P3　Logisim 单周期 CPU 设计文档

一、　整体结构

本文档所描述的处理器为32位单周期处理器，采用Logisim实现。该处理器支持的指令集为{addu, subu, ori, lw, sw, beq, lui, nop}，并进行了适当扩展。

该处理器采用模块化和层次化设计，包含 Controller（控制器）、IFU（取指令单元）、GRF（通用寄存器组）、ALU（算术逻辑单元）、DM（数据存储器）、 EXT（位扩展器）等基本部件。处理器采用内置时钟信号，顶层有效驱动信号有且仅有 reset。

二、　模块规格

1. IFU（取指令单元）

1) 基本描述

IFU 内部包括 PC（程序计数器）、IM（指令存储器，容量为 32bit*32，起始地址为 0x00000000）及相关逻辑。IFU 输出当前指令和下一条指令地址并根据相应的控制信号在时钟上升沿更新 PC 的值。

2) 端口说明

表 1 IFU 端口说明

信号名	方向	描述
Clk	I	时钟信号。
Reset	I	复位信号（高电平有效）。
Beq	I	标志当前指令是否为 beq 指令（高电平有效）。
Bne	I	标志当前指令是否为 bne 指令（高电平有效）。
Blez	I	标志当前指令是否为 blez 指令（高电平有效）。
Bgtz	I	标志当前指令是否为 bgtz 指令（高电平有效）。
Bltz	I	标志当前指令是否为 bltz 指令（高电平有效）。
Bgez	I	标志当前指令是否为 bgez 指令（高电平有效）。
J	I	标志当前指令是否为 j 或 jal 指令（高电平有效）。
Jr	I	标志当前指令是否为 jr 或 jalr 指令（高电平有效）。
Equal	I	相等标志信号（高电平有效），用于标记 ALU 的两操作数是否相等，作为分支条件。
LTZ	I	小于 0 标志信号（高电平有效），用于标记 ALU 的第一个操作数是否小于 0。
EQZ	I	等于 0 标志信号（高电平有效），用于标记 ALU 的第一个操作数是否等于 0。
Imm32[31:0]	I	输入 32 位立即数，用于计算分支或跳转后 PC 的值。
Rs[31:0]	I	输入 rs 寄存器的值，用于计算跳转后 PC 的值。
Instr[31:0]	O	输出当前指令。
PC4[31:0]	O	输出 PC+4。

3) 功能定义

表 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	根据当前 PC 的值从 IM 取出相应的指令并由 Instr 输出端输出。
3	输出 PC+4	在 PC4 输出 PC+4。
4	计算下一条指令地址	当 Beq 和 Equal 都为真时，下一条指令地址计算为 PC+4+Imm32[29:0]||0^2； 当 Bne 为真，Equal 为假时，下一条指令地址计算为 PC+4+Imm32[29:0]||0^2； 当 Blez 为真，LTZ 和 EQZ 中至少有一个为真时，下一条指令地址计算为 PC+4+Imm32[29:0]||0^2； 当 Bgtz 为真，LTZ 和 EQZ 均为假时，下一条指令地址计算为 PC+4+Imm32[29:0]||0^2； 当 Bltz 和 LTZ 都为真时，下一条指令地址计算为 PC+4+Imm32[29:0]||0^2； 当 Bgez 为真，LTZ 为假时，下一条指令地址计算为 PC+4+Imm32[29:0]||0^2； 当 J 为真时，下一条指令地址计算为(PC+4)[31:28]||Imm32[25:0]||0^2； 当 Jr 为真时，下一条指令地址计算为 Rs； 否则，下一条指令地址计算为 PC+4。
5	更新 PC	当时钟上升沿到来时，更新 PC 为计算出的下一条指令地址。

2. GRF（通用寄存器组）

1) 基本描述

GRF 内部包括 32 个具有写使能和复位功能的寄存器。其中，0 号寄存器的值始终保持为 0，其他寄存器初始值均为 0。GRF 提供同时读取 2 个寄存器和写入 1 个寄存器的功能。

