

中山大学微电子科学与技术学院

《数字电路》实验报告

姓名 1: 蔡晨翔 学号 1: 19301002 姓名 2: 曾颖昕 学号 2: 19322002
姓名 3: 陈怡君 学号 3: 21311458 姓名 4: 郑茗元 学号 4: 20355063

期末大作业 基于 vivado 的 CPU 设计

2023 年 1 月 9 日

一、实验目的

- 1、基于 Verilog 描述语言设计一个 risc-v 处理器核。
- 2、实现 CPU 五级流水的设计。

二、实验原理

2.1 RISC-V 指令集

RISC-V 指令集是由加州大学伯克利分校于 2010 年提出的是一个基于精简指令集(RISC)原则的开源指令集架构(ISA)。RISC-V 采用模块化的指令集，易于扩展、组装。

RISC-V ISA = 1 个基本整数指令集(I) + 多个可选的扩展指令集

其中唯一强制要求实现的基础指令集，其他指令集都是可选的扩展模块。
RISC-V 允许在实现中以可选的形式实现其他标准化和非标准化的指令集扩展。

表 2.1 RISC-V 基本指令集

基本指令集	指令数	描述
RV32I	47	支持 32 位地址空间与整数指令，支持 32 个通用整数寄存器
RV32E	47	RV32I 的子集，仅支持 16 个通用整数寄存器
RV64I	59	支持 64 位地址空间与整数指令及一部分 32 位的整数指令
RV128I	71	支持 128 位地址空间与整数指令及一部分 64 位和 32 位的整数指令

表 2.2 RISC-V 指令集扩展

扩展指令集	指令数	描述
M	8	整数乘法与除法指令
A	11	存储器原子 (atomic) 操作指令和 load-reserved/store-conditional 指令
F	26	单精度 (32 位) 浮点指令
D	26	双精度 (64 位) 浮点指令，必须支持 F 扩展指令集
C	46	压缩指令，指令长度为 16 位

2.2 CPU 架构

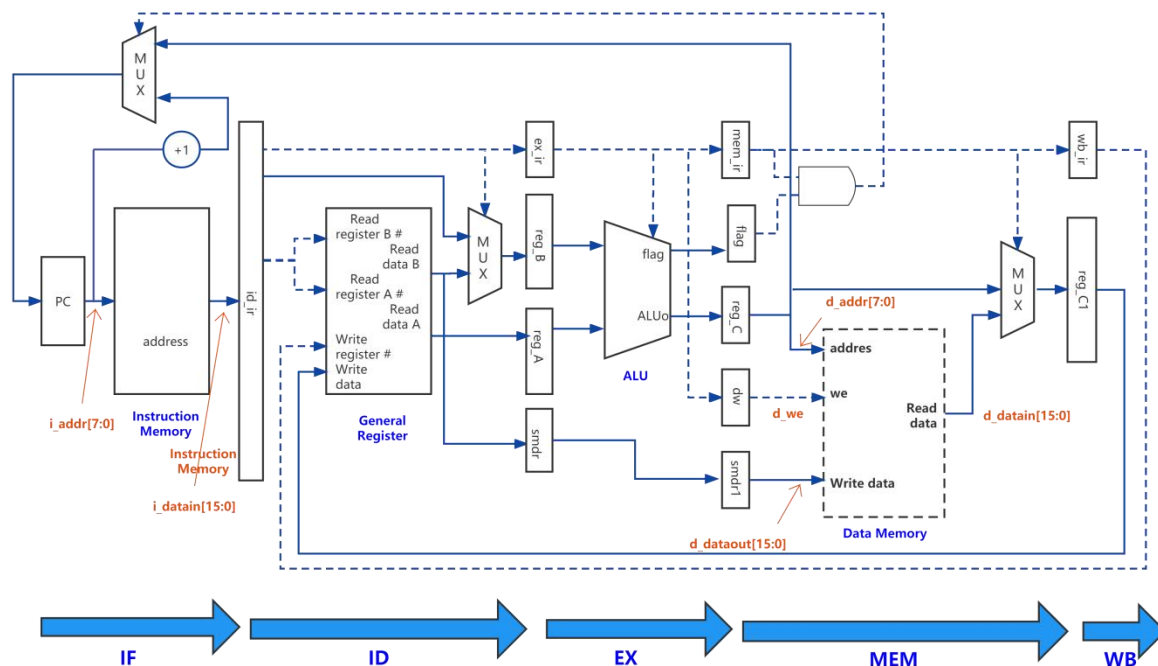


图 2.1 CPU 五级流水线架构

2.3 五级流水线

1. 取指 (Instruction Fetch, IF)

本阶段主要是根据 PC 寄存器的值来从存储器相应地址中读取对应的指令信息的过程。

2. 译码 (Instruction Decode, ID)

指令译码是指将存储器中取出的指令进行翻译的过程。经过译码之后得到指令需要的操作数寄存器索引，可以使用此索引从通用寄存器组中将操作数读出。

3. 执行 (Instruction Execute, EX)

指令译码之后所需要进行的计算类型都已得知，并且已经从通用寄存器组中读取出了所需的操作数，那么接下来便进行指令执行。指令执行是指对指令进行真正运算的过程。譬如，如果指令是一条加法运算指令，则对操作数进行加法操作；如果是减法运算指令，则进行减法操作。在“执行”阶段的最常见部件为算术逻辑部件运算器(ALU)，作为实施具体运算的硬件功能单元。

4. 访存 (Memory Access, MEM)

存储器访问指令往往是指令集中最重要的指令类型之一，访存是指存储器访问指令将数据从存储器中读出，或者写入存储器的过程。

5. 写回 (Write-Back, WB)

写回是指将指令执行的结果写回通用寄存器组的过程。如果是普通运算指令，该结果值来自于“执行”阶段计算的结果；如果是存储器读指令，该结果来自于“访存”阶段从存储器中读取出来的数据。一个简单的五级流水线指令执行过程时空图如表 2.3 所示，我们可以看出，随着时钟周期的深入，最终将同时有五条指令处在不同的处理阶段，这就十分明显的增加了系统对指令的吞吐速率。

表 2.3 简单 RISC 流水线

指令编号	时钟编号								
	1	2	3	4	5	6	7	8	9
指令 i	IF	ID	EX	MEM	WB				
指令 i+1		IF	ID	EX	MEM	WB			
指令 i+2			IF	ID	EX	MEM	WB		
指令 i+3				IF	ID	EX	MEM	WB	
指令 i+4					IF	ID	EX	MEM	WB

2.4 寄存器

在 CPU 中至少要有六类寄存器：指令寄存器(IR)、程序计数器(PC)、地址寄存器(AR)、数据寄存器(DR)、累加寄存器(AC)、程序状态字寄存器(PSW)。这些寄存器用来暂存一个计算机字，其数目可以根据需要进行扩充。

1. 数据寄存器

数据寄存器(Data Register, DR)又称数据缓冲寄存器，主要功能是作为 CPU 和主存、外设之间信息传输的中转站，用以弥补 CPU 和主存、外设之间操作速度上的差异。数据寄存器用来暂时存放由主存储器读出的一条指令或一个数据字；当向主存存入一条指令或一个数据字时，也将它们暂时存放在数据寄存器中。

2. 指令寄存器

指令寄存器(Instruction Register, IR)用来保存当前正在执行的一条指令。当执行一条指令时，首先把该指令从主存读取到数据寄存器中，然后再传送至指令寄存器。指令寄存器中操作码字段的输出就是指令译码器的输入。操作码一经译码，即可向操作控制器发出具体操作的特定信号。

3. 程序计数器

程序计数器(Program Counter, PC)用来指出下一条指令在主存储器中的地址。在程序执行之前，首先必须将程序的首地址，即程序第一条指令所在主存单元的地址送入 PC，因此 PC 的内容即是从主存提取的第一条指令的地址。

4. 地址寄存器

地址寄存器(Address Register, AR)用来保存 CPU 当前所访问的主存单元的地址。由于在主存和 CPU 之间存在操作速度上的差异, 所以必须使用地址寄存器来暂时保存主存的地址信息, 直到主存的存取操作完成为止。

当 CPU 和主存进行信息交换时, 要使用地址寄存器和数据寄存器;如果我们把外围设备与主存单元进行统一编址, 那么当 CPU 和外围设备交换信息时, 我们同样要使用地址寄存器和数据寄存器。

5. 累加寄存器

累加寄存器通常简称累加器(Accumulator, AC), 是一个通用寄存器。其功能是在运算器的算术逻辑单元 ALU 执行算术或逻辑运算时, 为 ALU 提供一个工作区, 可以为 ALU 暂时保存一个操作数或运算结果。显然, 运算器中至少要有一个累加寄存器。

6. 程序状态字寄存器

程序状态字(Program Status Word, PSW)用来表征当前运算的状态及程序的工作方式。程序状态字寄存器用来保存由算术/逻辑指令运行或测试的结果所建立起来的各种条件码内容, 如运算结果进/借位标志(C)、运算结果溢出标志(O)、运算结果为零标志(Z)、运算结果为负标志(N)等, 这些标志位通常用 1 位触发器来保存。

除此之外, 程序状态字寄存器还用来保存中断和系统工作状态等信息, 以便 CPU 和系统及时了解机器运行状态和程序运行状态。因此, 程序状态字寄存器是一个保存各种状态条件标志的寄存器。

2.5 运算器

1. 算术逻辑运算单元 ALU(Arithmetic and Logic Unit)

ALU 主要完成对二进制数据的定点算术运算(加减乘除)、逻辑运算(与或非异或)以及移位操作。在某些 CPU 中还有专门用于处理移位操作的移位器。通常 ALU 由两个输入端和一个输出端。整数单元有时也称为 IEU(Integer Execution Unit)。我们通常所说的“CPU 是 XX 位的”就是指 ALU 所能处理的数据的位数。

2. 浮点运算单元 FPU(Floating Point Unit)

FPU 主要负责浮点运算和高精度整数运算。有些 FPU 还具有向量运算的功能, 另外一些则有专门的向量处理单元。

2.6 加法器

1. 半加器

半加器是指对输入的两个一位二进制数相加(A 与 B), 输出一个结果位(SUM)和进位(C), 没有进位的输入加法器电路, 是一个实现一位二进制数的加法电路。

当 A 和 B 相同时 SUM 为 0，否则为 1，逻辑关系属于异或；当 A 和 B 同时为 1 时，C 等于 1，其余都为零，逻辑关系为与。所以我们可以得到如下的逻辑表达式：

$$SUM = A \oplus B, C = AB$$

2. 全加器

全加器是指对输入的两个二进制数相加(A 与 B)同时会输入一个低位传来的进位(C_{i-1})，得到和数(SUM)和进位(C_i)；一位全加器可以处理低位进位，并输出本位加法进位。两个半加器可以构成一个全加器，多个一位全加器进行级联可以得到多位全加器。常用二进制四位全加器 74LS283。最常用的逻辑表达式：

$$SUM_i = A_i \oplus B_i \oplus C_{i-1}, C_i = A_i C_{i-1} + A_i B_i + B_i C_{i-1} = (A_i \oplus B_i) \oplus C_{i-1} + A_i B_i$$

三、实验内容、方法与结果

3.1 设计思路

采用自顶向下的设计思路将设计分解成 CPU 核心和辅助外设两部分。CPU 采用五级流水线设计，分为获取指令的 IF 部分，解析指令的 ID 部分，执行指令的 EX 部分，访内存读写的 MEM 部分和将数据写回寄存器堆的 WB 部分，除此之外还需要用到进行逻辑运算的 ALU 模块，然后再用指令流水线串联以上模块。

在编写代码时，先将指令进行归类，分为控制、计算、存储等类型，再按类型进行该部分代码编写，可简化编写代码时的思路，提高编写效率。对于外设部分，分为解决时序问题的时钟降频模块，显示数据的数码管模块，用于存储数据和指令的 DMEM 和 MEM 存储模块等。

为解决 Hazard 问题，我们采取哈佛架构将数据和指令分开存储的方法，用物理空间复杂换取控制逻辑上的简洁；对于数据冒险，先判断是否为 LOAD 冒险。先查阅到相关冒险出现的条件后，用 if 语句进行判断，然后对其进行阻塞处理；对于其他数据冒险，采用数据直通处理。

