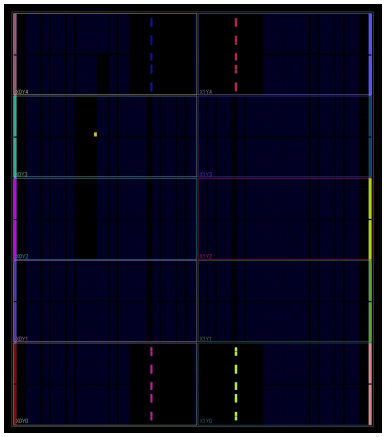
“龙芯杯”第六届全国大学生计算机系统能力培养大赛

北方工业大学“不忘初芯”队

hhhhMIPS项目

初赛设计报告



葛钰峣 姚佳丽

陈信达 牛吉祥

2022 年 8 月

封面图为 CPU 综合后的资源占用情况，不同颜色代表不同组件

目录

第一部分　概述

1　项目背景

本项目的成果是在龙芯提供的 FPGA 实验平台上设计并实现基于 MIPS 32 的 CPU，并使用实验板上的周边硬件，成为一个片上系统。不仅支持大赛要求的57条指令以及其他复杂运算等32条指令，还完成了CP0和TLB的设计以及实现了对异常和冲突的处理。能够运行功能、性能测试。

2　项目概览

本项目计划设计和实现的部分主要包括：CPU、SoC 设计。项目使用的硬件语言为 Verilog，开发平台为 Xilinx Vivado 2019.2。下面为各个部分的概览。

2.1 CPU

CPU 的设计包含指令集、流水线结构。

指令集 本项目的 CPU 实现的是 MIPS 32标准指令集的子集，包括所有的逻辑运算指令、控制流指令以及大部分特权指令，涵盖了大赛所有所要求的全部57条指令以及其他复杂运算等32条指令。

流水线结构 本项目采取经典的5级流水架构。5级分别包括IF(取指)，ID(译码)，EX(执行)，MEM(访存)和WB(写回)。这个5级流水架构使得CPU在运行时不会出现太大的性能浪费，使得每一阶段的电路在运行多指令时可以同时都在执行，这相较于单周期的CPU来说，可以大幅度提高指令的执行周期。CPU采取的时**哈佛结构**存储指令与数据，不是像传统的冯诺依曼架构一样将指令与数据都存在一个存储器里，而是将指令和数据分别单独的存放在不同的两个存储器当中。

2.2Soc 设计

为了验证 CPU 的设计，并且使用实验箱上提供的硬件进行更多的功能演示，我们以 hhhMIPS CPU 为核心搭建了一个 SoC (System-on-Chip)，具有以下模块：

CPU hhhhMIPS CPU

DRAM 使用板载 DDR3 SDRAM 作为主存

GPIO 使用龙芯 confreg 组件，软件控制 LED、数码管，读取开关、按键等状态

2.3开发平台

2.3.1 硬件平台

本项目使用的硬件平台为龙芯 FPGA 实验箱，其主要部件为 Xilinx 的 Artix 7 系列FPGA，型号为 xc7a200t，包含外部器件：

DRAM 128MB DDR3 SDRAM

NOR Flash (CFG) 16MB

NOR Flash (SPI) 32MB

VGA 接口 RGB 输出，各 4 bit

以太网接口 DM9161 100Mbps Ethernet PHY

其他接口 标准 RS232 串口、PS/2 接口、USB PHY

显示屏 NT35510 LCD，分辨率 800 × 480

GPIO 数码管 × 8，LED × 26，拨码开关 × 8，按键 × 19

2.3.2 软件平台

开发 IDE Xilinx Vivado 2019.2 Web HL Edition

CI 系统 Ubuntu 20.04.3 LTS

编译器套件 cross-mipsel-linux-gnu-binutils 2.32-1, cross-mipsel-linux-gnu-gcc 9.1.0-(AUR)

