]’\*op[2]’\*op[1]’\*op[0]’\*func[5]\*func[4]’\*func[3]’\*func[2]’ \*func[0]

+op[5]’\*op[4]’\*op[3]\*op[2] \*op[0]+ +op[5]\*op[4]’\*op[3]’\*op[2]’\*op[1]\*op[0]

**（3）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。**

答：

Nop指令不对电路进行任何操作，故可理解为控制信号均为0，而控制信号根据指令类型利用或门构建，A+0=A，所以无需在真值表中加入它。

1. 测试CPU

1、测试代码及其测试期望

#ori

ori $0, $0, 123

ori $a0, $0, 123 #$4=123

ori $a1, $a0, 456 #$5=507

#lui

lui $a2, 123 #$6=8060928

lui $a3, 0xffff

ori $a3, $a3, 0xffff #$7=0xffffffff=-1

#addu

addu $s0, $a0, $a2 #$16=8061051 ++

addu $s1, $a0, $a3 #$17=122 +-

addu $s2, $a3, $a3 #$18=-2 --

#subu

subu $s3, $a0, $a2 #$19=-8060805 ++

subu $s4, $a0, $a3 #$20=124 +-

subu $s5, $a3, $a3 #$21=0 --

#sw

ori $t0, $0, 0x0000

sw $a0, 0($t0)

sw $a1, 4($t0)

sw $a2, 8($t0)

sw $a3, 12($t0)

sw $s0, 16($t0)

sw $s1, 20($t0)

sw $s2, 24($t0)

#lw

lw $s0, 24($t0) #$16=$18=-2

lw $s2, 16($t0) #$18=$16=8061051

#beq

ori $s6, $0, 123 #$22=123

beq $a0, $a1, loop1 #if($4==$5)

#0x0000001c=507 0x00000020=123

beq $a0, $s6, loop2 #if($4==$22)

#0x0000001c=0 0x00000020=123

loop1:sw $a1, 28($t0)

loop2:sw $s6, 32($t0)

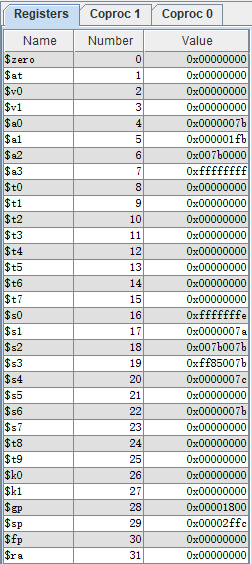


图 3 寄存器测试期望

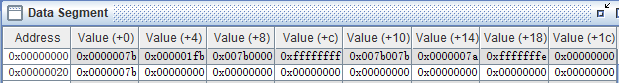


图 4 内存测试期望

2、CPU(Logisim)测试结果

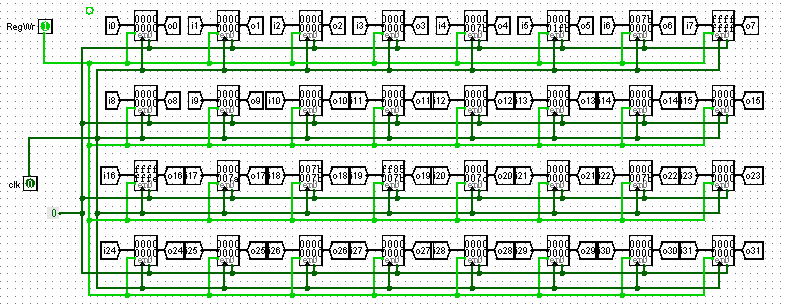


图 5 寄存器测试结果

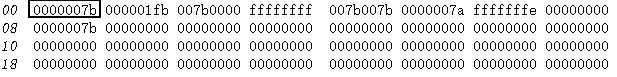


图 6 内存测试结果

3、思考题

**（1）前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。**

答：

片选，也称选片。在有很多芯片挂在同一总线上，而我们只需要对其中特定的某个芯片进行读写时，可以通过地址的比对，利用与门、或门、非门的组合产生片选信号CS（chip select）或SS(slave select)，决定对哪个芯片进行操作。

假设本次设计中MARS的Memory Configuration选择为Compact, Text at Address 0，即.data从0x00002000开始，就可以加入片选信号，将地址前20位与0x00002比较，相同时则对DM操作。

假设每个DM有256MB容量，并且映射在在0x30000000~0x3fffffff之间。将地址前4位与0x3进行比较，相同时则对当前DM操作。但本次设计中，DM最多存储到0x00000080，因此不需要使用片选信号。

**（2）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，简要阐述相比与测试，形式验证的优劣。**

答：

优势：

1. 形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了100%。
2. 形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试激励，测试者不需要考虑如何获得测试向量。
3. 形式验证可以进行从系统级到门级的验证，验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

劣势：

随着硬件设计复杂性的增加，需要对比的模型呈爆炸性增长，这使形式验证更多也只是停留在模块级别，面对大规模硬件设计的时候往往会比较吃力。