对于测试方法，采用观察数据存储内数据变化为验证方法，在仿真中查看波形，在 Tcl Console 中查看打印出的不同地址的内存数据，然后进行对比分析。

3.2 实验方法

本次基于 vivado 的 CPU 设计主要分为 CPU 总控制、指令集、测试数据集、测试指令集、时钟模块、顶部文件和测试文件。**(具体代码见附录)**

3.2.1 CPU 总控制——PCPU.V

该模块为核心部分，内含有 CPU 状态控制、IF、ID、ALU、EX、MEM 和 WB 这 7 部分。

CPU 状态控制为时序逻辑，表示 CPU 当前状态，有两个枚举值 exec 和 idle，

分别是正在运行和空闲两种状态，其通过 reset 信号控制 state 信号的值，通过 state、enable、start、wb_ir 信号控制下一状态是运行还是空闲。

IF 为时序逻辑，每一个上升沿从 Instruction Memory 中根据地址 pc 取出一条指令，LOAD 指令需要根据不同的指令从当前 CPU 运算中特定位置读取；Branch 指令及其标志位为 true 则 Flush 掉一条指令；跳转指令则直接跳转并处理 pc 值延后一个周期的情况，其余情况直接读取下一条指令。

ID 为时序逻辑，根据不同的指令，从通用寄存器中取出相应的值出来作运算或者存储到 Data Memory 中；如果为 Branch 指令且标志位为 true 则需要 Flush 掉当前内容；如果冲突出现，则需要根据不同的运算指令从不同的地方取出内容。

ALU 为组合逻辑，根据不同的指令将对 reg_A 和 reg_B 进行计算并将结果送入 {cf,ALUo} 中，指令均为 EX 阶段的指令。

EX 为时序逻辑，根据不同的运算指令作不同的运算。如果为存储指令则直接转移到下一级寄存器，并将写入指令标记为 true；如果为其余指令，则保持当前内容不变。

MEM 为时序逻辑，只有当 LOAD 指令需要从 Data Memory 里面取出内容，其余情况把 reg_C 传递给 reg_C1 即可。cf 进位标志在此阶段更新。

WB 为时序逻辑，对需要写回到寄存器的指令把运算结果写回到寄存器中，其余情况只需让寄存器保持不变。

流水数据传输方法是在 ID、EX、MEM、WB 阶段分别定义 id_ir, ex_ir, mem_ir, 和 wb_ir 这 4 种 16 位指令寄存器，会有 4 个指令寄存器的重置代码，也有寄存器传递数据的代码例如“ex_ir <= id_ir”。

3.2.2 指令集——headfile.v

本次用到的 16 位指令集组成如图 3.1 所示，第 15~11 位是操作码(op code)、第 10~8 位是操作数 1，存储 3 位的通用寄存器编码；第 7~4 位是操作数 2，存储通用寄存器编码或 4 位立即数；第 3~0 位是操作数 3，存储通用寄存器编码或 4 位立即数。

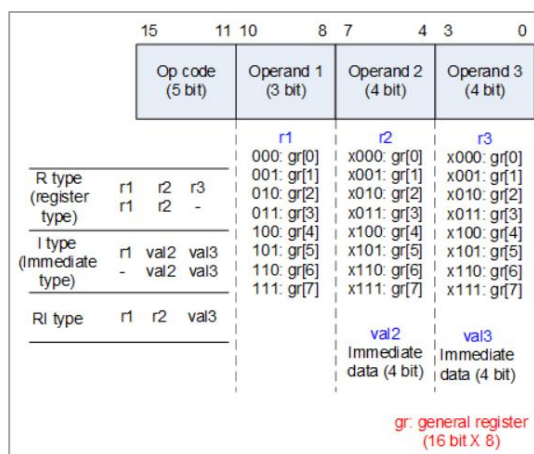


图 3.1 指令集说明图

3.2.3 测试数据集——D_mem.v

D_mem 是数据存储器，读取数据时，根据地址直接读取数据输出。写入 Data 是时序逻辑，每一个周期可能会写入一次。测试的功能需要的数据也不同，故在注释中内含多组数据，对应 I_mem 中提到的五种测试所需要的数据。

3.2.4 测试指令集——I_mem.v

I_mem 用于存储各指令的编码，组合逻辑，读入地址后，输出各指令的 16 位编码。测试的功能需要的指令不同，故内含多组指令，可实现全指令测试、冒泡排序测试、最大公因数及最小公倍数测试、从小到大排序测试五种测试。

3.2.5 时钟模块——clkd4.v

用于生成时钟脉冲信号。本文件内编写分频系数 M 为 16 和计数器值 N 为 8，即有 16 分频效果。

3.2.6 顶部文件——CPU.v

顶层模块，实例化 I_mem、D_mem、PCPU、clkd4 各模块，并描述各实例互相的连接关系，具体的连接关系可以见 CPU 结构框图。

3.2.7 测试文件——test.v

仿真测试文件，需要各使能信号初始化，通过输出用到的通用寄存器和用到的 d_mem 里面的变量来观察整个 CPU 的流程结果，测试 CPU 是否正确工作。以及观察仿真流程的变化过程测试 CPU 是否正常工作。最终能显示时序波形功能图和在 TCL 控制台显示数据。

3.3 实验内容及结果

在 vivado 当中编写工程文件，设置 xsim. simulate.runtime 为 40ns 以便于观察结果，并选择储存于 ADD=0 ~ b 的数据添加到波形图中显示。

3.3.1 复杂指令测试

```
# run 40ns
ADD:00000000, CELL:fffd
ADD:00000001, CELL:0004
ADD:00000002, CELL:0005
ADD:00000003, CELL:c369
ADD:00000004, CELL:69c3
ADD:00000005, CELL:0041
ADD:00000006, CELL:ffff
ADD:00000007, CELL:0001
ADD:00000008, CELL:0000
ADD:00000009, CELL:0000
ADD:0000000a, CELL:0000
ADD:0000000b, CELL:xxxx
ADD:0000000c, CELL:xxxx
ADD:0000000d, CELL:xxxx
ADD:0000000e, CELL:xxxx
ADD:0000000f, CELL:xxxx
ADD:00000010, CELL:b600
ADD:00000011, CELL:0001
ADD:00000012, CELL:0005
ADD:00000013, CELL:0001
ADD:00000014, CELL:ffff
ADD:00000015, CELL:fffe
ADD:00000016, CELL:fffe
ADD:00000017, CELL:fffe
ADD:00000018, CELL:fffe
ADD:00000019, CELL:c369
ADD:0000001a, CELL:86d2
ADD:0000001b, CELL:3690
ADD:0000001c, CELL:8000
ADD:0000001d, CELL:c369
ADD:0000001e, CELL:61b4
ADD:0000001f, CELL:00c3
ADD:00000020, CELL:0001
ADD:00000021, CELL:c369
ADD:00000022, CELL:86d2
ADD:00000023, CELL:6900
ADD:00000024, CELL:8000
ADD:00000025, CELL:69c3
ADD:00000026, CELL:d386
ADD:00000027, CELL:c300
ADD:00000028, CELL:8000
ADD:00000029, CELL:c369
ADD:0000002a, CELL:61b4
ADD:0000002b, CELL:00c3
ADD:0000002c, CELL:0001
ADD:0000002d, CELL:69c3
ADD:0000002e, CELL:34e1
ADD:0000002f, CELL:0069
ADD:00000030, CELL:0000
ADD:00000031, CELL:xxxx
ADD:00000032, CELL:xxxx
ADD:00000033, CELL:xxxx
ADD:00000034, CELL:xxxx
ADD:00000035, CELL:xxxx
ADD:00000036, CELL:xxxx
ADD:00000037, CELL:xxxx
ADD:00000038, CELL:0030
ADD:00000039, CELL:xxxx
ADD:0000003a, CELL:xxxx
ADD:0000003b, CELL:xxxx
```

图 3.2 复杂指令测试结果

复杂指令测试的结果如图 3.2 所示，由于数据较多，在此不展示波形图。

3.3.2 最大公约数和最小公倍数测试

测试结果如图 3.3 所示，这是一个求最大公约数和最小公倍数的例子，图中数字是以十六进制显示的，转换为十进制就是求 32 和 24 的最大公约数和最小公倍数，结果分别为 8 和 96，以十六进制显示在图 3.3 中就是 ADD=3 时的 8H 和 ADD=4 时的 60H。

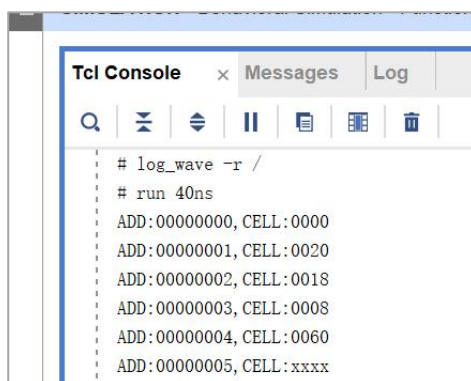


图 3.3 最大公约数和最小公倍数测试结果

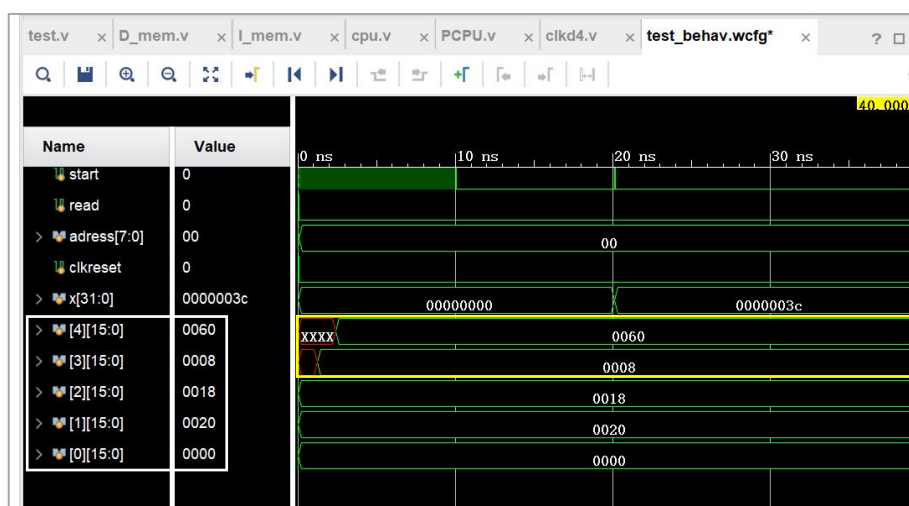


图 3.4 最大公约数和最小公倍数波形图

测试得到的波形图如图 3.4 所示。可以看出除了黄色框中储存 0060H 和 0008H 的部分外，其余部分的时序图最开始都没有不定态，电路工作正常；而他们一开始为不定态是因为他们是本次测试的输出，在最开始并没有产生信号，因此为不定态。