2) 端口说明

表 3 GRF 端口说明

信号名	方向	描述
Clk	I	时钟信号。
Reset	I	复位信号（高电平有效）。
A1[4:0]	I	地址输入信号 1，将其对应寄存器中存储的数据输出至 RD1。
A2[4:0]	I	地址输入信号 2，将其对应寄存器中存储的数据输出至 RD2。
A3[4:0]	I	地址输入信号 3，指定写入操作所对应的寄存器。
WD[31:0]	I	数据输入信号，即要写入寄存器中的数据。
RegWrite	I	写使能信号（高电平有效）。
RD1[31:0]	O	数据输出信号，输出 A1 对应寄存器中的 32 位数据。
RD2[31:0]	O	数据输出信号，输出 A2 对应寄存器中的 32 位数据。

3) 功能定义

表 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，GRF 中的每一个寄存器都被设置为 0x00000000。
2	读取数据	读取 A1 和 A2 所对应寄存器中的数据至 RD1 和 RD2。
3	写入数据	当时钟上升沿到来时，如果 RegWrite 有效，就将 WD 写入 A3 所对应的寄存器中。

3. ALU（算术逻辑单元）

1) 基本描述

ALU 对输入的两个操作数提供 32 位加、减、或、与、或非、异或、移位运算、大小比较功能和与 0 的大小比较功能，输出运算结果以及相应的比较标志。

2) 端口说明

表 5 ALU 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[3:0]	I	指定 ALU 所要进行的操作： 0000：A+B； 0001：A-B； 0010：A|B； 0011：A&B； 0100：~(A|B)； 0101：A^B； 0110：A<<B[4:0]； 0111：A>>B[4:0]； 1000：A>>>B[4:0]； 1001：(A<B)?1:0。
Result[31:0]	O	数据输出信号，输出 ALU 的计算结果。
Equal	O	相等标志信号（高电平有效），标志两操作数是否相等。
LTZ	O	小于 0 标志信号（高电平有效），标志第一个操作数是否小于 0。
EQZ	O	等于 0 标志信号（高电平有效），标志第一个操作数是否等于 0。

3) 功能定义

表 6 ALU 功能定义

序号	功能名称	功能描述
1	加法	当 ALUOp 为 0000 时，Result 输出 A+B 的值。
2	减法	当 ALUOp 为 0001 时，Result 输出 A-B 的值。
3	或运算	当 ALUOp 为 0010 时，Result 输出 A|B 的值。
4	与运算	当 ALUOp 为 0011 时，Result 输出 A&B 的值。
5	或非运算	当 ALUOp 为 0100 时，Result 输出~(A|B)的值。
6	异或运算	当 ALUOp 为 0101 时，Result 输出 A^B 的值。
7	逻辑左移运算	当 ALUOp 为 0110 时，Result 输出 A<<B[4:0]的值。
8	逻辑右移运算	当 ALUOp 为 0111 时，Result 输出 A>>B[4:0]的值。
9	算术右移运算	当 ALUOp 为 1000 时，Result 输出 A>>>B[4:0]的值。
10	小于比较运算	当 ALUOp 为 1001 时，若 A<B，则 Result 输出 1；否则 Result 输出 0。
11	相等比较运算	当 ALUOp 为 xxx1 时，若 A=B，则 Equal 信号有效；否则无效。
12	零比较运算	若 A<0，则 LTZ 信号有效；否则无效。 若 A=0，则 EQZ 信号有效；否则无效。

4. DM（数据存储器）

1) 基本描述

DM 用于存储数据，其容量为 32bit*32，起始地址为 0x00000000。DM 支持复位功能，并且数据读取和写入端口分离。

2) 端口说明

表 7 DM 端口说明

信号名	方向	描述
Clk	I	时钟信号。
Reset	I	复位信号（高电平有效）。
Addr[31:0]	I	地址信号，指定要操作的存储单元的地址。
WD[31:0]	I	数据输入信号，输入要写入到 Addr 所对应的存储单元的数据。

MemWrite	I	写使能信号（高电平有效）。
RD[31:0]	O	数据输出信号，输出 Addr 所对应的存储单元的数据。

3) 功能定义

表 8 DM 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，DM 中的每一个存储单元都被设置为 0x00000000。
2	读取	RD 输出 Addr 所对应的存储单元的数据。
3	写入	当时钟上升沿到来时，如果 MemWrite 有效，就将 WD 写入 Addr 所对应的存储单元中。

