单周期 CPU 报告

一、 模块规格

(一) IFU(Instruction Fetch Unit)

1. 基本描述

IFU 的主要功能是完成取指令的功能，内部包括 PC、IM(指令存储器)以及其他相关逻辑。IFU 除了能顺序执行指令以外，还能根据 BEQ 指令的执行情况决定顺序取指令还是转移取指令。

2. 基本部件端口定义

部件名称	功能	信号名	方向	位数	功能描述
PC	指向指令存储器	NPC	in	[31:0]	下一条指令的地址
		Clk	in	1	时钟信号
		Reset	in	1	复位信号 1：复位 0：无效
		PC	out	[31:0]	要读出指令寄存器的地址
NPC	计算下一个 PC 值	PC	in	[31:0]	读入当前指令的地址
		Imm	in	[15:0]	分支指令的地址偏移量
		Br	in	1	IFU 的选择信号判断指令是否为 beq 1：指令为 beq 0：指令不为 beq
		Zero	in	1	ALU 计算结果是否为 0 1：ALU 结果计算为 0 0：ALU 结果计算不为 0
		NPC	out	[31:0]	计算出下一个 PC 值 若指令不是 beq，则 PC = PC+1 若指令是 beq，且 zero 为 1，则 PC = PC+1+Imm
IM	指令存储器	Ad	in	[31:2]	读取指令的地址
		Do	out	[31:0]	取出当前指令

(二) GRF(General Register File 通用寄存器组)

1. 基本描述

GRF 主要功能是读写寄存器。GRF 内部包括 32 个寄存器以及其他相关逻辑。时钟上升沿到来时，写使能有效时，将数据写入相应的寄存器，并将相应寄存器的数据输出。

2. 端口定义及功能描述

信号名	方向	位数	功能描述
RA	in	[4:0]	读取寄存器 A 的下标
RB	in	[4:0]	读取寄存器 B 的下标
RW	in	[4:0]	写入寄存器的下标
WD	in	[31:0]	写入寄存器的数据
WE	in	1	写使能端口 在时钟上升沿到来时，如果 WE 为 1，则将 WD 的数据写入 RW 对应下标的寄存器 如果 WE 为 0，则数据无法写入
Clk	in	1	时钟信号
Reset	in	1	复位信号 1：复位有效，所有寄存器被置为 0x00000000 0：复位信号无效
busA	out	[31:0]	A 寄存器读出的数据
busB	out	[31:0]	B 寄存器读出的数据

(三) ALU(算数逻辑单元)

1. 基本描述

ALU 主要功能是将相应的数进行与控制信号相对应的运算。支持加减与或，右移 16 位等运算。

2. 端口定义

信号名	方向	位数	功能描述
A	in	[31:0]	读入的运算数 A
B	in	[31:0]	读入的运算数 B
Result	out	[31:0]	运算结果

Zero	out	1	运算结果是否为 0 1：为 0 0：不为 0
Op	in	[3:0]	运算功能选择信号

3. Op 信号功能定义

功能编号	功能	功能描述
4’b0000	加	result = A + B
4’b0001	减	result = A - B
4’b0010	按位与	result = A & B
4’b0011	按位或	result = A | B
4’b0100	左移 16 位	result = B<<16

(四) DM(Data Memory 数据存储器)

1. 基本描述

DM 主要功能是储存数据，使用 RAM 实现，容量为 32bit * 32。

2. 端口定义

信号名	方向	位数	功能描述
DataIn	in	[31:0]	写入的数据
DataOut	out	[31:0]	读出的数据
Addr	in	[4:0]	数据的写入地址
WrEn	in	1	是否写入数据到 Addr
LdEn	in	1	是否读入数据到 Addr
Reset	in	1	复位
Clk	in	1	时钟信号

(五) EXT

1. 基本描述

EXT 主要功能是将 16 位立即数扩展成 32 位。支持符号扩展和 0 扩展。

2. 端口定义

信号名	方向	位数	功能描述
Imm16	in	[15:0]	16 位立即数
Imm32	out	[31:0]	32 位拓展结果
ExtOp	in	1	控制立即数做拓展方式 0：0 拓展 1：符号拓展