图 3.4 中白色框内也显示了 ADD 从 0~4 的储存情况，储存的数据与波形图显示的数据一致，也与图 3.3 中的数据一致。符合实验预期。

3.3.3 64 位加法器

测试结果如图 3.5 所示，易得：

$$0000_fffe_fffe_fffe + 0000_ffff_ffff_ffff = 0001_fffe_fffe_ffff$$

因此图 3.5 中最下面的红框中的数据为上面两个红框中的数据相加之和，实现了 64 位加法器的功能。

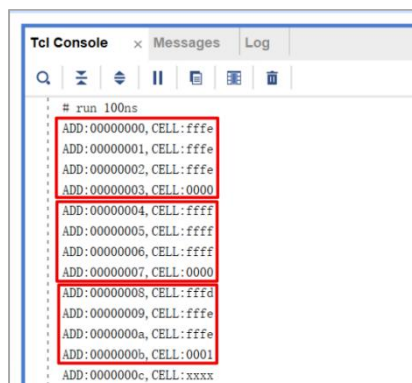


图 3.5 64 位加法器测试结果

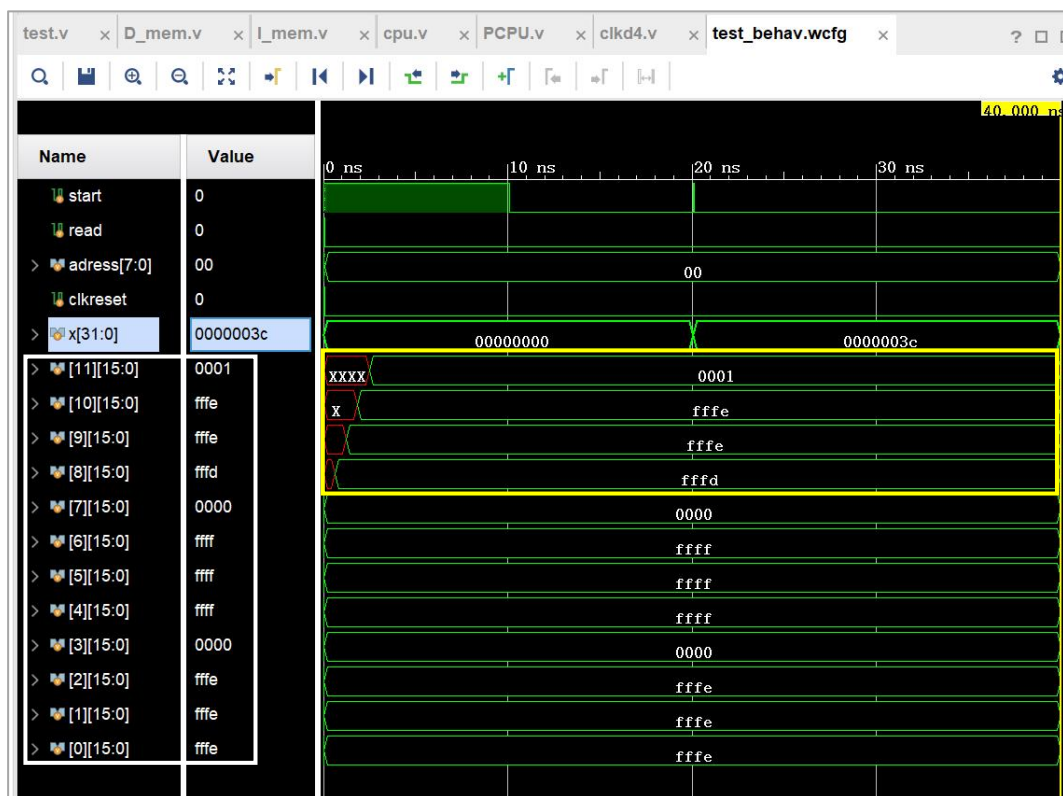


图 3.6 64 位加法器波形图

测试得到的波形图如图 3.6 所示。可以看出除了黄色框储存 0001H、fffeH、fffeH 和 fffdH 的部分外，其余部分的时序图最开始都没有不定态，电路工作正常；而他们一开始为不定态是因为他们是本次测试的输出，在最开始并没有产生信号，因此为不定态。

图 3.6 中白色框内也显示了 ADD 从 0~11(b)的储存情况，储存的数据与波形图显示的数据一致，也与图 3.5 中的数据一致。符合实验预期。

3.3.4 冒泡排序测试

测试结果如图 3.7 所示，ADD=0 时的 000a 表示对 10 个十六进制数进行冒泡

排序，可以看见 ADD=1 到 ADD=a 的数字已经按照由大到小的顺序排列。

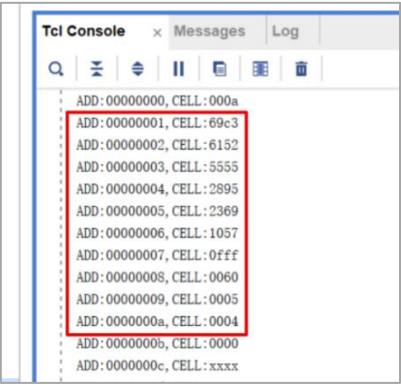


图 3.7 冒泡排序测试结果

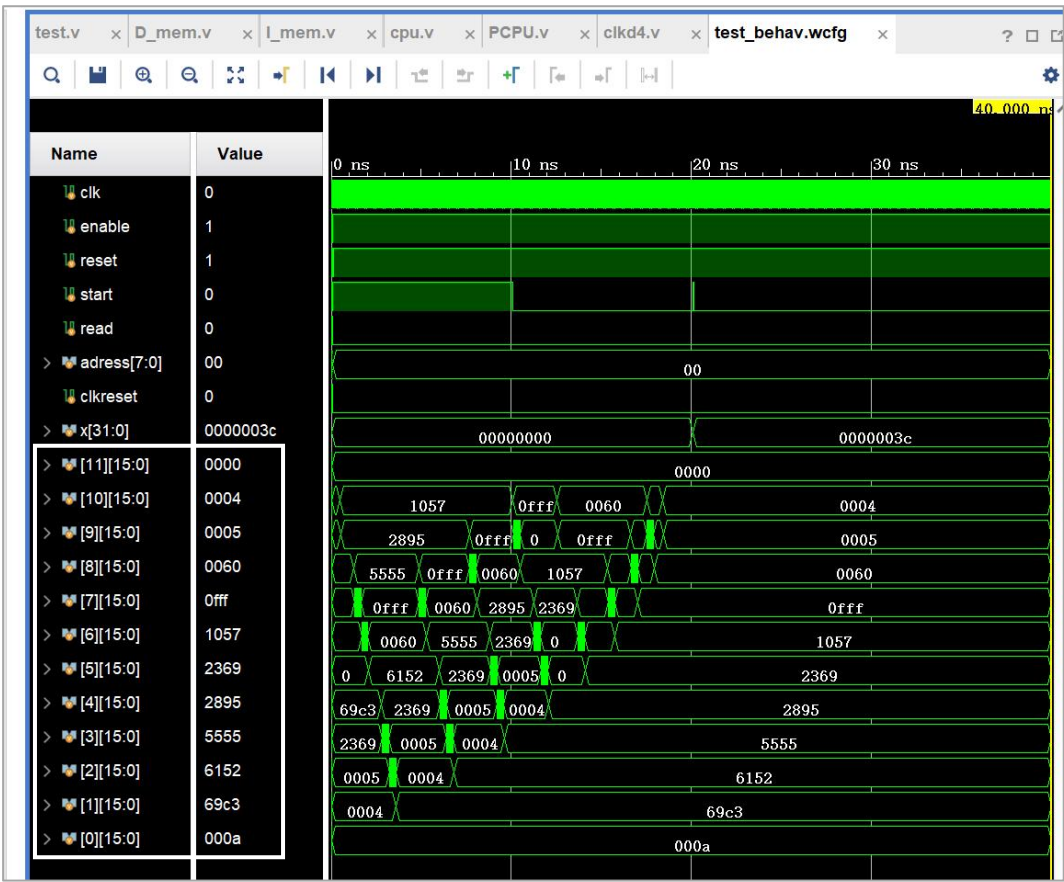


图 3.8 冒泡排序波形图

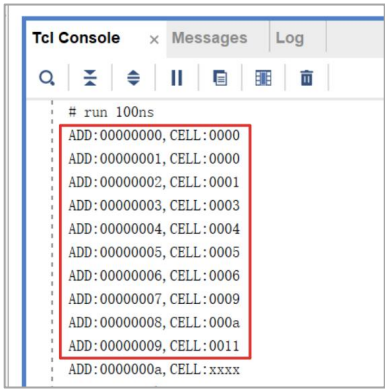
测试得到的波形图如图 3.8 所示，可以看到时序图没有出现不定态，电路工作正常。

且从波形图中显示的数据可以看出，冒泡排序就是从 ADD 大的地址开始，相邻的数据进行比较，然后较大者将被交换到 ADD 小的地址中储存，最后得到的结果就是 ADD=1 储存了最大值 69c3H，ADD=b 储存了最小值 0004H。

图 3.8 中白色框内也显示了 ADD 从 0~11(b)的储存情况，储存的数据与波形图显示的数据一致，也与图 3.7 中的数据一致。符合实验预期。

3.3.5 快速排序测试

测试结果如图 3.9 所示, 可以看见 ADD=2 到 ADD=9 的数字已经按照由小到大的顺序排列。



```
# run 100ns
ADD:00000000, CELL:0000
ADD:00000001, CELL:0000
ADD:00000002, CELL:0001
ADD:00000003, CELL:0003
ADD:00000004, CELL:0004
ADD:00000005, CELL:0005
ADD:00000006, CELL:0006
ADD:00000007, CELL:0009
ADD:00000008, CELL:000a
ADD:00000009, CELL:0011
ADD:0000000a, CELL:xxxx
```

图 3.9 快速排序测试结果

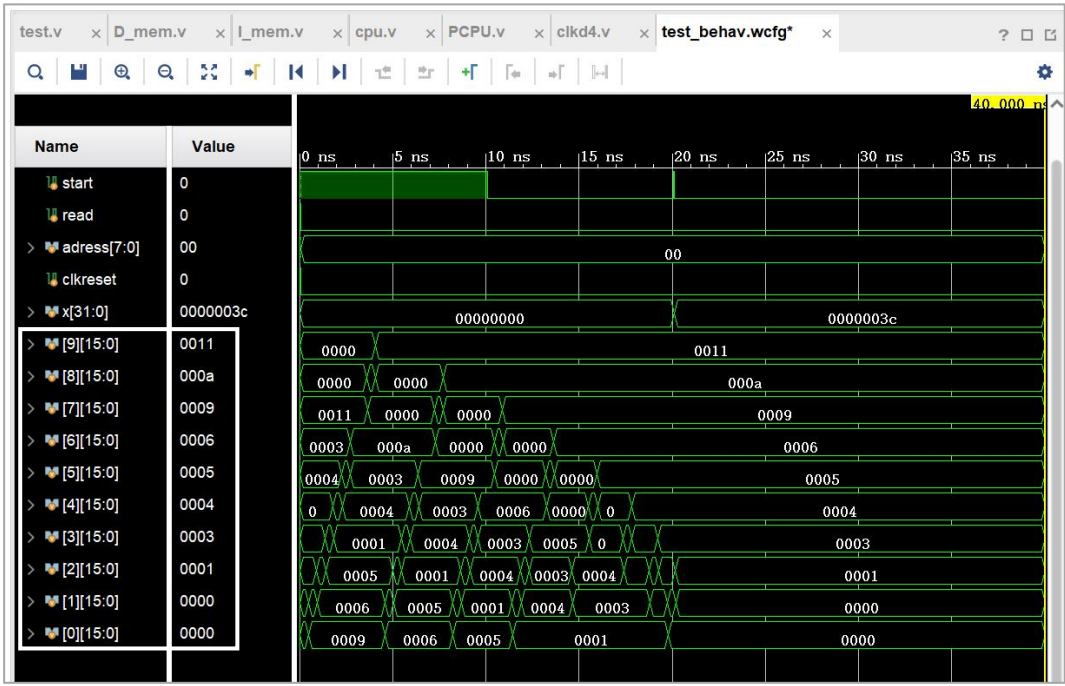


图 3.10 快速排序波形图

测试得到的波形图如图 3.10 所示。可以看到时序图没有出现不定态, 电路工作正常。且从波形图中显示的数据可以看出, 快速排序最后得到的结果就是 ADD=0 储存了最小值 0000H, ADD=9 储存了最小值 0011H。

图 3.10 中白色框内也显示了 ADD 从 0~9 的储存情况, 储存的数据与波形图显示的数据一致, 也与图 3.9 中的数据一致。符合实验预期。

3.3.6 PPA

PPA 的测试结果如图 3.11 所示。由图中显示的数据可以看出, 我们小组设计的 CPU 性能较好, 功耗较低, 为 1.282W, 且这较低的功耗中大部分为动态功耗, 只有 6%的功耗是静态功耗。还可以看出逻辑门所占用的功耗较小, 因此 CPU 的