2.4 参考资料

本项目的设计、开发过程需要参考包括且不限于下面列出的书籍、文献和资料：

[1]雷思磊.《自己动手写 CPU》[M].北京:电子工业出版社,2014.9.
[2]李亚民.《计算机原理与设计：Verilog HDL 版》[M].清华大学出版社
[3]汪文祥.《CPU 设计实战》[M].北京:机械工业出版社,2021.1
[4] David A. Patterson《计算机组成与设计 硬件/软件接口》[m].北京:机械工业出版社,2020.4
[5]姚勇斌《超标量处理器设计》[m].北京:清华大学出版社,2011

第二部分　CPU

1　总体结构

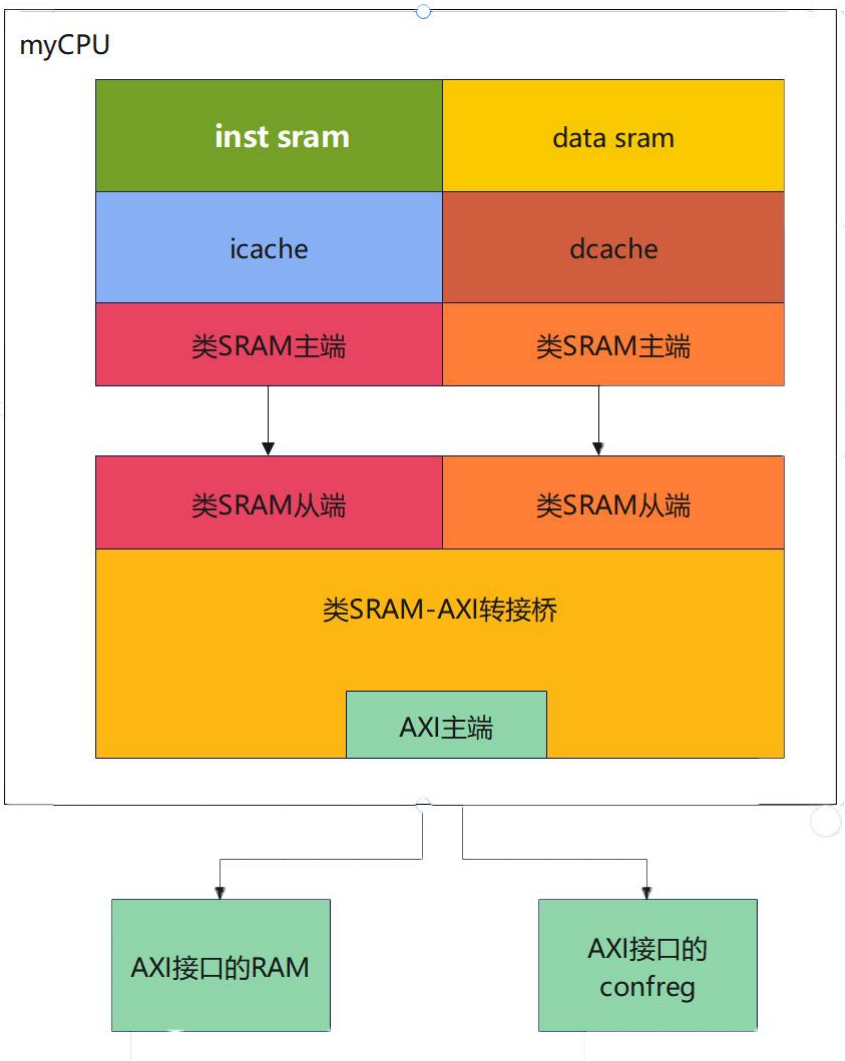


图 1 总体结构图

2 指令集结构

逻辑运算指令 OR, AND, XOR, NOR, ORI, ANDI, XORI, LUI
算术运算指令 ADD, ADDU, SUB, SUBU, SLT, SLTU, ADDI, ADDIU, SLTI, SLTIU, MULT, MULTU,MUL,DIV, DIVU
移位指令 SLL, SRL, SRA, SLLV, SRLV, SRAV
