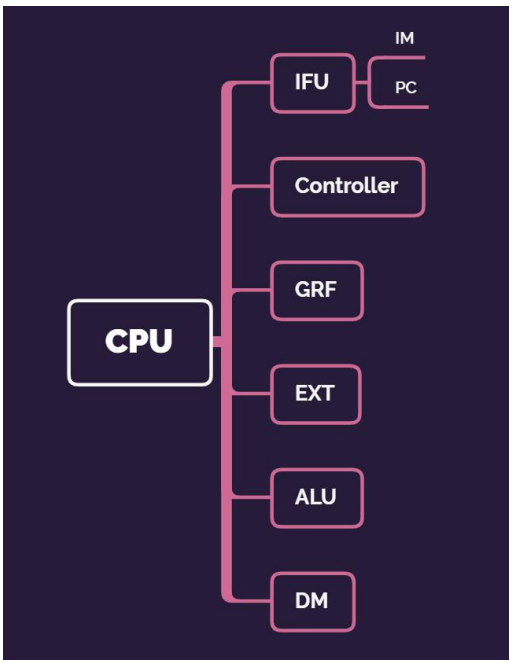
计算机组成原理实验报告参考模板

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的流水线 MIPS - CPU，支持的指令集包含{addu, subu, ori, lw, sw, beq, lui, nop}。为了实现这些功能，CPU 主要包含了 IM、GRF、ALU、PC、IM、EXT、Controller、IFU 等模块，这些模块按照以下顶层设计逐级展开：



（二）关键模块定义

1. GRF

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	异步复位信号， 1：清零 0：保持
3	WE	I	写使能功能，从外界接收 RegWrite

			1：可向 GRF 写入数据 0：不能向 GRF 写入数据
4	A1[4:0]	I	5 位地址输入，从外界接收[25：21]，代表 32 个寄存器中的一个，并将其中的值输出到 RD1
5	A2[4:0]	I	5 位地址输入，从外界接收[20：16]，代表 32 个寄存器中的一个，并将其中的值输出到 RD2
6	A3[4:0]	I	5 位地址输入，选择 32 个寄存器中的一个，将 WD 输入的数据存储到其中
7	WD[31:0]	I	32 位写入数据
8	RD1[31:0]	O	输出 A1 代表的寄存器中的值
9	RD2[31:0]	O	输出 A2 代表的寄存器中的值

（2）功能定义

序号	功能	描述
1	异步复位	Reset 为 1 时，所有寄存器被清零
2	读数据	将 A1 和 A2 对应的寄存器中的值输出到 RD1 和 RD2
3	写数据	如果 WE 为 1，时钟上升沿时将 WD 中的数据写入到 A3 对应的寄存器中

2. DM

(1) 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	异步复位信号， 1：清零 0：保持
3	A[31:0]	I	择取其[6:2]位，作为读取或写入信

			号的地址
4	WD[31:0]	I	32 位要写入的数据
5	WE	I	写使能信号，外界对应 MemWrite
6	RD[31:0]	O	32 位读出的数据

（2）功能定义

序号	功能	描述
1	异步复位	Reset 为 1 时，所有寄存器被清零
2	读数据	将 A 地址对应的数据输出到 RD（这个功能始终存在，不管 WE 是否为 1）
3	写数据	WE 为 1 且时钟上升沿到来时，将 WD 的数据写入到 A 对应的地址

3. ALU

(1) 端口说明

序号	信号名	方向	描述
1	ALUControl[2:0]	I	选择执行哪一个操作 000：加法 001：减法 010：与 011：或 100：左移 16 位
2	SrcA[31:0]	I	参与运算的第一个数
3	SrcB[31:0]	I	参与运算的第二个数
4	Zero	O	如果 SrcA 等于 ScrB，输出 1； 如果不等，输出 0
5	ALUResult[31:0]	O	输出 SrcA 和 SrcB 操作后的结果

（2）功能定义

序号	功能	描述
1	加法	将两个输入加起来输出到 ALUResult
2	减法	将两个输入相减输出到 ALUResult
3	与运算	对两个输入进行与操作输出结果到 ALUResult
4	或运算	对两个输入进行或操作输出结果到 ALUResult
5	左移 16 位	将 SrcB 左移 16 位输出到 ALUResult

4. EXT

(1) 端口说明

序号	信号名	方向	描述
1	Imm16[15:0]	I	需要符号扩展的立即数，取自指令[15:0]
2	sign	I	判断是进行符号扩展还是无符号扩展 1：符号扩展 0：无符号扩展
3	SignImm[31:0]	O	符号扩展后的 32 位输出

（2）功能定义

序号	功能	描述
1	无符号扩展	当 sign 为 0 时，进行无符号扩展
2	符号扩展	当 sign 为 1 时，进行有符号扩展

5. Controller

(1) 端口说明

序号	信号名	方向	描述
1	op[5:0]	I	Instr[31:26]，6 位控制信号
2	func[5:0]	I	Instr[5:0]，6 位控制信号
3	sign	O	判断有符号扩展 or 无符号扩展的信号
4	Branch	O	判断是否是 beq 信号
5	MemWrite	O	DM 的写入信号

6	RegWrite	O	GRF 的写入信号
7	MemtoReg	O	将哪个数据写入 GRF 0：ALUResult 1：DM 的 ReadData
8	ALUSrc	O	哪个数是 SrcB 0：GRF 的 RD2 1：EXT 的 SignImm
9	RegDst	O	GRF 写入地址的选择 0：rt 1：rd
10	ALUControl	O	ALU 的控制信号