硬件开销也较小。除此之外，我们还得到该 CPU 的结温较低，热裕度也较良好。

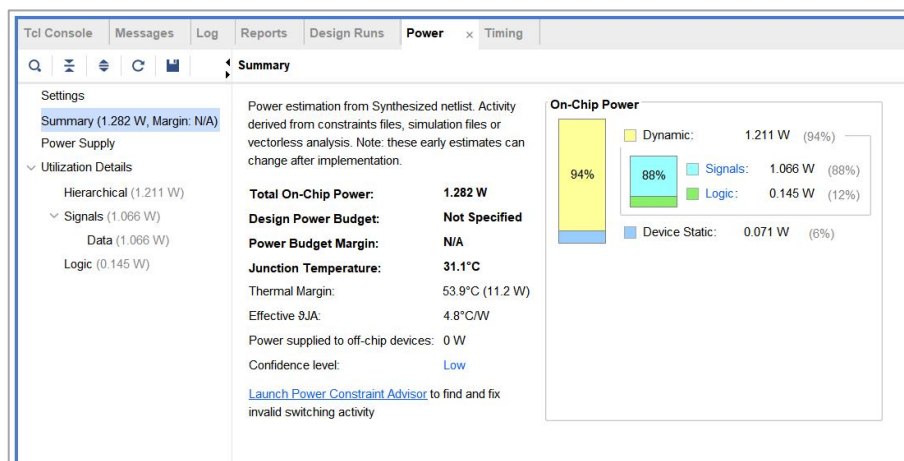


图 3.11 PPA 结果显示

四、实验总结与心得

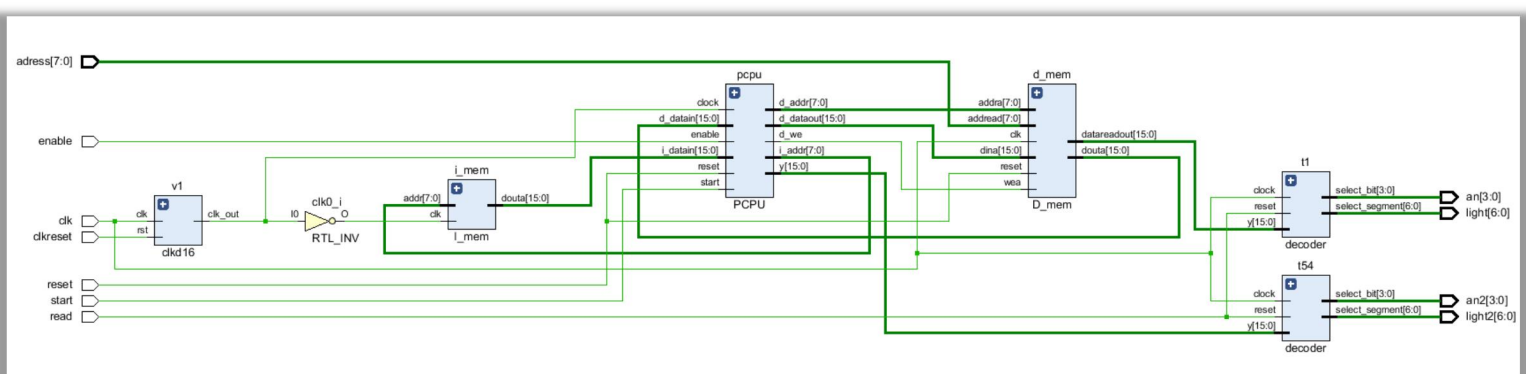
本实验成功利用 vivado 实现了基于 RISC-V 处理器核的 CPU，且通过多组测试可以看出 CPU 可以正常运作功能。它还拥有流水线操作提高效率同时减少功耗的优点。

通过本次实验，我们小组更了解了 CPU 的五级流水线架构，对五级操作 IF、ID、EX、MEM 和 WB 有了更深入的了解，同时也对 CPU 中的零件组成例如寄存器、加法器等有了更多的认识。我们也利用哈佛架构解决了 Hazard 问题，在 PCPU 当中利用了多个 if 语句对不同情况进行讨论从而得到一个好的 CPU 控制。

最后是一些实验过程中的小感想。CPU 的设计是非常难的，原理也很复杂，而且对于本身也就没有学过太多 verilog 语法和没有练习过太多上机实验的我们来说更是难上加难，但是最终还是都克服困难并做出来了一个 CPU。同时也很期待 FPGA 的上板实验。

五、附录

1. RTL 电路图



2. headfile.v

```
`timescale 1ns / 1ps
`ifndef HEADFILE_H_

`define idle 1'b0
`define exec 1'b1
`define NOP 5'b00000
`define HALT 5'b00001
`define LOAD 5'b00010
`define STORE 5'b00011
`define SLL 5'b00100
`define SLA 5'b00101
`define SRL 5'b00110
`define SRA 5'b00111
`define ADD 5'b01000
`define ADDI 5'b01001
`define SUB 5'b01010
`define SUBI 5'b01011
`define CMP 5'b01100
`define AND 5'b01101
`define OR 5'b01110
`define XOR 5'b01111
`define LDIH 5'b10000
`define ADDC 5'b10001
`define SUBC 5'b10010
`define JUMP 5'b11000
`define JMPR 5'b11001
`define BZ 5'b11010
`define BNZ 5'b11011
`define BN 5'b11100
`define BNN 5'b11101
`define BC 5'b11110
`define BNC 5'b11111
`define gr0 3'b000
`define gr1 3'b001
`define gr2 3'b010
`define gr3 3'b011
`define gr4 3'b100
`define gr5 3'b101
`define gr6 3'b110
`define gr7 3'b111

`endif
```

3. clkd4.v

```
`timescale 1ns / 1ps
module clkd16#(
parameter N = 8,
    WIDTH = 2
)
(
    input clk,
    input rst,
    output reg clk_out
);

reg [WIDTH:0]counter;
always @(posedge clk or posedge rst) begin
    if (rst) begin
        counter <= 0;
    end
    else begin
        counter <= counter + 1;
    end
end

always @(posedge clk or posedge rst) begin
    if (rst) begin
        clk_out <= 0;// reset
    end
    else if (counter == N-1) begin
        clk_out <= !clk_out;
    end
end
endmodule
```


4. cpu.v

```

`timescale 1ns / 1ps
module cpu(
input wire clk,enable,reset,read,clkreset,start,
[7:0]adress ,
output[6:0]light,
[3:0]an,
output[6:0]light2,
[3:0]an2
);

wire [15:0] d_datain; //数据输入
wire [15:0] i_datain; //指令输入
wire [7:0] d_addr;    //读取数据地址 s
wire [7:0] i_addr;    //pc 指令地址
wire [15:0] d_dataout;//数据输出到存储
wire d_we;           //写使能信号
reg [15:0] d_data[255:0];
wire [15:0]datarst_memout;
wire [15:0]out;

PCPU pcpu(
    .clock(clk_out6_25MHh),
    .enable(enable),
    .reset(reset),
    .start(start),
    .d_datain(d_datain),
    .i_datain(i_datain),
    .d_addr(d_addr),
    .i_addr(i_addr),
    .d_dataout(d_dataout),
    .d_we(d_we),
    .y(out)
);

I_mem i_mem(
    .addr(i_addr),
    .clk(!clk_out6_25MHh),
    .douta(i_datain)
);

D_mem d_mem(
    .clk(clk),

```

```
.reset(reset),
.wea(d_we),
.addra(d_addr),
.dina(d_dataout),
.douta(d_datain),
.address(address),//数据地址
.datareadout(datarst_memout)//datareadout 是 D_mem 的输出
);

decoder t1(
.clock(clk),
.y(datarst_memout),
.reset(read),
.select_segment(light),
.select_bit(an));

decoder t54(
.clock(clk),
.y(out),
.reset(read),
.select_segment(light2),
.select_bit(an2));

clkdiv v1
( .clk_out(clk_out6_25MHz),
  .rst(clkreset),
  .clk(clk) );

endmodule
```

5. D_mem.v

```

`timescale 1ns / 1ps
module D_mem(
    input wire wea,clk,
    input wire [7:0] addra,
    input wire [15:0] dina,
    input [7:0]addread,
    input reset,
    output reg[15:0] douta,
    output wire [15:0]datareadout//输出
);

reg [15:0] d_mem[255:0];
assign datareadout = d_mem[addread];
always @(posedge clk or negedge reset)
begin
    if(!reset)
        begin
//下面是测试时需要用的数据
            d_mem[0] <= 16'h000a;
            d_mem[1] <= 16'h0009;
            d_mem[2] <= 16'h0006;
            d_mem[3] <= 16'h0005;
            d_mem[4] <= 16'h0001;
            d_mem[5] <= 16'h0004;
            d_mem[6] <= 16'h0003;
            d_mem[7] <= 16'h0011;
            d_mem[8] <= 16'h0000;
            d_mem[9] <= 16'h0000;
        end
    end
    else begin
        if(wea == 1'b1)
            d_mem[addra] <= dina;
        else
            douta <= d_mem[addra];
        end
    end
end
//以下是数据集，测试不同的数据需要选取相应的部分到上面的测试区
//不同的测试功能 I_mem 中也要改到相应的指令集
initial begin
    //复杂指令
/*
    d_mem[0] <= 16'hfffd;

```

```

d_mem[1] <= 16'h0004;
d_mem[2] <= 16'h0005;
d_mem[3] <= 16'hc369;
d_mem[4] <= 16'h69c3;
d_mem[5] <= 16'h0041;
d_mem[6] <= 16'hffff;
d_mem[7] <= 16'h0001;
d_mem[8] <= 16'h0000;
d_mem[9] <= 16'h0000;
d_mem[10] <= 16'h0000;
*/
//最大公约数和最小公倍数
/*
d_mem[0] <= 16'h0000;
d_mem[1] <= 16'h0020; //32
d_mem[2] <= 16'h0018; //24      8 96
*/
//64 位加法器
/*
d_mem[0] <= 16'hfffe;
d_mem[1] <= 16'hfffe;
d_mem[2] <= 16'hfffe;
d_mem[3] <= 16'h0000;
d_mem[4] <= 16'hffff;
d_mem[5] <= 16'hffff;
d_mem[6] <= 16'hffff;
d_mem[7] <= 16'h0000;
*/
//冒泡排序
/*
d_mem[0] <= 16'h000a;
d_mem[1] <= 16'h0004;
d_mem[2] <= 16'h0005;
d_mem[3] <= 16'h2369;
d_mem[4] <= 16'h69c3;
d_mem[5] <= 16'h0060;
d_mem[6] <= 16'h0fff;
d_mem[7] <= 16'h5555;
d_mem[8] <= 16'h6152;
d_mem[9] <= 16'h1057;
d_mem[10] <= 16'h2895;
d_mem[11] <= 16'h0000;
*/
//排序由小到大

```

```
/*  
  d_mem[0] <= 16'h000a;  
  d_mem[1] <= 16'h0009;  
  d_mem[2] <= 16'h0006;  
  d_mem[3] <= 16'h0005;  
  d_mem[4] <= 16'h0001;  
  d_mem[5] <= 16'h0004;  
  d_mem[6] <= 16'h0003;  
  d_mem[7] <= 16'h0011;  
  d_mem[8] <= 16'h0000;  
  d_mem[9] <= 16'h0000;  
*/  
//  
end  
endmodule
```