数据移动指令 MFHI, MFLO, MTHI, MTLO, MOVN, MOVZ
跳转/分支指令 J, JAL, JR, JALR, BEQ, BNE, BGTZ, BLEZ, BGEZ, BGEZAL, BLTZ, BLTZAL
加载/存储指令 LB, LBU, LH, LHU, LW, SB, SH, SW, LWL, LWR, SWL, SWR, LL, SC
特权指令 MFC0, MTC0, ERET
自陷指令 BREAK, SYSCALL, TEQ, TNE, TGE, TGEU, TLT, TLTU, TEQI, TNEI, TGEI,TGEIU,TLTI, TLTIU

3 流水线结构

我们初期为了简便快捷正确的完成CPU的大体架构，所以采用了较为简单的5级流水架构。5级分别包括IF(取指)，ID(译码)，EX(执行)，MEM(访存)和WB(写回)。这个5级流水架构使得CPU在运行时不会出现太大的性能浪费，使得每一阶段的电路在运行多指令时可以同时都在执行，这相较于单周期的CPU来说，可以大幅度提高指令的执行周期。CPU采取的时哈佛结构存储指令与数据，不是像传统的冯诺依曼架构一样将指令与数据都存在一个存储器里，而是将指令和数据分别单独的存放在不同的两个存储器当中。

