CPU 设计文档

目录：

一、整体结构

目标：使用 Logisim 开发一个简单的 MIPS 单周期处理器，并使用 Mars 编写测试程序，验证 CPU 设计的正确性。

基本思路：要求设计的 CPU 将包含 Controller（控制器）、IFU（取指令单元）、GRF（寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、EXT（位扩展器）等基本部件，通过 MUX、Splitter 等内置器件组合连接成数据通路。

设计说明：

1.处理器为 32 位处理器，支持的指令集为：{addu, subu, ori, lw, sw, beq, lui, nop}，其中 nop 机器码为 0x00000000，即空指令，不进行任何有效行为，并且 addu,subu 不支持溢出。

2.处理器为单周期设计，采用模块化和层次化设计。顶层有效的驱动信号要求包括且仅包括 reset，clk 使用内置时钟模块。

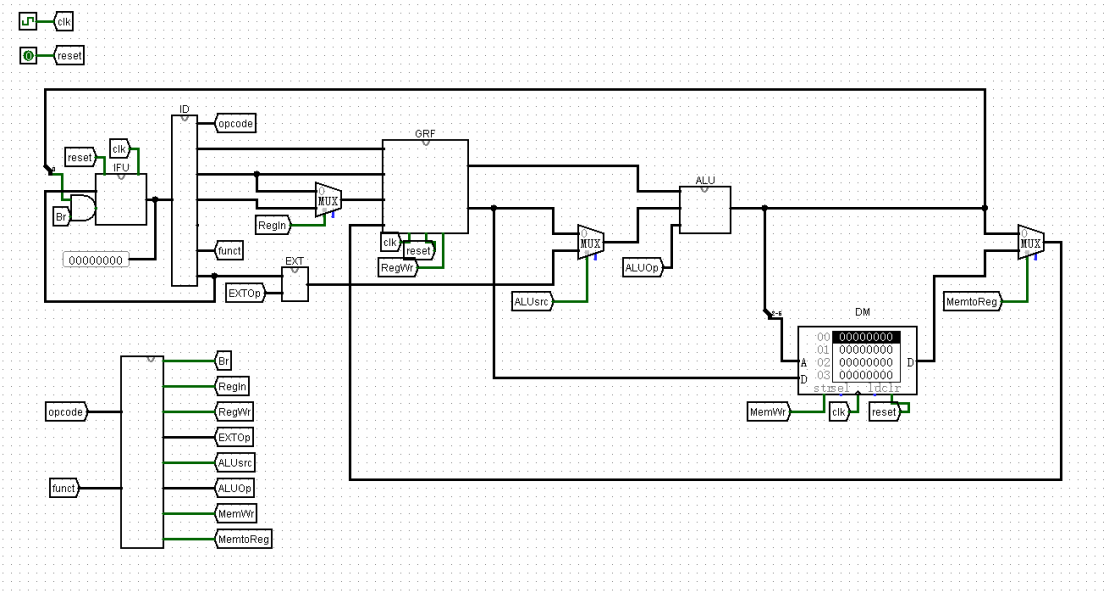

图 1.1 CPU 整体结构示意图

二、模块规格

1. IFU（取指令单元）

（1）PC（程序计数器）

PC 用 32 位寄存器实现，具有复位功能。

起始地址：0x00000000。

（2）IM（指令存储器）

IM 用 ROM 实现，容量为 32bit×32，故数据位宽 32，地址位宽 5。

由于 IM 地址宽度为 5 位，PC 为 32 位，要连接 PC 与 IM，需要用分线器将 PC 的 2-6 位提取出来，与 IM 连接。

（3）NPC（计算下一 PC）

表 2.1.1 NPC 模块端口定义

信号名	方向	描述
Imm[15:0]	I	由上一条指令提取出的 16 位立即数
Br	I	控制信号，确定下一指令的计算方法
PC[31:0]	I	当前执行指令地址
NPC[31:0]	O	下一条应执行的指令地址

表 2.1.2 NPC 模块功能描述

序号	功能描述	描述
1	连续指令	执行下一条指令
2	分支/跳转指令	执行跳转指令指向的指令

（4）端口定义与功能描述

表 2.1.3 IFU 模块端口定义

信号名	方向	描述
Imm[15:0]	I	由上一条指令提取出的 16 位立即数
Br	I	控制信号，确定下一指令的计算方法
clk	I	时钟信号
reset	I	异步复位信号
Instr	O	输出当前地址的指令