二、 控制器设计

1. 基本描述

Controller 主要功能是通过指令的 opcode 和 funct 字段来判断指令的类型，从而决定各个部件的控制信号。

2. 端口定义

信号名	方向	位数	功能描述
OpCode	in	[5:0]	指令的 OpCode 字段
func	in	[5:0]	指令的 func 字段
ALUCtr	out	[3:0]	决定 ALU 的运算操作 4’b0000 加 4’b0001 减 4’b0010 按位与 4’b0011 按位或 4’b0100 左移 16 位
RegDst	out	1	0：写入到 rt：写入编号为指令中 16-20 位的寄存器 1：写入到 rd，即写入编号是指令中 11-15 位的寄存器
Reg_31W	out	1	1:向 31 位寄存器写数据
RegWrite	out	1	0：寄存器写使能端口无效 1：寄存器写使能端口有效
ALUSrc	out	1	0： ALU B 口读入的是 GRF 的 busB 1： ALU B 口读入的是立即数
MemtoReg	out	1	0：从 ALU 运算结果读入到寄存器 1：从 DM 中读入值到寄存器
Reg_PC	out	1	1: 将 pc+4 作为写入寄存器的数据
MemWrite	out	1	0：DM 写使能无效 1：DM 写使能有效
nPC_sel	out	1	0：取当前指令的下一条地址 1：当前指令地址跳转分支
ExtOp	out	1	0：进行 0 拓展 1：进行有符号拓展

3. 指令分析

```
addu:GPR[rd]<GPR[rs]+GPR[rt];PC<-PC+4
subu:GPR[rd]<- GPR[rs]-GPR[rt];PC<-PC+4
ori: GPR[rt]<- GPR[rs]+ zero_ext(imm16);PC<-PC+4
lw: GPR[rd]<- MEM[R[rs]+sign_ext(imm16)]; PC<-PC+4
sw:MEM[GPR[rs]+sign_ext(imm16)]<-GPR[rs];PC<-PC+4;
lui: GPR[rt]<-imm<<16;PC<-PC+4
beq:if(GPR[rs]==GPR[rt])
   then PC<-PC+4+[sign_ext(imm16)||00]
   else PC<-PC+4
```

指令	IFU	ALU		EXT	GRF		DM	
	imm	A	B	In	WD	RW	Addr	Din
addu		busA	busB		ALU_Result	imm15:11		
subu		busA	busB		ALU_Result	imm15:11		
ori		busA	ext_out		ALU_Result	imm15:11		
lw		busA	ext_out	imm16	DM_Do	imm20:16	ALU_Result	
sw		busA	ext_out	imm16		imm20:16	ALU_Result	busB
beq	ext_out	busA	busB	imm16				
lui		busA	ext_out	imm16	ALU_Result	imm20:16		
综合	ext_out	busA	ext_out busB	imm16	ALU_Result	imm15:11 imm20:16	ALU_Result	busB

4. 指令与控制信号真值表

func	100001	100011						000000
op	000000	000000	001101	100011	101011	000100	001111	000000
	addu	subu	ori	lw	sw	beq	lui	nop
RegDst	1	1	0	0	x	x	0	x
Reg_31W	0	0	0	0	x	x	0	x
RegWrite	1	1	1	1	0	0	1	x
ALUSrc	0	0	1	1	1	0	1	x
MemtoReg	0	0	0	1	x	x	0	x
Reg_PC	0	0	0	0	x	x	0	x
MemWrite	0	0	0	0	1	0	0	0

nPC_sel	00	00	00	00	00	01	00	00
ExtOp	0	0	0	1	1	1	0	x
ALUctr3	0	0	0	0	0	0	0	x
ALUctr2	0	0	0	0	0	0	1	x
ALUctr1	0	0	1	0	0	0	0	x
ALUctr0	0	1	1	0	0	1	0	x

func		100011						000000
op	001000	000000	001101	100011	101011	000100	001111	000000
	addi	subu	ori	lw	sw	beq	lui	nop
RegDst	0	1	0	0	x	x	0	x
Reg_31W	0	0	0	0	x	x	0	x
RegWrite	1	1	1	1	0	0	1	x
ALUSrc	1 imm	0	1	1	1	0	1	x
MemtoReg	0 alu	0	0	1	x	x	0	x
Reg_PC	0	0	0	0	x	x	0	x
MemWrite	0	0	0	0	1	0	0	0
nPC_sel	00	0	0	0	0	1	0	0
ExtOp	1 sign	0	0	1	1	1	0	x
ALUctr3	0	0	0	0	0	0	0	x
ALUctr2	0	0	0	0	0	0	1	x
ALUctr1	0	0	1	0	0	0	0	x
ALUctr0	0	1	1	0	0	1	0	x