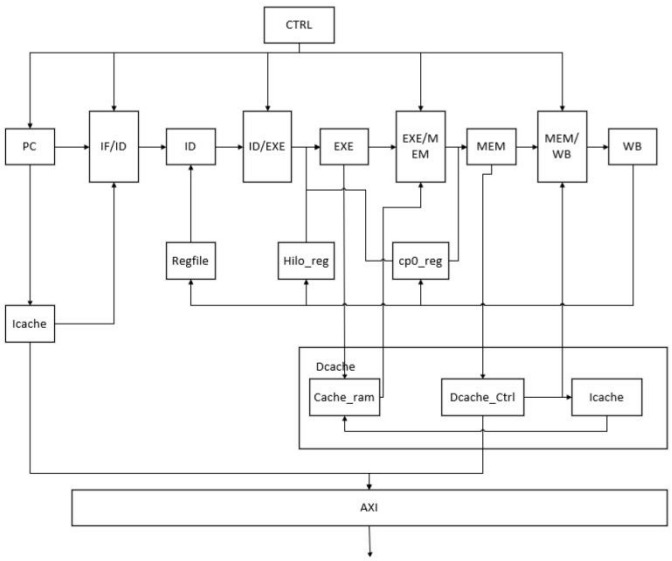


图2 五级流水线示意图

3.1 取指阶段

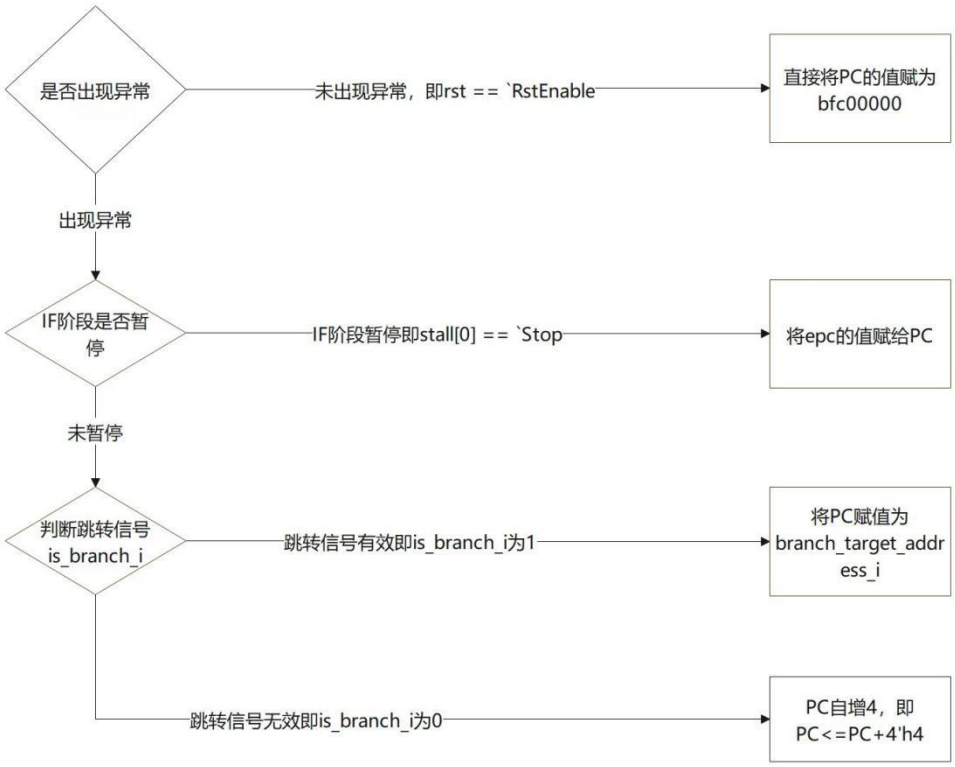


图3 取指阶段流程图

图4 取指阶段模块封装示意图

本阶段从外部只读存储器中读取一条指令，并更新程序计数器即PC使其指向下一条指令。本阶段CPU会先将PC的值传入TLB来得到物理地址，然后将物理地址传入icache中进行比对并最终将指令取得传入ID阶段。另外，此阶段还将处理译码阶段传来的跳转指令，得到跳转信号与新的程序地址后将新的地址写入PC，实现程序跳转的功能。而当CPU的运行出现异常和例外等问题时，PC将可能在解决例外和异常的过程中造成值的丢失与错乱，所以此时将会用CP0中的epc寄存器来存储这些问题发生前PC的值，等到CPU重新正常运转时，再将epc的值赋回给PC当中。