5. EXT

1) 基本描述

EXT 用于将输入的 16 位立即数符号扩展或无符号扩展成 32 位并输出。

2) 端口说明

表 9 EXT 端口说明

信号名	方向	描述
Imm16[15:0]	I	数据输入信号，输入要进行扩展的数据。
SignExt	I	符号扩展信号（高电平有效）。
Imm32[31:0]	O	数据输出信号，输出扩展后的数据。

3) 功能定义

表 10 EXT 功能定义

序号	功能名称	功能描述
1	无符号扩展	当符号扩展信号无效时，将 Imm16 无符号扩展至 32 位并输出至 Imm32。
2	符号扩展	当符号扩展信号有效时，将 Imm16 符号扩展至 32 位并输出至 Imm32。

三、 数据通路设计

表 11 数据通路

部件	IFU		GRF				EXT	ALU		DM	
输入信号	Imm32	Rs	A1	A2	A3	WD	Imm16	A	B	Addr	WD
addu			IFU.Instr[25:21]	IFU.Instr[20:16]	IFU.Instr[15:11]	ALU.Result		GRF.RD1	GRF.RD2		
subu			IFU.Instr[25:21]	IFU.Instr[20:16]	IFU.Instr[15:11]	ALU.Result		GRF.RD1	GRF.RD2		
ori			IFU.Instr[25:21]		IFU.Instr[20:16]	ALU.Result	IFU.Instr[15:0]	GRF.RD1	EXT.Imm32		
lw			IFU.Instr[25:21]		IFU.Instr[20:16]	DM.RD	IFU.Instr[15:0]	GRF.RD1	EXT.Imm32	ALU.Result	
sw			IFU.Instr[25:21]	IFU.Instr[20:16]			IFU.Instr[15:0]	GRF.RD1	EXT.Imm32	ALU.Result	GRF.RD2
beq	EXT.Imm32		IFU.Instr[25:21]	IFU.Instr[20:16]			IFU.Instr[15:0]	GRF.RD1	GRF.RD2		
lui					IFU.Instr[	IFU.Instr[1					

					20:16]	5:0]||0^{16}					
nop											
综合	EXT.Imm32		IFU.Instr[25:21]	IFU.Instr[20:16]	IFU.Instr[20:16] IFU.Instr[15:11]	ALU.Result DM.RD IFU.Instr[15:0]||0^{16}	IFU.Instr[15:0]	GRF.RD1	GRF.RD2 EXT.Imm32	ALU.Result	GRF.RD2

表 12 MUX 连接关系

名称	输入 0	输入 1	输入 2	输出
RegDst	IFU.Instr[20:16]	IFU.Instr[15:11]		GRF.A3
RegSrc	ALU.Result	DM.RD	IFU.Instr[15:0]||0^{16}	GRF.WD
ALUSrc	GRF.RD2	EXT.Imm32		ALU.B