6. I_mem.v

```

`timescale 1ns / 1ps
`include "headfile.v"
module I_mem(
input wire [7:0] addr,
input clk,
output reg [15:0] douta
);

always @(posedge clk)
begin
/*
    case(addr)
    0:douta <= {'ADDI, `gr1, 4'b0000, 4'b1001};
    1:douta <= {'ADDI, `gr2, 4'b0000, 4'b1001};
    2:douta <= {'JUMP, 11'b000_0000_0101};//jump to start
    3:douta <= {'SUBI, `gr1, 4'd0, 4'd1};//new_round
    4:douta <= {'BZ, `gr7, 4'b0001, 4'b0010};//jump to end
    5:douta <= {'LOAD, `gr3, 1'b0, `gr0, 4'd0};//start
    6:douta <= {'LOAD, `gr4, 1'b0, `gr0, 4'd1};
    7:douta <= {'CMP, 3'd0, 1'b0, `gr3, 1'b0, `gr4};
    8:douta <= {'BN, `gr7, 4'h0, 4'b1011};//jump to NO_op
    9:douta <= {'STORE, `gr3, 1'b0, `gr0, 4'd1};
    10:douta <= {'STORE, `gr4, 1'b0, `gr0, 4'd0};
    11:douta <= {'ADDI, `gr0, 4'b0000, 4'b0001};//NO_OP
    12:douta <= {'CMP, 3'd0, 1'b0, `gr0, 1'b0, `gr2};
    13:douta <= {'BN, `gr7, 4'b0001, 4'b0001};//jump to continue
    14:douta <= {'SUBI, `gr2, 4'd0, 4'd1};
    15:douta <= {'SUB, `gr0, 1'b0, `gr0, 1'b0, `gr0};
    16:douta <= {'JUMP, 11'b000_0000_0011};//jump to new round
    17:douta <= {'JUMP, 11'b000_0000_0101};//jump to start,continue
    18:douta <= {'HALT, 11'd0};//end
    endcase
end
*/

//不同的指令集对应 D_mem 中不同的数据集，使用是删去注释
//-----复杂指令-----
/*
    case(addr)
        0:douta={'ADDI,`gr7,4'd1,4'd0}; // gr7 <= 16'h0010 for store address
        1:douta={'LDIH,`gr1,4'b1011,4'b0110}; // test for LDIH gr1<="16'hb600"
        2:douta={'STORE,`gr1,1'b0,`gr7,4'h0}; // store to mem10

```



```

3:douta={'LOAD','gr1,1'b0,'gr0,4'h0};      // gr1 <= fffd
4:douta={'LOAD','gr2,1'b0,'gr0,4'h1};      // gr2 <= 4
5:douta={'ADDC','gr3,1'b0,'gr1,1'b0,'gr2}; // gr3 <= fffd + 4 + cf(=0) = 1, cf<=1
6:douta={'STORE','gr3,1'b0,'gr7,4'h1};      // store to mem11
7:douta={'ADDC','gr3,1'b0,'gr0,1'b0,'gr2}; // gr3 <= 0 + 4 + cf(=1) = 5, cf<=0
8:douta={'STORE','gr3,1'b0,'gr7,4'h2};      // store to mem12
9:douta={'LOAD','gr1,1'b0,'gr0,4'h2};      // gr1 <= 5
10:douta={'SUBC','gr3,1'b0,'gr1,1'b0,'gr2}; // gr3 <= 5 - 4 + cf(=0) = 1, cf<=0
11:douta={'STORE','gr3,1'b0,'gr7,4'h3};      // store to mem13
12:douta={'SUB','gr3,1'b0,'gr2,1'b0,'gr1}; // gr3 <= 4 - 5 = -1, cf<=1
13:douta={'STORE','gr3,1'b0,'gr7,4'h4};      // store to mem14
14:douta={'SUBC','gr3,1'b0,'gr2,1'b0,'gr1}; // gr3 <= 5 - 4 - cf(=1) = 2, cf<=0
15:douta={'STORE','gr3,1'b0,'gr7,4'h5};      // store to mem15
16:douta={'LOAD','gr1,1'b0,'gr0,4'h3};      // gr1 <= c369
17:douta={'LOAD','gr2,1'b0,'gr0,4'h4};      // gr2 <= 69c3
18:douta={'AND','gr3,1'b0,'gr1,1'b0,'gr2}; // gr3 <= gr1 & gr2 = 4141
19:douta={'STORE','gr3,1'b0,'gr7,4'h6};      // store to mem16
20:douta={'OR','gr3,1'b0,'gr1,1'b0,'gr2};   // gr3 <= gr1 | gr2 = ebeb
21:douta={'STORE','gr3,1'b0,'gr7,4'h7};      // store to mem17
22:douta={'XOR','gr3,1'b0,'gr1,1'b0,'gr2}; // gr3 <= gr1 ^ gr2 = aaaa
23:douta={'STORE','gr3,1'b0,'gr7,4'h8};      // store to mem18
24:douta={'SLL','gr3,1'b0,'gr1,4'h0};      // gr3 <= gr1 < 0
25:douta={'STORE','gr3,1'b0,'gr7,4'h9};      // store to mem19
26:douta={'SLL','gr3,1'b0,'gr1,4'h1};      // gr3 <= gr1 < 1
27:douta={'STORE','gr3,1'b0,'gr7,4'ha};      // store to mem1a
28:douta={'SLL','gr3,1'b0,'gr1,4'h4};      // gr3 <= gr1 < 8
29:douta={'STORE','gr3,1'b0,'gr7,4'hb};      // store to mem1b
30:douta={'SLL','gr3,1'b0,'gr1,4'hf};      // gr3 <= gr1 < 15
31:douta={'STORE','gr3,1'b0,'gr7,4'hc};      // store to mem1c
32:douta={'SRL','gr3,1'b0,'gr1,4'h0};      // gr3 <= gr1 > 0
33:douta={'STORE','gr3,1'b0,'gr7,4'hd};      // store to mem1d
34:douta={'SRL','gr3,1'b0,'gr1,4'h1};      // gr3 <= gr1 > 1
35:douta={'STORE','gr3,1'b0,'gr7,4'he};      // store to mem1e
36:douta={'SRL','gr3,1'b0,'gr1,4'h8};      // gr3 <= gr1 > 8
37:douta={'STORE','gr3,1'b0,'gr7,4'hf};      // store to mem1f
38:douta={'SRL','gr3,1'b0,'gr1,4'hf};      // gr3 <= gr1 > 15
39:douta={'ADDI','gr7,4'd1,4'd0};           // gr7 <= 16'h20 for store address
40:douta={'STORE','gr3,1'b0,'gr7,4'h0};      // store to mem20
41:douta={'SLA','gr3,1'b0,'gr1,4'h0};      // gr3 <= gr1 < 0
42:douta={'STORE','gr3,1'b0,'gr7,4'h1};      // store to mem21
43:douta={'SLA','gr3,1'b0,'gr1,4'h1};      // gr3 <= gr1 < 1
44:douta={'STORE','gr3,1'b0,'gr7,4'h2};      // store to mem22
45:douta={'SLA','gr3,1'b0,'gr1,4'h8};      // gr3 <= gr1 < 8
46:douta={'STORE','gr3,1'b0,'gr7,4'h3};      // store to mem23

```

```

47:douta={'SLA,'gr3,1'b0,'gr1,4'hf}; // gr3 <= gr1 < 15
48:douta={'STORE,'gr3,1'b0,'gr7,4'h4}; // store to mem24
49:douta={'SLA,'gr3,1'b0,'gr2,4'h0}; // gr3 <= gr1 < 0
50:douta={'STORE,'gr3,1'b0,'gr7,4'h5}; // store to mem25
51:douta={'SLA,'gr3,1'b0,'gr2,4'h1}; // gr3 <= gr1 < 1
52:douta={'STORE,'gr3,1'b0,'gr7,4'h6}; // store to mem26
53:douta={'SLA,'gr3,1'b0,'gr2,4'h8}; // gr3 <= gr1 < 8
54:douta={'STORE,'gr3,1'b0,'gr7,4'h7}; // store to mem27
55:douta={'SLA,'gr3,1'b0,'gr2,4'hf}; // gr3 <= gr1 < 15
56:douta={'STORE,'gr3,1'b0,'gr7,4'h8}; // store to mem28
57:douta={'SRA,'gr3,1'b0,'gr1,4'h0}; // gr3 <= gr1 > 0
58:douta={'STORE,'gr3,1'b0,'gr7,4'h9}; // store to mem29
59:douta={'SRA,'gr3,1'b0,'gr1,4'h1}; // gr3 <= gr1 > 1
60:douta={'STORE,'gr3,1'b0,'gr7,4'ha}; // store to mem2a
61:douta={'SRA,'gr3,1'b0,'gr1,4'h8}; // gr3 <= gr1 > 8
62:douta={'STORE,'gr3,1'b0,'gr7,4'hb}; // store to mem2b
63:douta={'SRA,'gr3,1'b0,'gr1,4'hf}; // gr3 <= gr1 > 15
64:douta={'STORE,'gr3,1'b0,'gr7,4'hc}; // store to mem2c
65:douta={'SRA,'gr3,1'b0,'gr2,4'h0}; // gr3 <= gr1 > 0
66:douta={'STORE,'gr3,1'b0,'gr7,4'hd}; // store to mem2d
67:douta={'SRA,'gr3,1'b0,'gr2,4'h1}; // gr3 <= gr1 > 1
68:douta={'STORE,'gr3,1'b0,'gr7,4'he}; // store to mem2e
69:douta={'SRA,'gr3,1'b0,'gr2,4'h8}; // gr3 <= gr1 > 8
70:douta={'STORE,'gr3,1'b0,'gr7,4'hf}; // store to mem2f
71:douta={'ADDI,'gr7,4'd1,4'd0}; // gr7 <= 16'h30 for store address
72:douta={'SRA,'gr3,1'b0,'gr2,4'hf}; // gr3 <= gr1 > 15
73:douta={'STORE,'gr3,1'b0,'gr7,4'h0}; // store to mem30
74:douta={'LOAD,'gr1,1'b0,'gr0,4'h5}; // gr1 <= 41
75:douta={'LOAD,'gr2,1'b0,'gr0,4'h6}; // gr2 <= ffff
76:douta={'LOAD,'gr3,1'b0,'gr0,4'h7}; // gr3 <= 1
77:douta={'JUMP,'3'd0,8'h4f}; // jump to 4f
78:douta={'STORE,'gr7,1'b0,'gr7,4'h1}; // store to mem31
79:douta={'JMPR,'gr1,8'h10}; // jump to 41+10 = 51
80:douta={'STORE,'gr7,1'b0,'gr7,4'h2}; // store to mem32
81:douta={'ADD,'gr4,1'b0,'gr2,1'b0,'gr3}; // gr4<= ffff + 1,cf<=1
82:douta={'BNC,'gr1,8'h28}; // if(cf==0) jump to 69
83:douta={'BC,'gr1,8'h14}; // if(cf==1) jump to 55
84:douta={'STORE,'gr7,1'b0,'gr7,4'h3}; // store to mem33
85:douta={'ADD,'gr4,1'b0,'gr3,1'b0,'gr3}; // gr4<= 1 + 1 , cf<=0
86:douta={'BC,'gr1,8'h28}; // if(cf==1) jump to 69
87:douta={'BNC,'gr1,8'h18}; // if(cf==0) jump to 59
88:douta={'STORE,'gr7,1'b0,'gr7,4'h4}; // store to mem34
89:douta={'CMP,'3'd0,1'b0,'gr3,1'b0,'gr3}; // 1-1=0 , zf<=1,nf<=0
90:douta={'BNZ,'gr1,8'h28}; // if(zf==0) jump to 69