3.2 译码阶段

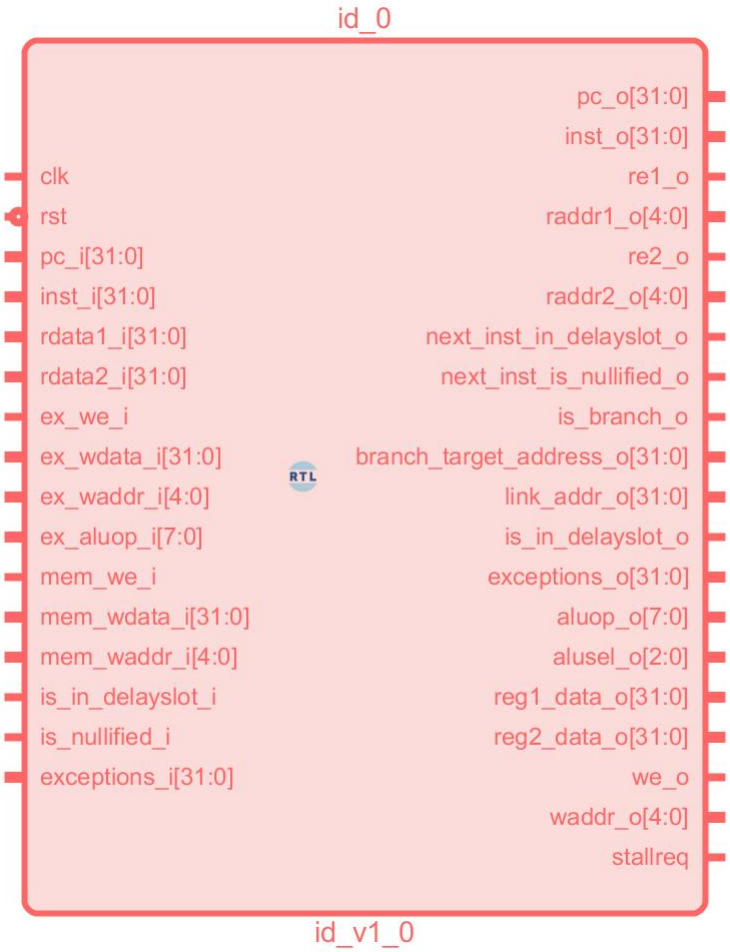


图5 译码阶段模块封装示意图

本阶段主要负责将取指阶段取得的32位指令进行译码，并将译码后得到的信号传递给EX阶段来进行指令的进一步执行。而在译码中，不仅是将指令的请求（对数据进行读写的请求）翻译出来并传递给EX阶段，还要将一些指令所需要的立即数和地址翻译出来。而当处理的指令是条件跳转指令时，需要在本阶段判断是否进行跳转，并对跳转后的地址进行赋值。而当处理的指令是加载指令时，需要解决该指令因在流水线中执行所带来的相关问题。本阶段还将接收来自执行阶段和访存阶段的数据前推。

3.3 执行阶段

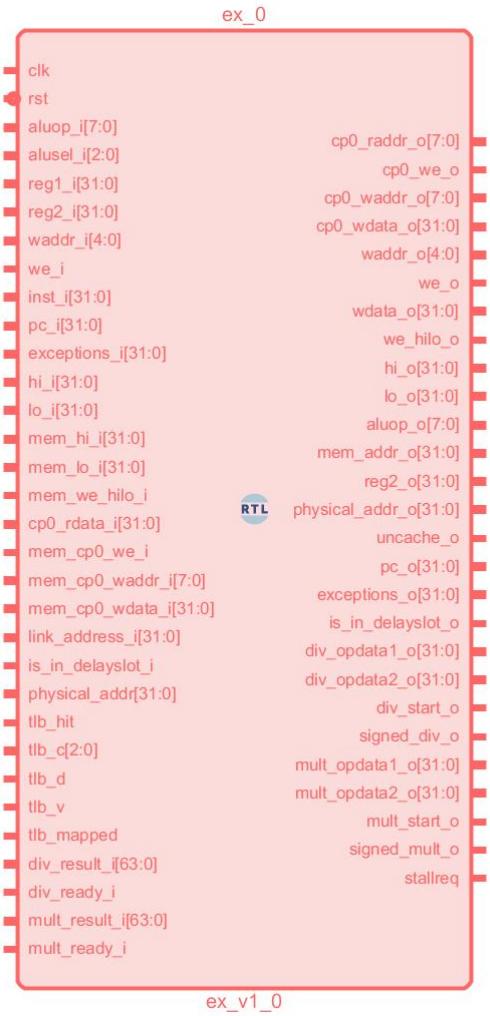


图6 执行阶段模块封装示意图

执行阶段是流水线的第 3 个阶段。此阶段会生成指令的结果，同时计算异常请求，计算访存地址。对于内存写指令（如 SW、SC 等）会在此阶段读取需要写的数据。对于多周期指令，在计算完前会生成一个暂停信号。流水线仅有一个乘法和除法单元。