四、 控制器（Controller）设计

1. 基本描述

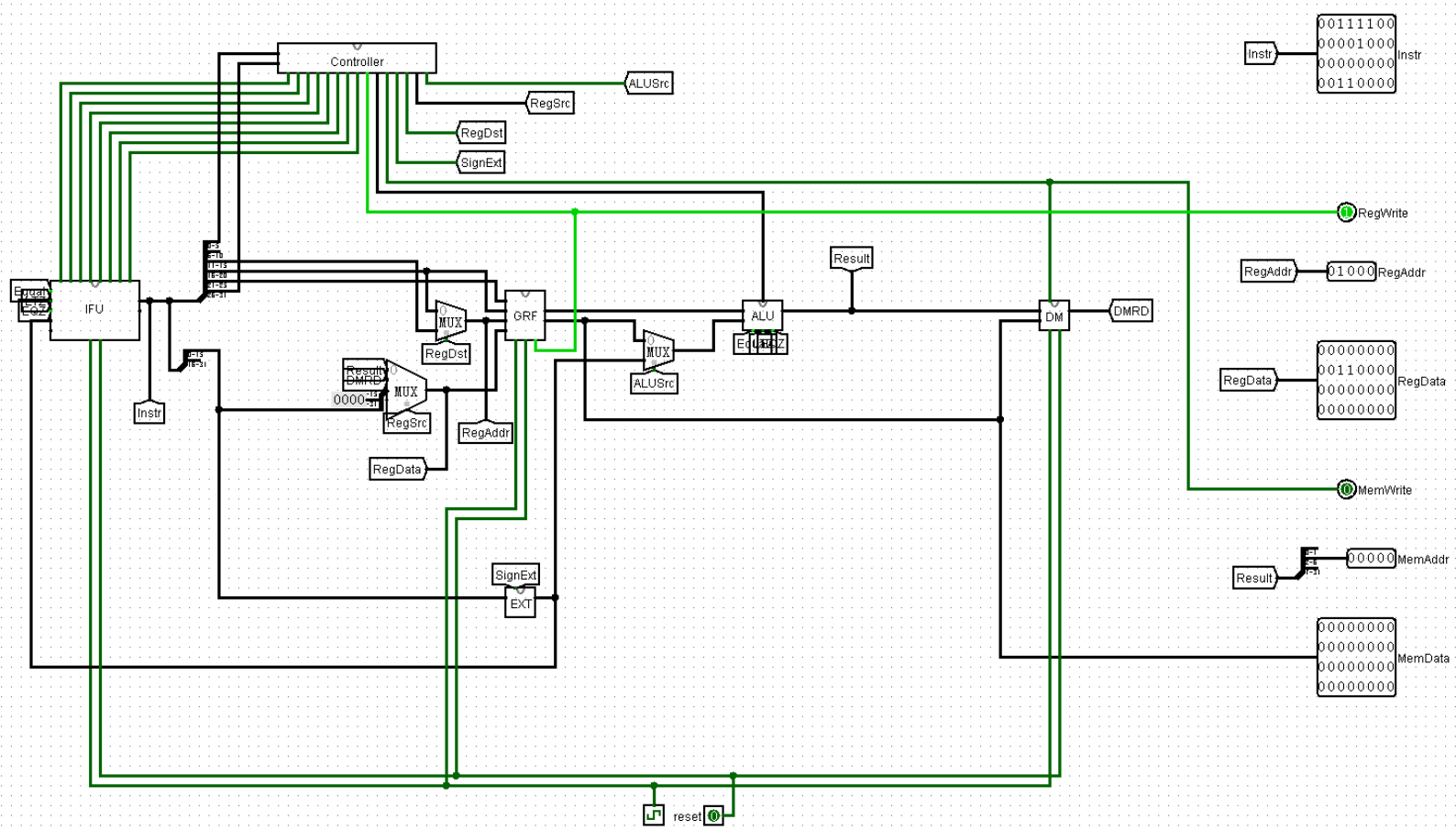
控制器通过输入的 Op 和 Funct 信号产生数据通路所需的控制信号，通过与或逻辑实现。

2. 真值表

表 13 控制器真值表

指令 (Op/Funct)	Beq	Bne	Blez	Bgtz	Bltz	Bgez	J	Jr	RegWrite	ALUOp[3:0]	MemWrite	SignExt	RegDst	RegSrc[1:0]	ALUSrc
addu (000000/100001)	0	0	0	0	0	0	0	0	1	0000	0	x	1	00	0
subu (000000/100011)	0	0	0	0	0	0	0	0	1	0001	0	x	1	00	0
ori (001101)	0	0	0	0	0	0	0	0	1	0010	0	0	0	00	1
lw (100011)	0	0	0	0	0	0	0	0	1	0000	0	1	0	01	1
sw (101011)	0	0	0	0	0	0	0	0	0	0000	1	1	x	xx	1
beq (000100)	1	0	0	0	0	0	0	0	0	xxx1	0	1	x	xx	0
lui (001111)	0	0	0	0	0	0	0	0	1	xxxx	0	x	0	10	x
nop (000000)	0	0	0	0	0	0	0	0	0	xxxx	0	x	x	xx	x

五、 顶层设计图示



六、 CPU 测试

1. 测试程序

```
lui $t0, 49
ori $t0, 18
ori $s0, $0, 16
sw $t0, 4($0)
lw $t1, -12($s0)
sw $t0, -8($s0)
lw $t2, 8($0)
addu $t3, $t1, $0
subu $t4, $0, $t1
Label:
beq $t2, $0, Skip
lui $t5, 1
beq $t0, $0, Label
beq $t1, $t2, Skip
```

```
lui $t6, 1
Skip:
nop
```