（2）真值表

端口	addu	subu	ori	lw	sw	beq	lui
func	100001	100011	n/a	n/a	n/a	n/a	n/a
op	000000	000000	001101	100011	101011	000100	001111
sign	x	x	0	1	1	x	x
Branch	0	0	0	0	0	1	0
MemWrite	0	0	0	0	1	0	0
RegWrite	1	1	1	1	0	0	1
MemtoReg	0	0	0	1	x	x	0
ALUSrc	0	0	1	1	1	0	1
RegDst	1	1	0	0	x	x	0
ALUControl	000	001	011	000	000	001	100

6. IFU

(1) 端口说明

序号	信号名	方向	描述

1	clk	I	时钟信号
2	reset	I	异步复位信号
3	PC’[31:0]	I	PC 下一时刻的值
4	Instr[31:0]	O	本时刻 PC 对应从 IM 中取出的指令
5	PC+4[31:0]	O	PC 可能的下一时刻的值，还要与 PCBranch 做一个多路选择

（2）功能定义

序号	功能	描述
1	异步复位	当 reset 为 1 时，PC 置零
2	更新 PC	将 PC’存入 PC
3	输出指令	根据 PC 从 IM 中读取 Instr 并输出

二、测试方案

1.测试代码

```
.text
   ori $t0, $0, 100
   ori $t1, $0, 50
   ori $t2, $0, 250
   ori $t3, $0, 4
   sw $t0, ($t3)
   lw $s0, ($t3)
   loop:
      beq $t1, $t2, end
      addu $t1, $t1, $t0
      beq $t1, $t4, loop
   end:
```

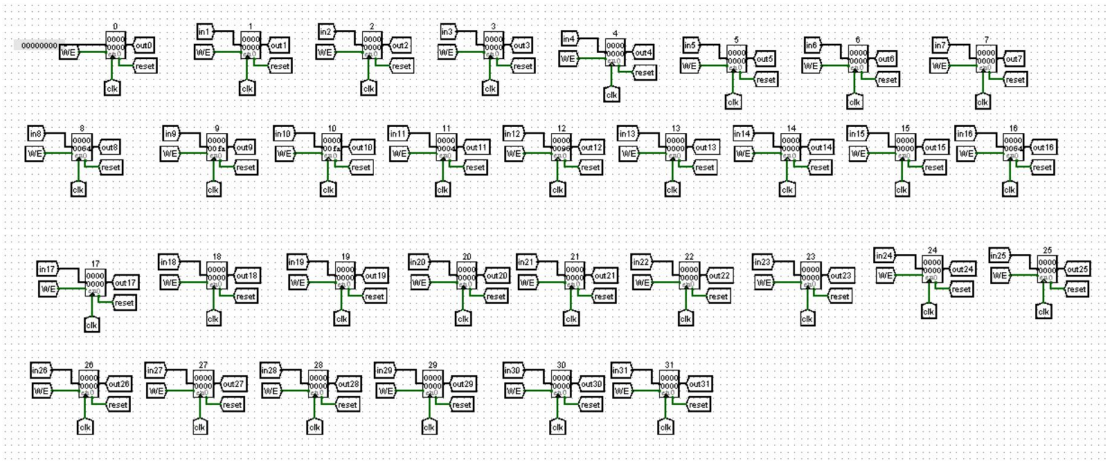
2. MARS 中运行结果

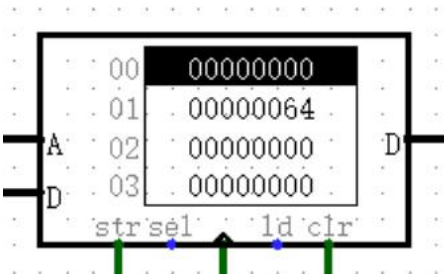
Registers | Coproc 1 | Coproc 0

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	0x00000064
$t1	9	0x000000fa
$t2	10	0x000000fa
$t3	11	0x00000004
$t4	12	0x00000096
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000064
$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		0x00003028
hi		0x00000000
lo		0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x00000064	0x00000000	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
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

3. 该 CPU 运行结果





三、思考题

（一）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。

合理。

IM 只需要被读取，而 ROM 只有读取功能，刚好适配；

DM 有写入和读取功能，对应的有 RAM 和寄存器；由于 DM 需要很大的空间存储需求，所以用寄存器就很浪费，RAM 正好；

GRF 需要高速读写功能，所以用寄存器。

无意见

（二）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。

因为 nop 空指令全为 0，control 模块全输出 0，并不会改变 CPU 里面的模块，所以不用将其加入控制信号。

（三）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，可以通过为 DM 增添片选信号，来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。

如果地址在 0x30000000 到 0x3fffffff 之间，且 CPU 地址从 0 开始的话，直接将地址减去 0x30000000 即可；如果 DM 起始地址未知的话，可以在 DM 设置两个 RAM，比较地址和 0x30000000，得到片选信号，选择哪个 RAM。

（四）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，简要阐述相比于测试，形式验证的优劣之处。

形式验证优点：

- 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%。
- 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。
- 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：

- 形式验证只能进行逻辑检查，并不能进行动态验证
- 随着设计复杂性的增加，形式验证所需状态空间将呈指数型增长，因此极大限制了形式验证系统级别的应用