3.4 访存阶段

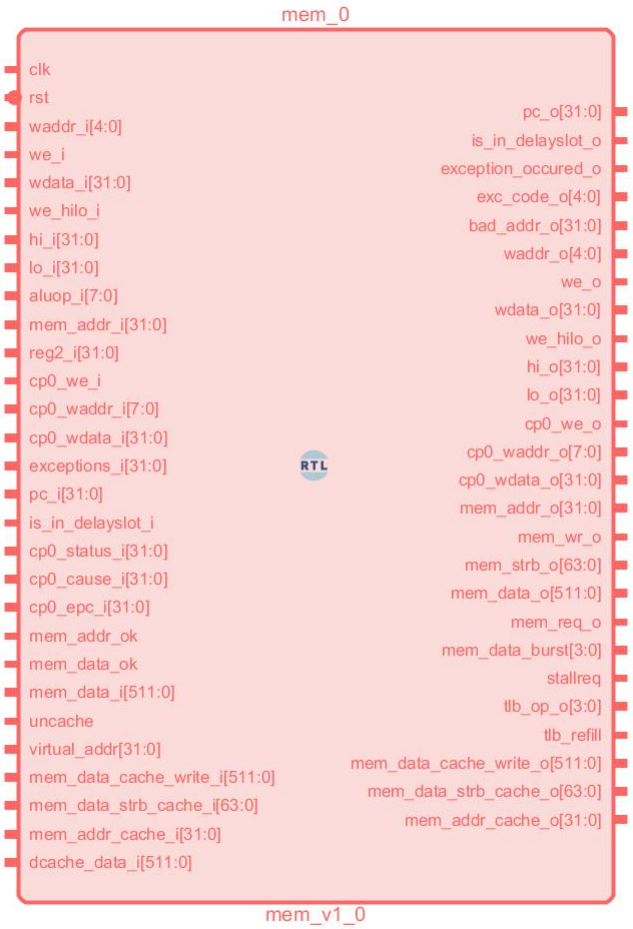


图7 访存阶段模块封装示意图

本阶段把从EX阶段传递来的地址传入TLB中进行转换，然后把从TLB当中传递出来的物理地址传递给dcache中进行对比最终取出所需要的数据。当EX阶段传递过来的信息指示要将数据写到内存当中时，会同样将要写回的地址传入dcache中进行比较，把从dcache传出的数据根据从EX阶段传递过来的信息运行（运行对应的加载指令所对应的操作）并得出要传入回写阶段的信息。

另外，整个流水线的异常处理都集中在本阶段进行，在之前阶段的异常问题只是进行记录标记，一直到本阶段才进行详细的翻译分类，并最终传递给ctrl模块来处理。

3.5 写回阶段

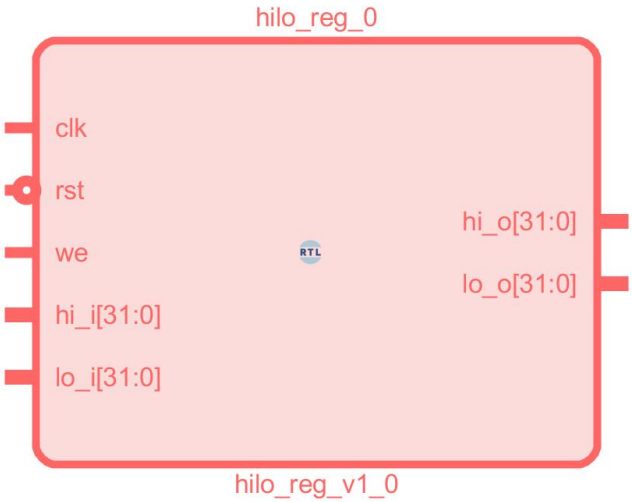


图8 HILO寄存器模块封装示意图

本阶段只是概念上的阶段，因为写回阶段的电路时根据写回目标的不同而不同的，例如寄存器堆的写回阶段在寄存器堆模块内或HILO寄存器，实际并不存在本阶段的实现电路，只是留出一拍时间使寄存器进行数据的写入操作。