```

```

91:douta={'BZ,'gr1,8'h1c};           // if(zf==1) jump to 5d
92:douta={'STORE,'gr7,1'b0,'gr7,4'h5}; // store to mem35
93:douta={'CMP, 3'd0,1'b0,'gr4,1'b0,'gr3}; // 2-1=1 , zf<=0,nf<=0
94:douta={'BZ,'gr1,8'h28};           // if(zf==1) jump to 69
95:douta={'BNZ,'gr1,8'h20};           // if(zf==0) jump to 61
96:douta={'STORE,'gr7,1'b0,'gr7,4'h6}; // store to mem36
97:douta={'CMP, 3'd0,1'b0,'gr3,1'b0,'gr4}; // 1-2=-1, nf<=1,zf<=0
98:douta={'BNN,'gr1,8'h28};           // if(nf==0) jump to 69
99:douta={'BN,'gr1,8'h24};           // if(nf==1) jump to 65
100:douta={'STORE,'gr7,1'b0,'gr7,4'h7}; // store to mem37
101:douta={'CMP, 3'd0,1'b0,'gr4,1'b0,'gr3}; // 2-1=1, nf<=0,zf<=0
102:douta={'BN,'gr1,8'h28};           // if(nf==1) jump to 69
103:douta={'BNN,'gr1,8'h27};          // if(nf==0) jump to 68
104:douta={'STORE,'gr7,1'b0,'gr7,4'h8}; // store to mem38
105:douta={'HALT, 11'd0};             // STOP

endcase
end
*/

//-----最大公约数和最小公倍数-----

/*
case(addr)
0:douta <= {'LOAD, 'gr1, 1'b0, 'gr0, 4'b0001}; //gr1=0009
1:douta <= {'LOAD, 'gr2, 1'b0, 'gr0, 4'b0010}; //gr2 = 0003
2:douta <= {'ADD, 'gr3, 1'b0, 'gr0, 1'b0, 'gr1}; //gr3=0009 0006 0003
3:douta <= {'SUB, 'gr1, 1'b0, 'gr1, 1'b0, 'gr2}; //gr1=0006 0003 0
4:douta <= {'BZ, 'gr0, 8'b0000_1001}; //no no jump to 09
5:douta <= {'BNN, 'gr0, 8'b0000_0010}; //jump to 2 jump to 2
6:douta <= {'ADD, 'gr1, 1'b0, 'gr0, 1'b0, 'gr2};
7:douta <= {'ADD, 'gr2, 1'b0, 'gr0, 1'b0, 'gr3};
8:douta <= {'JUMP, 11'b000_0000_0010};
9:douta <= {'STORE, 'gr2, 1'b0, 'gr0, 4'b0011}; //d[3:douta=0003
10:douta <= {'LOAD, 'gr1, 1'b0, 'gr0, 4'b0001}; //gr1 = 0009
11:douta <= {'LOAD, 'gr2, 1'b0, 'gr0, 4'b0010}; //gr2 = 0003
12:douta <= {'ADDI, 'gr4, 8'h1}; // gr4=1
13:douta <= {'SUB, 'gr2, 1'b0, 'gr2, 1'b0, 'gr3}; //gr2 = 0
14:douta <= {'BZ, 'gr0, 8'b0001_0000}; //jump to 16
15:douta <= {'JUMP, 11'b000_0000_1100}; //
16:douta <= {'SUBI, 'gr4, 8'h1}; // gr4 = 0 -1
17:douta <= {'BN, 'gr0, 8'b0001_0100}; //no jump to 20
18:douta <= {'ADD, 'gr5, 1'b0, 'gr5, 1'b0, 'gr1}; //gr5 = 0009
19:douta <= {'JUMP, 11'b000_0001_0000}; //jump to 16
20:douta <= {'STORE, 'gr5, 1'b0, 'gr0, 4'b0100}; //d[4:douta=0009
21:douta <= {'LOAD, 'gr1, 1'b0, 'gr0, 4'b0011}; //gr1=0003
22:douta <= {'LOAD, 'gr2, 1'b0, 'gr0, 4'b0100}; //gr2=0009

```

```

23:douta <= {'HALT, 11'b000_0000_0000};
endcase
end
*/

//-----64 位加法器-----
/*
case(addr)
0:douta=16'h4c04 ; //{`ADDI,`gr4,8'h04}; gr4=0004
1:douta=16'h1100 ; //{`LOAD, `gr1, 1'b0, `gr0, 4'b0000}; gr1 = fffe
2:douta=16'h1204 ; //{`LOAD, `gr2, 1'b0, `gr0, 4'b0100}; gr2 = ffff
3:douta=16'h4312 ; //{`ADD, `gr3, 1'b0, `gr1, 1'b0, `gr2}; gr3 = fffd
4:douta=16'hfd06 ; //{`BNC, `gr5, 8'b0000_0110}; no
5:douta=16'h4e01 ; //{`ADDI,`gr6,8'h01}; gr6=0001
6:douta=16'h4337 ; //{`ADD, `gr3, 1'b0, `gr3, 1'b0, `gr7}; gr3 = fffd
7:douta=16'hfd0b ; //{`BNC, `gr5, 8'b0000_1011};jump to 11
8:douta=16'h5e00 ; //{`SUBI, `gr6, 8'h0};
9:douta=16'hdd0b ; //{`BNZ, `gr5, 8'h0b};
10:douta=16'h4e01 ; //{`ADDI,`gr6,8'h01};
11:douta=16'h5777 ; //{`SUB, `gr7, 1'b0, `gr7, 1'b0, `gr7}; gr7 = 0
12:douta=16'h4776 ; //{`ADD, `gr7, 1'b0, `gr7, 1'b0, `gr6}; gr7 = 0001
13:douta=16'h5666 ; //{`SUB, `gr6, 1'b0, `gr6, 1'b0, `gr6}; gr6 = 0
14:douta=16'h1b08 ; //{`STORE, `gr3, 1'b0, `gr0, 4'b1000}; d[8:douta=fffd
15:douta=16'h4801 ; //{`ADDI,`gr0,8'h01}; gr0=0001
16:douta=16'h6004 ; //{`CMP, 3'b0,1'b0,`gr0,1'b0,`gr4}; 1<4
17:douta=16'he501 ; //{`BN, `gr5, 8'b0000_0001}; jump to 1
18:douta=16'h0800 ; //halt
endcase
end
*/

//-----冒泡排序-----
/*
case(addr)
0:douta <= {'LOAD, `gr3, 1'b0, `gr0, 4'b0000}; //gr3=000a;
1:douta <= {'SUBI, `gr3, 4'd0, 4'd2}; //gr3=0008
2:douta <= {'ADD, `gr1, 1'b0, `gr0, 1'b0, `gr0}; //gr1=0
3:douta <= {'ADD, `gr2, 1'b0, `gr3, 1'b0, `gr0}; //gr2=0008
4:douta <= {'LOAD, `gr4, 1'b0, `gr2, 4'd1}; //gr4=d[9:douta=1057
5:douta <= {'LOAD, `gr5, 1'b0, `gr2, 4'd2}; //gr5=d[10:douta=2895
6:douta <= {'CMP, 3'd0, 1'b0, `gr5, 1'b0, `gr4}; //d[10:douta > d[9:douta
7:douta <= {'BN, `gr0, 4'b0000, 4'b1010}; //no
8:douta <= {'STORE, `gr4, 1'b0, `gr2, 4'd2}; //d[10:douta=gr4
9:douta <= {'STORE, `gr5, 1'b0, `gr2, 4'd1}; //d[9:douta=gr5;
10:douta <= {'SUBI, `gr2, 4'd0, 4'd1}; //gr2=0007

```

```

11:douta <= {'CMP, 3'd0, 1'b0, `gr2, 1'b0, `gr1}; //gr2>gr1
12:douta <= {'BNN, `gr0, 4'h0, 4'b0100}; //
13:douta <= {'ADDI, `gr1, 4'd0, 4'd1};
14:douta <= {'CMP, 3'd0, 1'b0, `gr3, 1'b0, `gr1};
15:douta <= {'BNN, `gr0, 4'h0, 4'b0011};
16:douta <= {'HALT, 11'd0};

endcase

end

*/

//-----sort 排序由小到大-----

/*
case(addr)
0:douta <= {'ADDI, `gr1, 4'b0000, 4'b1001};
1:douta <= {'ADDI, `gr2, 4'b0000, 4'b1001};
2:douta <= {'JUMP, 11'b000_0000_0101};//jump to start
3:douta <= {'SUBI, `gr1, 4'd0, 4'd1};//new_round
4:douta <= {'BZ, `gr7, 4'b0001, 4'b0010};//jump to end
5:douta <= {'LOAD, `gr3, 1'b0, `gr0, 4'd0};//start
6:douta <= {'LOAD, `gr4, 1'b0, `gr0, 4'd1};
7:douta <= {'CMP, 3'd0, 1'b0, `gr3, 1'b0, `gr4};
8:douta <= {'BN, `gr7, 4'h0, 4'b1011};//jump to NO_op
9:douta <= {'STORE, `gr3, 1'b0, `gr0, 4'd1};
10:douta <= {'STORE, `gr4, 1'b0, `gr0, 4'd0};
11:douta <= {'ADDI, `gr0, 4'b0000, 4'b0001};//NO_OP
12:douta <= {'CMP, 3'd0, 1'b0, `gr0, 1'b0, `gr2};
13:douta <= {'BN, `gr7, 4'b0001, 4'b0001};//jump to continue
14:douta <= {'SUBI, `gr2, 4'd0, 4'd1};
15:douta <= {'SUB, `gr0, 1'b0, `gr0, 1'b0, `gr0};
16:douta <= {'JUMP, 11'b000_0000_0011};//jump to new round
17:douta <= {'JUMP, 11'b000_0000_0101};//jump to start,continue
18:douta <= {'HALT, 11'd0};//end

endcase

end

// */

endmodule

```

7. PCPU.v

```

`timescale 1ns / 1ps
`include "headfile.v"
module PCPU(input wire reset, enable, clock, start,
    input wire [15:0] d_datain, //数据输入
    i_datain, //指令输入
    output wire [7:0] d_addr, //读取数据地址
    output wire [7:0] i_addr, //读取 pc 指令地址
    output wire [15:0] d_dataout, //数据输出储存
    output wire d_we, //写使能信号
    output wire [15:0] y
);
reg state,next_state; //状态机
reg [7:0] pc; //pc 指令
reg [15:0] id_ir,ex_ir,mem_ir,wb_ir; //指令流水线,id 译码,ex 执行,mem 存储,wb 写回
reg [15:0] reg_A, reg_B, reg_C, reg_C1, smdr, smdr1, ALUo; //CPU 内部寄存器
reg dw, zf, nf, cf, cin; //flags, 状态标志位
reg [15:0] gr[0:7]; //寄存器堆, 二维数组
assign d_dataout = smdr1; //数据输出
assign d_we = dw; //使能信号
assign d_addr = reg_C[7:0];
assign i_addr = pc;

//***** CPU control *****/
always @(posedge clock or negedge reset)
//状态机控制 CPU 运行与停止
begin
    if (!reset)
        state <= `idle;
        //idle 为停止, exec 为运行
    else
        state <= next_state; //运行下一个状态
    end
always @(*)
begin
    case (state)
        `idle :
            if ((enable == 1'b1) && (start == 1'b1)) //继续运行
                next_state <= `exec; //继续运行
            else
                next_state <= `idle; //否则停止运行
        `exec :
            if ((enable == 1'b0) || (wb_ir[15:11] == `HALT)) //HALT 停止运行

```



```

        next_state <= `idle;//idle 为停止
    else
        next_state <= `exec;//否则接着运行
    endcase
end

/***** IF *****/
//取指阶段，读取与选择 PC 中的指令
always @(posedge clock or negedge reset)
begin
    if (!reset)//重置清零
    begin
        id_ir <= 16'b0000_0000_0000_0000;pc <= 8'b0000_0000;
    end
    else if (state ==`exec)//运行状态
    begin
        if((ex_ir[15:11] == `BZ && zf == 1'b1) || (ex_ir[15:11] == `BN && nf == 1'b1)
        ||(ex_ir[15:11] == `BNZ && zf == 1'b0) || (ex_ir[15:11] == `BNN && nf == 1'b0)
        ||(ex_ir[15:11] == `BC && cin == 1'b1) || (ex_ir[15:11] == `BNC && cin == 1'b0)
        || ex_ir[15:11] == `JMPR)//已计算目标指令，符合 flags 要求，然后跳转
        begin
            pc <= ALUo[7:0];//pc 指令跳转到目标指令
            id_ir <= 16'b0;//清空当前位置指令
        end
        else if(id_ir[15:11] == `JUMP)//如果是 JUMP 指令则直接跳转
        begin
            pc <= id_ir[7:0]; //跳转到{val2,val3}
            id_ir <= 16'b0;//清空当前位置指令
        end
        else if(id_ir[15:11] == `HALT) //HALT 停止指令
        begin
            pc <= pc;id_ir <= id_ir;
        end
        //开始处理冒险情况
        else if((id_ir[15:11] == `LOAD)&&(i_datain[15:11]!=`JUMP)&&(i_datain[15:11]!=`NOP)
        &&(i_datain[15:11]!=`HALT)&&(i_datain[15:11]!=`LOAD))//可能存在 LOAD 冒险
        begin //寄存器 r1 冒险
            if((id_ir[10:8]==i_datain[2:0])&&((i_datain[15:11]==`ADD)||i_datain[15:11]==`ADDC)
            ||(i_datain[15:11]==`SUB)||i_datain[15:11]==`SUBC)||i_datain[15:11]==`CMP)))
            begin
                pc <= pc;id_ir <= 16'b0;//流水线冒泡(BUBBLE)，停止
            end
            //寄存器 r2 冒险
        else