表 2.1.4 IFU 模块功能描述

序号	功能描述	描述
1	复位	reset 信号有效时，PC 寄存器清零
2	取指令	根据 PC 取出 IM 中相应指令并输出

2. GRF（寄存器堆）

用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。

此处直接使用 P0 课下测试提交的 GRF。

表 2.2.1 GRF 模块端口定义

信号名	方向	描述
clk	I	时钟信号
reset	I	异步复位信号，将 32 个寄存器中的值全部清零
WE	I	写使能信号。
A1[4:0]	I	地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD1
A2[4:0]	I	地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
A3[4:0]	I	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	O	输出 A1 指定的寄存器中的 32 位数据
RD2	O	输出 A2 指定的寄存器中的 32 位数据

表 2.2.2 GRF 模块功能描述

序号	功能描述	描述
1	复位	reset 信号有效时，所有寄存器的数值清零
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 所对应的寄存器中

3. ALU（算术逻辑单元）

提供 32 位加、减、或运算及大小比较功能，其中加减运算不检测溢出。

表 2.3.1 ALU 模块端口定义

信号名	方向	描述
A[31:0]	I	ALU 的第一个操作数
B[31:0]	I	ALU 的第二个操作数
ALUOp[1:0]	I	控制 ALU 的计算种类
C[31:0]	O	ALU 的运算结果

表 2.3.2 ALU 模块功能描述

序号	功能描述	描述
1	加运算	C = A + B
2	减运算	C = A - B
3	或运算	C = A | B
4	大小比较	若 A = B，则 C = 1，否则 C = 0

4. DM（数据存储器）

使用 RAM 实现，容量为 32bit×32，故数据位宽 32，地址位宽 5。RAM 使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports。

起始地址：0x00000000。由于只使用 RAM，故不单独设置模块。

表 2.4.1 DM 模块端口定义

信号名	方向	描述
A[5:0]	I	指向 DM 的地址
Input[31:0]	I	将要写入 DM 的数据
clk	I	时钟信号
reset	I	异步复位信号
MemWr	I	DM 写使能信号
D[31:0]	O	DM 输出的数据

表 2.4.2 DM 模块功能描述

序号	功能描述	描述
1	复位	reset 信号有效时，PC 寄存器清零

2	读数据	读出 A 地址中所存储的数据到 D
3	写数据	当 MemWr 信号有效且时钟上升沿来临时，将 Input 输入的数据写入 A 所对应的地址中

5. EXT（位扩展器）

使用 logisim 内置的 Bit Extender，实现各种神奇的扩展要求。

表 2.5.1 EXT 模块端口定义

信号名	方向	描述
A[16:0]	I	待扩展数
EXTOp[1:0]	I	控制扩展类型
B[31:0]	O	输出扩展结果

表 2.5.2 EXT 模块功能描述

序号	功能描述	描述
1	高位 0 扩展	B = {16{0}，A}
2	低位 0 扩展	B = {A，16{0}}
3	符号扩展	B = {16{A[15]}，A}
4	地址扩展	B = {14{A[15]}，A，0，0}

6.ID（指令解码）

将 IFU 提取出的指令解码为 opcode、rs、rt、rd、shamt、funct 与 Imm，方便后续使用。

表 2.6.1 ID 模块端口定义

信号名	方向	描述
Instr[31:0]	I	从 IFU 提取出的当前指令
opcode[5:0]	O	Instr[31:26]
rs[4:0]	O	Instr[25:21]
rt[4:0]	O	Instr[20:16]
rd[4:0]	O	Instr[15:11]
shamt[4:0]	O	Instr[10:6]
funct[5:0]	O	Instr[5:0]
Imm[15:0]	O	Instr[15:0]

表 2.6.2 ID 模块功能描述

序号	功能描述	描述
1	R 型指令解码	将 32 位 Instr 分解为 opcode、rs、rt、rd、shamt 和 funct
2	J 型指令解码	将 32 位 Instr 分解为 opcode、rs、rt 和 Imm

7. Controller（控制器）

使用与或门阵列构造控制信号，具体方法见下一章节。

三、控制器设计

控制器的设计，从最基本的层面来说，是一个译码的过程，将每一条机器指令中包含的信息，转化为给 CPU 各部分的控制信号。

在具体的操作中，我们把解码逻辑分解为和逻辑和或逻辑两部分：和逻辑的功能是识别，将输入的机器码识别为相应的指令；或逻辑的功能是生成，根据输入的指令的不同，产生不同的控制信号。