2. 期望结果

GRF：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00310012
$t1	9	0x00310012
$t2	10	0x00310012
$t3	11	0x00310012
$t4	12	0xffceffee
$t5	13	0x00010000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000010
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000303c
hi		0x00000000
lo		0x00000000

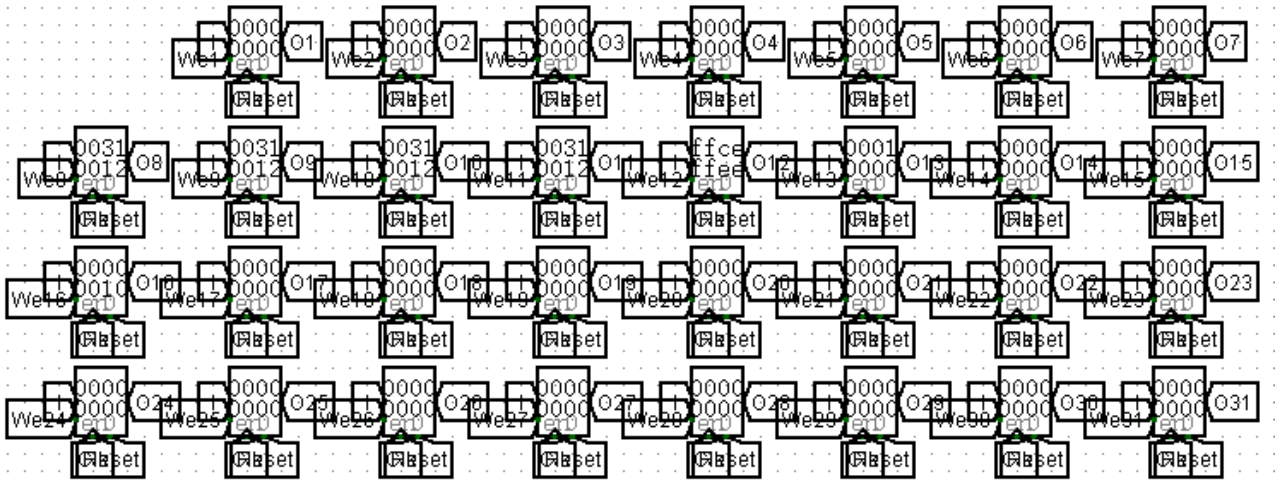
注：在本文档所描述的 CPU 中，$gp 和$sp 的值应为 0。

DM：

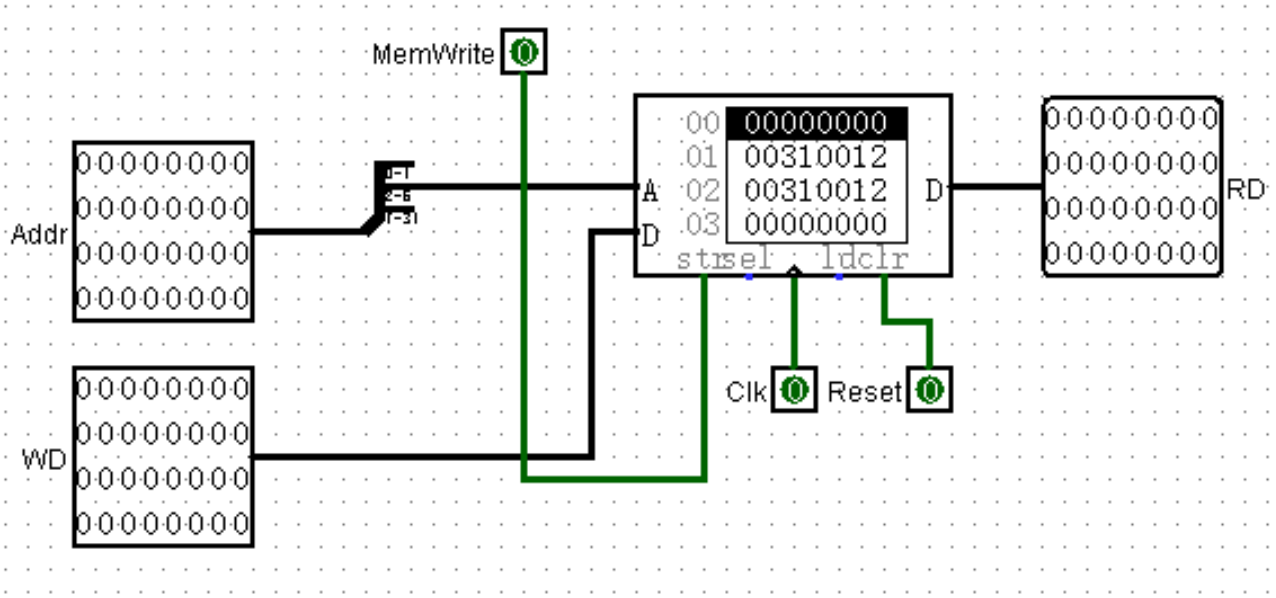
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x00310012	0x00310012	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