```

```

        if((id_ir[10:8]==i_datain[6:4])&&((i_datain[15:11]=='STORE')||(i_
        datain[15:11]=='ADD)
        ||(i_datain[15:11]=='ADDC')||(i_datain[15:11]=='SUB')||(i_datain[15:11]=='SUBC')|
        |(i_datain[15:11]=='CMP')||(i_datain[15:11]=='SLL')||(i_datain[15:11]=='SRL')|
        |(i_datain[15:11]=='SLA')||(i_datain[15:11]=='SRA)))
        begin
            pc <= pc;id_ir <= 16'b0;//流水线冒泡(BUBBLE), 停止
        end
        //寄存器 r3 冒险
    else
        if((id_ir[10:8]==i_datain[10:8])&&((i_datain[15:11]=='STORE')||(i_dat
        ain[15:11]=='LDIH)
        ||(i_datain[15:11]=='SUBI')||(i_datain[15:11]=='JMPR')||(i_datain[15:11]=='BZ')||(i_datain
        [15:11]=='BNZ)
        ||(i_datain[15:11]=='BN')||(i_datain[15:11]=='BNN')||(i_datain[15:11]=='BNC')||(i_datain
        [15:11]=='BNC)))
        begin
            pc <= pc;id_ir <= 16'b0;//流水线冒泡(BUBBLE), 停止
        end
    else
        begin
            pc <= pc + 1'b1;id_ir <= i_datain;//不出现上述情况则读取下一条指令
        end
    end
    else
        begin
            pc <= pc + 1'b1; id_ir <= i_datain;//不存在冒险则读取下一条指令
        end
    end
    end
    else if(state=='idle')//停止状态
        begin
            id_ir <= id_ir; pc <= pc;
        end
    end
end

/***** ID *****/
//译码阶段, 对指令进行译码, 控制寄存器工作
always @(posedge clock or negedge reset)
begin
    if (!reset )//重置清零
    begin
        ex_ir <= 16'b0000_0000_0000_0000;
        reg_A <= 16'b0000_0000_0000_0000;
        reg_B <= 16'b0000_0000_0000_0000;
    end
end

```

```

        smdr <= 16'b0000_0000_0000_0000;
    end
    else if(state == `exec && ((ex_ir[15:11]==`BZ && zf==1'b1) ||(ex_ir[15:11]==`BNZ && zf==1'b0)
        ||(ex_ir[15:11]==`BN && nf==1'b1) || (ex_ir[15:11]==`BNN && nf==1'b0) ||(ex_ir[15:11]==`BC
            && cin==1'b1)
        ||(ex_ir[15:11]==`BNC && cin==1'b0) || ex_ir[15:11]==`JMPR||ex_ir[15:11]==`JUMP))
        ex_ir <= 16'b0;//清空当前流水线
    else if (state == `exec)
        begin
            ex_ir <= id_ir;//传递指令
            //处理寄存器 regA, r1=r1+{val2,val3}
            if ((id_ir[15:11] == `BZ) || (id_ir[15:11] == `BNZ) || (id_ir[15:11] == `BN) || (id_ir[15:11] ==
                `BNN)
                || (id_ir[15:11] == `BC) || (id_ir[15:11] == `BNC) || (id_ir[15:11] == `LDIH) || (id_ir[15:11]
                == `ADDI)
                || (id_ir[15:11] == `SUBI) || (id_ir[15:11] == `JMPR))
            begin
                //处理冒险 r1, 未写回, 数据直接从 ALU 取
                if(id_ir[10:8]==ex_ir[10:8]&&(ex_ir[15:11] == `ADD || ex_ir[15:11] == `LDIH || ex_ir[15:11]
                    == `ADDI ||
                    ex_ir[15:11] == `SUB || ex_ir[15:11] == `SUBI || ex_ir[15:11] == `ADDC || ex_ir[15:11] ==
                    `SUBC ||
                    ex_ir[15:11] == `SLL || ex_ir[15:11] == `SRL || ex_ir[15:11] == `SLA || ex_ir[15:11] ==
                    `SRA ))
                reg_A <= ALUo;//数据直接从 ALU 取
                else if(id_ir[10:8] == mem_ir[10:8]&&(mem_ir[15:11] == `ADD || mem_ir[15:11] == `LDIH
                    || mem_ir[15:11] == `ADDI
                    || mem_ir[15:11] == `SUB || mem_ir[15:11] == `SUBI || mem_ir[15:11] == `ADDC||
                    mem_ir[15:11] == `SUBC
                    || mem_ir[15:11] == `SLL || mem_ir[15:11] == `SRL || mem_ir[15:11] == `SLA
                    || mem_ir[15:11] == `SRA
                    || mem_ir[15:11] == `LOAD))
                begin
                    if(mem_ir[15:11]==`LOAD)//若为 LOAD 指令则从外部内存输入
                    reg_A <= d_datain;//数据直接从外部内存取
                else
                    reg_A <= reg_C;//否则数据从寄存器 C 中取
                end
            end
            else if(wb_ir[10:8] == id_ir[10:8]&&(wb_ir[15:11] == `ADD || wb_ir[15:11] == `LDIH ||
                wb_ir[15:11] == `ADDI
                || wb_ir[15:11] == `SUB || wb_ir[15:11] == `SUBI || wb_ir[15:11] == `ADDC ||
                wb_ir[15:11] == `SUBC
                || wb_ir[15:11] == `SLL || wb_ir[15:11] == `SRL || wb_ir[15:11] == `SLA ||
                wb_ir[15:11] == `SRA ||

```

```

        wb_ir[15:11] == `LOAD ))
    reg_A <= reg_C1; //数据从寄存器 C1 中取
else
    reg_A <= gr[(id_ir[10:8]);]//gr1
end
//r1=r2#r3 or r1=r2#val
else if(id_ir[15:11] == `LOAD || id_ir[15:11] == `STORE || id_ir[15:11] == `ADD || id_ir[15:11]
== `SUB
    || id_ir[15:11] == `ADDC || id_ir[15:11] == `SUBC || id_ir[15:11] == `CMP || id_ir[15:11]
== `SLL
    || id_ir[15:11] == `SRL || id_ir[15:11] == `SLA || id_ir[15:11] == `SRA)
begin
    if( (ex_ir[15:11] == `ADD || ex_ir[15:11] == `LDIH || ex_ir[15:11] == `ADDI ||
ex_ir[15:11] == `SUB
        || ex_ir[15:11] == `SUBI || ex_ir[15:11] == `ADDC || ex_ir[15:11] == `SUBC ||
ex_ir[15:11] == `SLL
        || ex_ir[15:11] == `SRL || ex_ir[15:11] == `SLA || ex_ir[15:11] == `SRA )&&
ex_ir[10:8] == id_ir[6:4])
        reg_A <= ALUo;
    else if( (mem_ir[15:11] == `ADD || mem_ir[15:11] == `LDIH || mem_ir[15:11] == `ADDI
        || mem_ir[15:11] == `SUB || mem_ir[15:11] == `SUBI || mem_ir[15:11] ==
`ADDC
        || mem_ir[15:11] == `SUBC || mem_ir[15:11] == `SLL || mem_ir[15:11] == `SRL
        || mem_ir[15:11] == `SLA || mem_ir[15:11] == `SRA || mem_ir[15:11] == `LOAD)
        && mem_ir[10:8] == id_ir[6:4])
        if(mem_ir[15:11]==`LOAD)
            reg_A <= d_datain;
        else
            reg_A <= reg_C;
    else if( (wb_ir[15:11] == `ADD || wb_ir[15:11] == `LDIH || wb_ir[15:11] == `ADDI
|| wb_ir[15:11] == `SUB
        || wb_ir[15:11] == `SUBI || wb_ir[15:11] == `ADDC || wb_ir[15:11] == `SUBC
|| wb_ir[15:11] == `SLL ||
        wb_ir[15:11] == `SRL || wb_ir[15:11] == `SLA || wb_ir[15:11] == `SRA ||
wb_ir[15:11] == `LOAD )&&
        wb_ir[10:8] == id_ir[6:4])
        reg_A <= reg_C1;
    else
        reg_A <= gr[id_ir[6:4]]; //gr2
    end
//flush
else if((mem_ir[15:11]==`BZ && zf==1'b1) || (mem_ir[15:11]==`BNZ && zf==1'b0)
        || (mem_ir[15:11]==`BN && nf==1'b1) || (mem_ir[15:11]==`BNN && nf==1'b0)
        || (mem_ir[15:11]==`BC && cin==1'b1)

```

```

        ||(mem_ir[15:11]=='BNC && cin==1'b0) || mem_ir[15:11]=='JMPPR)
            reg_A <= 16'b0;//跳转, 清空寄存器
        else if(id_ir[15:11] == `JUMP)
            reg_A <= 16'b0;//跳转, 清空寄存器
        else
            reg_A <= gr[id_ir[6:4]];//读入 gr2
        //处理 regB
        if (id_ir[15:11] == `LOAD || id_ir[15:11] == `STORE || id_ir[15:11] == `SLL || (id_ir[15:11] ==
`SRL)
            || (id_ir[15:11] == `SLA) || (id_ir[15:11] == `SRA))
            reg_B <= {12'b0000_0000_0000, id_ir[3:0]};//处理 val3, 按位运算
        else if ((id_ir[15:11] == `BZ) || (id_ir[15:11] == `BNZ) || (id_ir[15:11] == `BN) || (id_ir[15:11]
== `BNN)
            || (id_ir[15:11] == `BC) || (id_ir[15:11] == `BNC)
            || (id_ir[15:11] == `ADDI) || (id_ir[15:11] == `SUBI) || (id_ir[15:11] == `JMPPR))
            reg_B <= {8'b0000_0000, id_ir[7:0]};//处理{00000000,value2,value3}, 按位运算
        else if(id_ir[15:11] == `LDIH)
            reg_B <= {id_ir[7:0], 8'b0000_0000};//处理{val2,val3,00000000}, 按位运算
        //r1=r2#r3
        else if(id_ir[15:11] == `ADD || id_ir[15:11] == `SUB || id_ir[15:11] == `ADDC || id_ir[15:11]
== `SUBC
            || id_ir[15:11] == `CMP )
            begin
                if(((ex_ir[15:11]==`ADD)||((ex_ir[15:11]==`LDIH)||((ex_ir[15:11]==`ADDI)||((ex_ir[15:11]
==`SUB)
                ||(ex_ir[15:11]==`SUBI)||((ex_ir[15:11]==`ADDC)||((ex_ir[15:11]==`SUBC)||((ex_ir[15:11]
]==`AND)
                ||(ex_ir[15:11]==`OR)||((ex_ir[15:11]==`XOR)||((ex_ir[15:11]==`SLL)||((ex_ir[15:11]==`S
RL)
                ||(ex_ir[15:11]==`SLA)||((ex_ir[15:11]==`SRA))&&(ex_ir[10:8]==id_ir[2:0]))
                //该阶段与 LOAD 指令没有关系
                reg_B<=ALUo;//此时 ALU 的值(ALUo)会传送至 reg_B 中
            else
                if(((mem_ir[15:11]==`ADD)||((mem_ir[15:11]==`LDIH)||((mem_ir[15:11]==`ADDI)||((mem_ir[15:11]==`SUB)
                ||(mem_ir[15:11]==`SUBI)||((mem_ir[15:11]==`ADDC)||((mem_ir[15:11]==`SUBC)||((me
m_ir[15:11]==`AND)
                ||(mem_ir[15:11]==`OR)||((mem_ir[15:11]==`XOR)||((mem_ir[15:11]==`SLL)||((mem_ir[1
5:11]==`SRL)
                ||(mem_ir[15:11]==`SLA)||((mem_ir[15:11]==`SRA)||((mem_ir[15:11]==`LOAD))&&(me
m_ir[10:8]==id_ir[2:0]))
                begin
                    if(mem_ir[15:11]==`LOAD)//若为 LOAD 指令则从外部内存输入
                        reg_B<=d_datain;//此时外部内存的值(d_datain)会传送至 reg_B 中
                else