5. 控制信号的真值表达式

指令对应的真值表达式：

addu = ~op5 ~op4 ~op3 ~op2 ~op1 ~op0 func5 ~func4 ~func3 ~func2 ~func1 func0

subu = ~op5 ~op4 ~op3 ~op2 ~op1 ~op0 func5 ~func4 ~func3 ~func2 func1 func0

ori = ~op5 ~op4 op3 op2 ~op1 op0

lw = op5 ~op4 ~op3 ~op2 op1 op0

sw = op5 ~op4 op3 ~op2 op1 op0

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

lui = ~op5 ~op4 op3 op2 op1 op0

nop = ~op5 ~op4 ~op3 ~op2 ~op1 ~op0 ~func5 ~func4 ~func3 ~func2 ~func1 ~func0

控制信号对应真值表达式：

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

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

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

MemWrite = op5 ~op4 op3 ~op2 op1 op0

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

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

ALUctr3 = 0

ALUctr2 = ~op5 ~op4 op3 op2 op1 op0

ALUctr1 = ~op5 ~op4 op3 op2 ~op1 op0

ALUctr0 = ~op5 ~op4 ~op3 ~op2 ~op1 ~op0 func5 ~func4 ~func3 ~func2 func1 func0+~op5 ~op4 op3 op2 ~op1 op0+~op5 ~op4 ~op3 op2 ~op1 ~op0

三、 测试程序

(一) MIPS 代码

```
nop          #空指令
lui $1 1     #将 1 号寄存器高 16 位设成 1    $1 = 0x00010000
addu $2 $1 $1   #$2 = $1+$1      $2 = 0x00100000
subu $3 $2 $1   #$3 = $2-$1      $3 = 0x00010000
ori $4 $1 1024  #$4 = $1or1024       $4 = 0x00010400
sw $3 12($zero)      #将$s3  存入地址 12      0x00010000
sw $2 8($zero)  #将$s2  存入地址 8       0x00100000
ori $5 $1 1024  #$5 = $1or1024       $5 = 0x00010400
beq $5 $4 lab    #若$5 = $4,跳转到标签 lab
sw $5 0($zero)
```

lab:

sw $5 4($zero)　#将$5 存入地址 4　　0x00010400

(二) 指令 16 进制码

v2.0 raw

00000000

3c010001

00211021

00411823

34240400

ac03000c

ac020008

34250400

10a40001

ac050000

ac050004

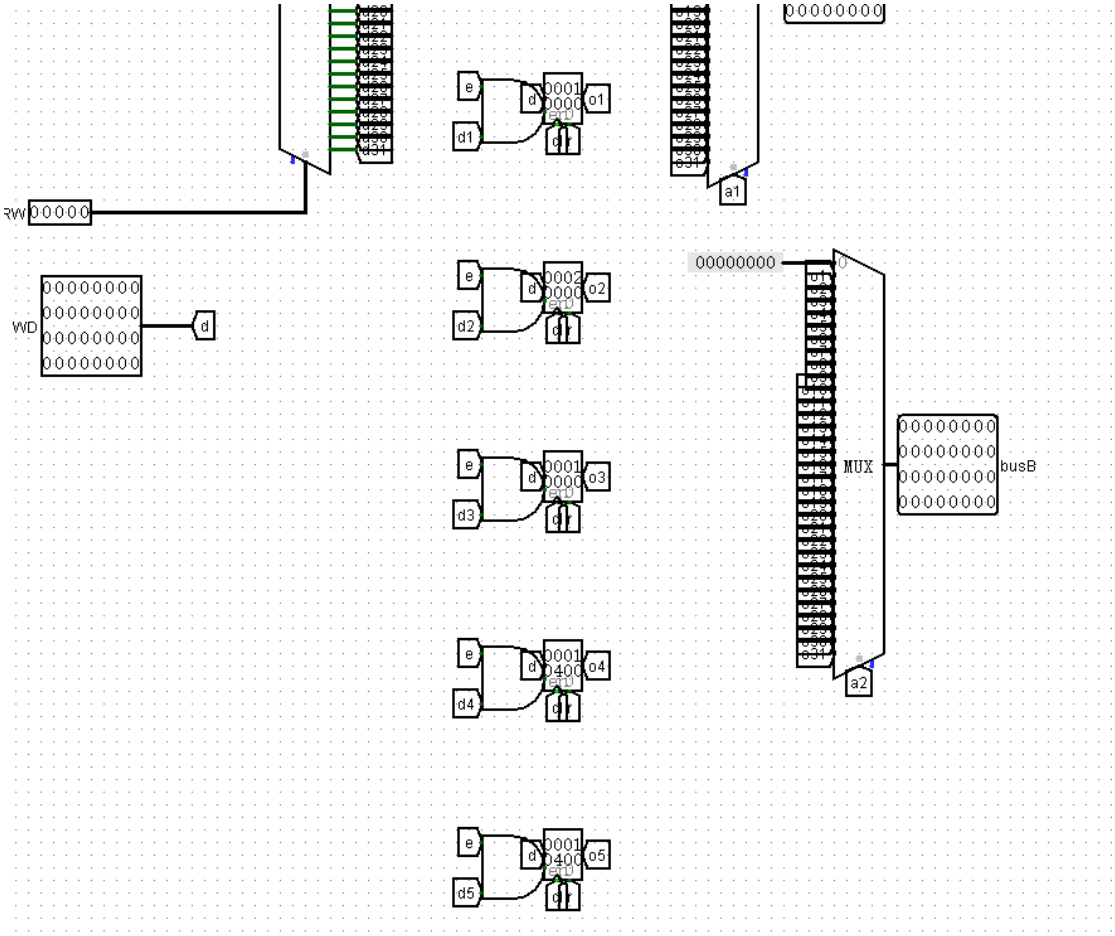
(三) mars 运行结果

$zero	0	0x00000000
$at	1	0x00010000
$v0	2	0x00020000
$v1	3	0x00010000
$a0	4	0x00010400
$a1	5	0x00010400
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000

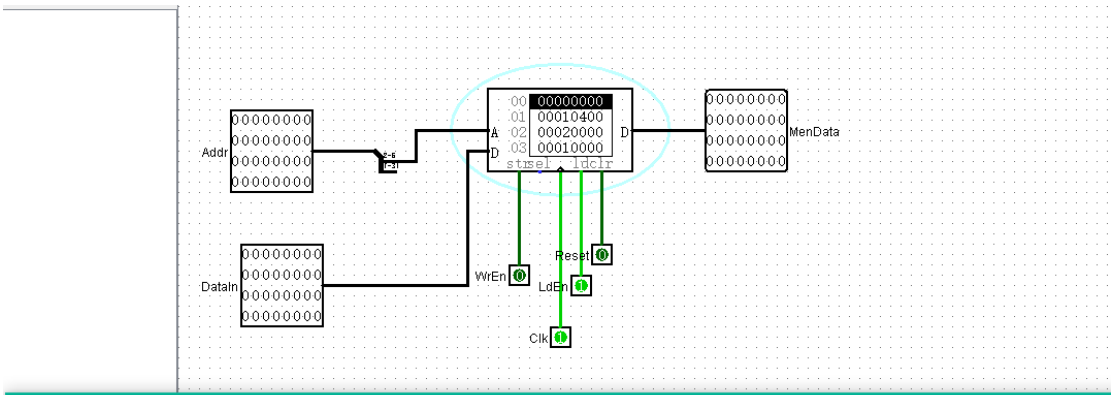
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)
0	0x00000000	0x00010400	0x00020000	0x00010000
32	0x00000000	0x00000000	0x00000000	0x00000000
64	0x00000000	0x00000000	0x00000000	0x00000000
96	0x00000000	0x00000000	0x00000000	0x00000000
128	0x00000000	0x00000000	0x00000000	0x00000000
160	0x00000000	0x00000000	0x00000000	0x00000000
192	0x00000000	0x00000000	0x00000000	0x00000000
224	0x00000000	0x00000000	0x00000000	0x00000000
256	0x00000000	0x00000000	0x00000000	0x00000000
288	0x00000000	0x00000000	0x00000000	0x00000000

(四) CPU 运行结果

GRF:



DM:



Logisim: Hex Editor

File Edit Project Simulate Window Help

00	00000000	00010400	00020000	00010000	00000000	00000000	00000000	00000000
08	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
18	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

P3 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。**

   30 位的优点：两条指令之间相差一个单位，30 位更加方便。

   计算 PC 的时候无需左移 2 位。

   30 位的缺点：读入的拓展后的立即数需要截取高 30 位，增加操作的复杂度。

   一条指令实际为 32 位，占 4 个地址，可读性高。

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。**

   合理，只需从 IM 中取指令，所以 IM 用 ROM 即可。

   DM 需要读数据和写数据，RAM 即能做到。

   GRF 使用 32 个寄存器方便对单个寄存器进行修改，需要快速读写操作，使用率高。

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

   MemtoReg = op0 op1 ~op2 ~op3 ~op4 op5

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

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

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

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

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

MemtoReg = op0 op1 ~op2 ~op3 ~op4 op5

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

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

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

5. 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 指令为 0x00000000，写入 0 号寄存器，在 mips 中对应的是 sll $0, $0, 0。0 号寄存器不会被改变，故没有影响。

6. 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。

可以将读入的地址信号右移两位，即为以字为单位的 DM 的地址位置。

7. 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。

形式验证方法分为等价性验证、模型检验和定理证明等。

形式验证的优点：

(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%。测试并不一定可以测试到所有的情况，相当于只是接受输入，得到将结果进行比较，不能涉及到内部逻辑。

(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。

(3)形式验证的验证时间短，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。

缺点：

相对测试，形式验证会比较复杂，涉及到数学方法，而测试只是简单比对。