表 3.1 控制信号的意义

控制信号	功能名称
Br	Br = 0 时，该指令为常规指令，PC = PC + 4； Br = 1 时，该指令为分支/跳转指令，PC = PC + 4 + sign_ext(Imm||00)。
RegIn	RegIn = 0 时，数据写入 rt 寄存器；RegIn = 1 时，数据写入 rd 寄存器。
RegWr	RegWr = 0 时，GRF 不能写入数据；RegWr = 1 时，GRF 可以写入数据。
EXTOp	EXTOp = 00 时，进行高位 0 扩展，B = {16{0}，A}； EXTOp = 01 时，进行低位 0 扩展，B = {A，16{0}}； EXTOp = 10 时，进行符号扩展，B = {16{A[15]}，A}； EXTOp = 11 时，进行地址扩展，B = {14{A[15]}，A，0，0}。
ALUSrc	ALUSrc = 0 时，ALU 的第二个运算数来自 rt 寄存器； ALUSrc = 1 时，ALU 的第二个运算数来自位扩展之后的立即数。
ALUOp	ALUOp = 00 时，进行加运算，C = A + B； ALUOp = 01 时，进行减运算，C = A - B； ALUOp = 10 时，进行或运算，C = A | B； ALUOp = 11 时，进行大小比较，若 A = B，则 C = 1，否则 C = 0。
MemWr	MemWr = 0 时，DM 不能写入数据；MemWr = 1 时，DM 可以写入数据。
MemtoReg	MemtoReg = 0 时，写入寄存器的数据来自 ALU； MemtoReg = 1 时，写入寄存器的数据来自 DM。

表 3.2 控制信号产生的真值表

指令	opcode	funct	Br	RegIn	RegWr	EXTOp	ALUSrc	ALUOp	MemWr	MemtoReg
addu	000000	100001	0	1	1	X	0	00	0	0
subu	000000	100011	0	1	1	X	0	01	0	0

ori	001101		0	0	1	00	1	10	0	0
lw	100011		0	0	1	10	1	00	0	1
sw	101011		0	X	0	10	1	00	1	X
beq	000100		1	X	0	X	0	11	0	X
lui	001111		0	0	1	01	1	00	0	0
nop	000000		0	0	0	0	0	0	0	0

附表 支持指令集编码与操作

指令	编码形式	操作
addu	000000 rs rt rd 00000 100001	GPR[rd] = GPR[rs] + GPR[rt]
subu	000000 rs rt rd 00000 100011	GPR[rd] = GPR[rs] − GPR[rt]
ori	100011 base rt offset	GPR[rt] = GPR[rs] OR zero_extend(immediate)
lw	100011 base rt offset	Addr = GPR[base] + sign_ext(offset) GPR[rt] = memory[Addr]
sw	101011 base rt offset	Addr = GPR[base] + sign_ext(offset) memory[Addr] = GPR[rt]

beq	000100 rs rt offset	if (GPR[rs] == GPR[rt])
		PC = PC + 4 + sign_extend(offset‖00)
		else
		PC = PC + 4

lui	001111 00000 rt immediate	GPR[rt] = immediate‖0^16
nop	0x00000000	/

四、测试 CPU

表 4 CPU 测试程序与期望输出

测试程序	期望输出
1 ori $t0,$t0,0x0000ffff	8 号寄存器存入 0x0000ffff
2 lui $t1,0xffff	9 号寄存器存入 0xffff0000
3 addu $t2,$t0,$t1	10 号寄存器存入 0xffffffff
4 subu $t3,$t1,$t0	11 号寄存器存入 0xfffe0001
5 nop	无现象
6 beq $t1,$t1,yes	跳转至第 10 行继续执行
7 ori $t4,4	不执行
8 sw $t3,0($t4)	不执行
9 lw $t5,0($t4)	不执行
10 *yes:*	
11 beq $t0,$t1,not	不跳转，顺序执行
12 ori $t4,8	12 号寄存器存入 0x00000008
13 sw $t3,0($t4)	DM[02]存入 0xfffe0001
14 lw $t5,0($t4)	13 号寄存器存入 0xfffe0001
15 *not:*	
16 nop	无现象