```

```

        reg_B<=reg_C;//此时 reg_C 会传送至 reg_B 中
    end
    else
if(((wb_ir[15:11]==`ADD)||((wb_ir[15:11]==`LDIH)||((wb_ir[15:11]==`ADDI)||((wb_ir[15:11]==`SUB)
        ||((wb_ir[15:11]==`SUBI)||((wb_ir[15:11]==`ADDC)||((wb_ir[15:11]==`SUBC)||((wb_ir[15:
11]==`AND)
        ||((wb_ir[15:11]==`OR)||((wb_ir[15:11]==`XOR)||((wb_ir[15:11]==`SLL)||((wb_ir[15:11]==
`SRL)
        ||((wb_ir[15:11]==`SLA)||((wb_ir[15:11]==`SRA)||((wb_ir[15:11]==`LOAD))&&(wb_ir[10:
8]==id_ir[2:0]))

        reg_B<=reg_C1;//此时 reg_C1 值会传送至 reg_B 中
    else
        reg_B <= gr[id_ir[2:0]];//读取 gr2
    end
    //处理跳转指令
    else if((mem_ir[15:11]==`BZ && zf==1'b1) ||(mem_ir[15:11]==`BNZ &&
zf==1'b0)||((mem_ir[15:11]==`BN && nf==1'b1)
        || (mem_ir[15:11]==`BNN && nf==1'b0) || (mem_ir[15:11]==`BC && cin==1'b1)
||((mem_ir[15:11]==`BNC && cin==1'b0)
        || mem_ir[15:11]==`JMPR)
        reg_B <= 16'bx;//跳转, 清空寄存器
    else if(id_ir[15:11] == `JUMP)
        reg_B <= 16'b0;//跳转, 清空寄存器
    else
        reg_B <= gr[id_ir[2:0]];//读入 gr3
    //处理存储指令
    if (id_ir[15:11] == `STORE)
        begin
            if(id_ir[10:8] == ex_ir[10:8] && (ex_ir[15:11] == `ADD || ex_ir[15:11] == `LDIH ||
ex_ir[15:11] == `ADDI
                || ex_ir[15:11] == `SUB || ex_ir[15:11] == `SUBI || ex_ir[15:11] == `ADDC ||
ex_ir[15:11] == `SUBC
                || ex_ir[15:11] == `SLL || ex_ir[15:11] == `SRL || ex_ir[15:11] == `SLA || ex_ir[15:11]
== `SRA ))
                smdr <= ALUo;
            else if(id_ir[10:8] == mem_ir[10:8] && (mem_ir[15:11] == `ADD || mem_ir[15:11] ==
`LDIH
                || mem_ir[15:11] == `ADDI || mem_ir[15:11] == `SUB || mem_ir[15:11] == `SUBI ||
mem_ir[15:11] == `ADDC
                || mem_ir[15:11] == `SUBC|| mem_ir[15:11] == `SLL || mem_ir[15:11] == `SRL ||
mem_ir[15:11] == `SLA
                || mem_ir[15:11] == `SRA || mem_ir[15:11] == `LOAD))
                begin
                    if(mem_ir[15:11]==`LOAD)

```



```

        smdr <= d_datain;
    else
        smdr <= reg_C;
    end
    else if(id_ir[10:8] == wb_ir[10:8] && (wb_ir[15:11] == `ADD || wb_ir[15:11] == `LDIH ||
wb_ir[15:11] == `LOAD
        || wb_ir[15:11] == `ADDI || wb_ir[15:11] == `SUB || wb_ir[15:11] == `SUBI || wb_ir[15:11]
== `ADDC ||
        wb_ir[15:11] == `SUBC || wb_ir[15:11] == `SLL || wb_ir[15:11] == `SRL || wb_ir[15:11]
== `SLA
        || wb_ir[15:11] == `SRA ))
        smdr <= reg_C1;
    else
        smdr <= gr[id_ir[10:8]]; //读入 gr[r1]
    end
    else
        smdr <= gr[id_ir[10:8]]; //读入 gr[r1]
    end
end
end

/***** ALU *****/
//ALU 模块, 进行逻辑与数值运算
reg signed [15:0] reg_A1; //用于 SRA 算数右移指令
always @(*)
begin
    reg_A1 <= reg_A;
end

always @(*)
begin
    if((ex_ir[15:11] == `LOAD) || (ex_ir[15:11] == `STORE) || (ex_ir[15:11] == `BN) || (ex_ir[15:11] == `BNN)
    || (ex_ir[15:11] == `BZ) || (ex_ir[15:11] == `BNZ) || (ex_ir[15:11] == `BC) || (ex_ir[15:11] == `BNC)
    || (ex_ir[15:11] == `JMPR) || (ex_ir[15:11] == `ADD) || (ex_ir[15:11] == `ADDI) || (ex_ir[15:11] == `LDIH))
        //指令为 LOAD、STORE、BN、BNN、BZ、BNZ、BC、BNC 或 JMPR 时, 将 reg_A 与 reg_B 相加并将结果
送入{cf,ALUo}中
        //指令为 ADD、LDIH 或 ADDI 时, 将 reg_A 与 reg_B 相加并把结果送入{cf,ALUo}中
        {cf,ALUo} <= reg_A + reg_B;
    else if((ex_ir[15:11] == `CMP) || (ex_ir[15:11] == `SUB) || (ex_ir[15:11] == `SUBI))
        //指令为 CMP、SUB 或 SUBI 时, 将 reg_A 减去 reg_B 并把结果送入{cf,ALUo}中; 比较的本质是减法
        {cf,ALUo} <= (reg_A - reg_B);
    else if(ex_ir[15:11] == `ADDC)
        {cf,ALUo} <= reg_A + reg_B + cin; //带进位的加法
    else if(ex_ir[15:11] == `SUBC)
        {cf,ALUo} <= reg_A - reg_B - cin; //带借位的减法

```

```

        else if(ex_ir[15:11]==`OR)
            {cf,ALUo}<=(reg_A|reg_B);
        else if(ex_ir[15:11]==`AND)
            {cf,ALUo}<=(reg_A&reg_B);
        else if(ex_ir[15:11]==`XOR)
            {cf,ALUo}<=(reg_A^reg_B);
        else if(ex_ir[15:11]==`SLL)
            {cf,ALUo}<=(reg_A<<reg_B);//移位
        else if(ex_ir[15:11]==`SLA)
            {cf,ALUo}<=(reg_A<<<reg_B);
        else if(ex_ir[15:11]==`SRL)
            {cf,ALUo}<=(reg_A>>reg_B);
        else if(ex_ir[15:11]==`SRA)
            {cf,ALUo}<=(reg_A>>>reg_B);
        else
            {cf, ALUo} <= 17'b0;
    end

/***** EX *****/
//执行阶段，执行运算，控制信号
always @(posedge clock or negedge reset)
    begin
        if (!reset)//重置清零
            begin
                mem_ir <= 16'b0000_0000_0000_0000;
                reg_C <= 16'b0000_0000_0000_0000;
                smdr1 <= 16'b0000_0000_0000_0000;
                zf <= 1'b0;nf <= 1'b0;cin <= 1'b0;dw <= 1'b0;
            end
        else if (state == `exec)
            begin
                mem_ir <= ex_ir;
                reg_C <= ALUo;
                //处理 zf 和 nf
                if ((ex_ir[15:11] == `ADD) || (ex_ir[15:11] == `CMP) || (ex_ir[15:11] == `LDIH) || (ex_ir[15:11]
== `ADDI)
                    || (ex_ir[15:11] == `ADDC) || (ex_ir[15:11] == `SUB) || (ex_ir[15:11] == `SUBI) || (ex_ir[15:11]
== `SUBC))
                    begin
                        if (ALUo == 16'b0000_0000_0000_0000)
                            zf <= 1'b1;//zf 为是否为 0 的状态标志位，为 0 则表示 zf=1
                        else
                            zf <= 1'b0;
                        if (ALUo[15] == 1'b1)//nf 为是否正负的状态标志位

```

```

        nf <= 1'b1;//为负数则 nf=1
    else
        nf <= 1'b0;
    end
else
    begin
        zf <= zf;
        nf <= nf;
    end
//处理 cf, 进位信号
if ((ex_ir[15:11] == `ADD) || (ex_ir[15:11] == `LDIH) || (ex_ir[15:11] == `ADDI) || (ex_ir[15:11]
== `ADDC)
    || (ex_ir[15:11] == `SUB) || (ex_ir[15:11] == `SUBI) || (ex_ir[15:11] == `SUBC))
    cin <= cf;
else
    cin <= cin;
//STORE
if (ex_ir[15:11] == `STORE)
    begin
        dw <= 1'b1;//使能信号
        smdr1 <= smdr;
    end
else
    begin
        dw <= 1'b0;
        smdr1 <= 16'b0;
    end

end

end

end

/***** MEM *****/
//存储阶段, 处理器访问内存进行读取或写入操作
always @(posedge clock or negedge reset)
begin
    if (!reset)//重置
    begin
        wb_ir <= 16'b0000_0000_0000_0000;
        reg_C1 <= 16'b0000_0000_0000_0000;
    end
    else if (state == `exec)
    begin
        wb_ir <= mem_ir;
        if (mem_ir[15:11] == `LOAD)//LOAD 指令则读取内存输入

```

```

        reg_C1 <= d_datain;

    else

        reg_C1 <= reg_C;//否则从寄存器 C 读取数据

    end

end

end

/***** WB *****/
//写回阶段，将结果写回寄存器堆
always @(posedge clock or negedge reset)//写回
begin
    if (!reset)//重置
    begin
        gr[7] <= 16'b0000_0000_0000_0000;
        gr[6] <= 16'b0000_0000_0000_0000;
        gr[5] <= 16'b0000_0000_0000_0000;
        gr[4] <= 16'b0000_0000_0000_0000;
        gr[3] <= 16'b0000_0000_0000_0000;
        gr[2] <= 16'b0000_0000_0000_0000;
        gr[1] <= 16'b0000_0000_0000_0000;
        gr[0] <= 16'b0000_0000_0000_0000;
    end
    else if (state == `exec)
    begin
        if ((wb_ir[15:11] == `LOAD) || (wb_ir[15:11] == `ADD) || (wb_ir[15:11] == `LDIH) ||
            (wb_ir[15:11] == `ADDI)
            || (wb_ir[15:11] == `ADDC) || (wb_ir[15:11] == `SUB) || (wb_ir[15:11] == `SUBI) ||
            (wb_ir[15:11] == `SUBC)
            || (wb_ir[15:11] == `SLL) || (wb_ir[15:11] == `SRL) || (wb_ir[15:11] == `SLA) || (wb_ir[15:11]
                == `SRA))
            gr[wb_ir[10:8]] <= reg_C1;//将处理完的数据写回 gr[r1]
        else
            gr[wb_ir[10:8]] <= gr[wb_ir[10:8]];//否则不进行修改
        end
    end
end
assign y = {8'B0,pc}; //
endmodule

```

8. test.v

```

`timescale 1ps / 1ps
module test;
    //输入
    reg clk;
    reg enable;
    reg reset;
    reg start;
    reg read;
    reg[7:0]address ;
    wire[6:0]light;
    wire [3:0]an;
    reg clkreset;
    integer x;

    //实例化测试对象 the Unit Under Test (UUT)
    cpu uut (
        .clk(clk),
        .enable(enable),
        .reset(reset),
        .clkreset(clkreset),
        .start(start),
        .an(an),
        .read(read),
        .light(light),
        .adress(address)
    );
    always #1 clk = ~clk;

    initial begin
        //以下设置是为了使得输出时序正常
        x=0;clk<=0;enable<=0;start<=0;clkreset<=0;read<=0;
        adress<=8'b00000000;
        #100 reset<=1;read<=1;
        #100 read<=0;
        #100 clkreset<=1;reset<=0;
        #100 reset<=1;clkreset<=0;
        #100 adress<=8'b11111111;
        #100 reset<=1;
        #100 enable<=1;
        #100 start<=1;
        #100 adress<=8'b00000000;
        #10000 start<=0;
    end

```

```
#10000 start<=1;
#100 start<=0;

repeat (65)
begin
    $display("ADD:%h,CELL:%h",x,uut.d_mem.d_mem[x]);
    //打印各个地址储存的数据
    x=x+1;
end
end
endmodule
```