3. 测试结果

GRF：



DM：


MemWrite
Addr
WD
A
D
00 00000000
01 00310012
02 00310012
03 00000000
stmsel
ldclr
D
RD
Clk
Reset

思考题

一、 L0.T2

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

30 位 PC 可以直接将输出连接到 IM 的地址输入端口，且下一条指令只需加 1 而不是加 4，更加自然，而且方便了分支指令和立即数跳转指令对下一条指令地址的计算（无需在低位拼接 0）。但是对于寄存器跳转指令，会导致需要丢弃寄存器低 2 位的麻烦；对于 jal 和 jalr 指令，存入寄存器的值也需要在低位补两位 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。

IM 需要导入指令且无需修改，只会同时读取 1 个存储单元；DM 需要对存储的数据进行读写操作，且只能同时操作 1 个存储单元；GRF 需要对存储的数据进行读写操作，且可以同时读取 2 个存储单元和写入 1 个存储单元。由此，IM 使用 ROM、DM 使用 RAM 是合理的；由于需要同时对多个寄存器操作，GRF 使用寄存器也是合理的。

二、 L0.T3

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。）

RegDst = (~op5 ~op4 ~op3 ~op2 ~op1 ~op0)(func5 ~func4 ~func3 ~func2 ~func1 ~func0 + func5 ~func4 ~func3 ~func2 func1 ~func0)

ALUSrc = (~op5 ~op4 op3 op2 ~op1 op0) + (op5 ~op4 ~op3 ~op2 op1 op0) + (op5 ~op4 op3 ~op2 op1 op0)

MemtoReg = op5 ~op4 ~op3 ~op2 op1 op0

RegWrite = (~op5 + op4 + ~op3 + op2 + ~op1 + ~op0)(op5 + op4 + op3 + ~op2 + op1 + op0)

MemWrite = op5 ~op4 op3 ~op2 op1 op0

nPC_Sel = ~op5 ~op4 ~op3 op2 ~op1 ~op0

ExtOp = op5 + op4 + ~op3 + ~op2 + op1 + ~op0

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。

RegDst = ~op5 ~op4 ~op3 ~op2 ~op1 ~op0 func5 ~func4 ~func3 ~func2 ~func0

ALUSrc = (~op5 ~op4 op3 op2 ~op1 op0) + (op5 ~op4 ~op2 op1 op0)

MemtoReg = op5 ~op4 ~op3 ~op2 op1 op0

RegWrite = (~op5 + op4 + ~op3 + op2 + ~op1 + ~op0)(op5 + op4 + op3 + ~op2 + op1 + op0)

MemWrite = op5 ~op4 op3 ~op2 op1 op0

nPC_Sel = ~op5 ~op4 ~op3 op2 ~op1 ~op0

ExtOp = op5 + op4 + ~op3 + ~op2 + op1 + ~op0

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。

   nop 指令不对应与逻辑的任何一个最小项，因此或逻辑的所有输出均为无效，此时可以满足 nop 指令的需求。

三、 L0.T4

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。

   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 0x00002000 开始实际上没有区别，故无需进行改造。

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。

   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。