4 暂定与冲突

4.1 暂停处理

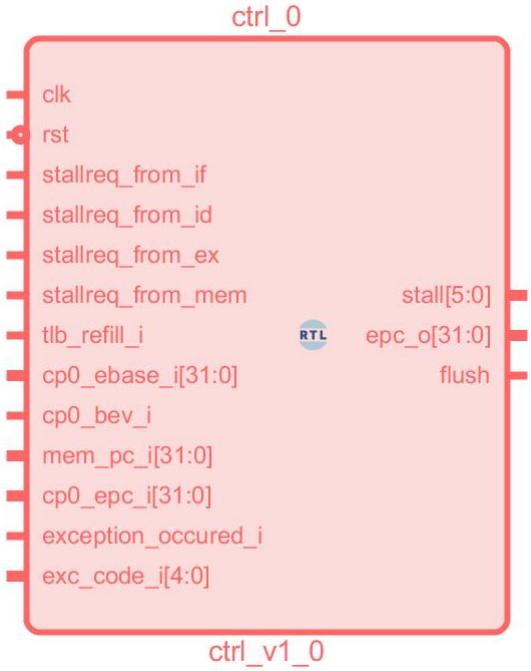


图9 暂停控制模块封装示意图

为了处理暂停和异常，由此设计实现了本模块。该模块和所有阶段的暂停信号相连，如果某个阶段想要暂停，则需要先经过本模块，然后本模块由的信息传来的阶段来判断该在流水线中进行怎样规模的暂停。当在MEM阶段中总结翻译后的异常信息传入本模块时，本模块会根据传入的信息来决定如何解决处理当次异常。

4.2 冲突处理

4.2.1 数据相关

本报告中所设计的 CPU，只需考虑并处理写后读造成的数据相关。第一种方法是暂停流水线，等待上一条指令完成写回后再将写回的内容给到访存阶段的指令；第二种方法是采用旁路技术，即数据前推。在译码阶段判断从执行阶段传递来的待写入数据寄存器编号与将要访问的寄存器是否相同，若是，则直接使用从执行阶段传递来的待写入数据。数据前推可以节约时间、提高效率，但同时也增加了电路的复杂度。

若是相隔一条指令的两条指令产生数据相关，解决方法同上。但有另外特殊情况，若是相隔两条指令的两条指令产生数据相关，那么则不使用上述两种方法，只需将寄存器设计为写优先，读请求读到的数据一定是最新写入的数据，就可以解决此问题，保证相隔两条指令后的那一条指令在译码阶段可以读取到正确的数据进行运算。

另外，还有一种由访存引起的数据相关。当第一条指令是访存指令且处在执行阶段，如果第二条指令恰好需要在译码阶段使用访存指令读内存到寄存器中的数据进行运算时，因为访存指令只有在 MEM 阶段才会读内存到寄存器中，那么此时只能暂停流水线，等待第一条访存指令执行到 MEM 阶段后，才能通过数据前推来解决这种数据相关。

4.2.2 控制相关

分支跳转指令破坏了流水线的顺序执行，本报告中设计的 CPU 在译码阶段判断是否为跳转指令，更新PC 值。但此时已有后一条指令进入取指阶段，为避免浪费掉这条指令，hhhhMIPS 将跳转指令的后一条指令位置设置为延迟槽，无论判断结果是否为跳转指令，都将执行延迟槽中的指令，以此解决控制相关。

5 Cache

5.1 挂载指令Cache

在取指阶段将实际地址传入icache中，然后将地址中的tag值与当前icache中的tag值进行比较，如果相同则直接从icache中读取实际地址中的index和offset值所对应的指令，将指令传递到IF_ID锁存器。如果tag对比时不相等，则需要暂停流水线，然后icache经过总线桥向存储器发送请求，并将要搜索的tag值一并发送，并最后接收到与实际地址中的tag匹配的数据块。然后再在该数据块内进行进一步的搜索。

另外，icache是在if阶段内实现的。

5.2 挂载数据Cache

在执行阶段 EX 计算得出地址 addr 后直接送入 dcache_ram 中查找得到相应的 16 字数据送入 mem阶段。mem 阶段通过比较 tag 得出 dcache 是否命中，如果命中则通过 index 选择从 EX 阶段获得的ram 数据，若是没有命中，则需要暂停流水线提出请求以获得数据。写操作则会将相应的数据与写使能信号传入 wb 阶段，在 wb 阶段将数据写入 dcache ram 中。在 wb 阶段设置了一个数据旁路将数据传回 mem 阶段以避免可能的数据相关。

6 接口与总线设计

6.1 AXI总线（转接桥）

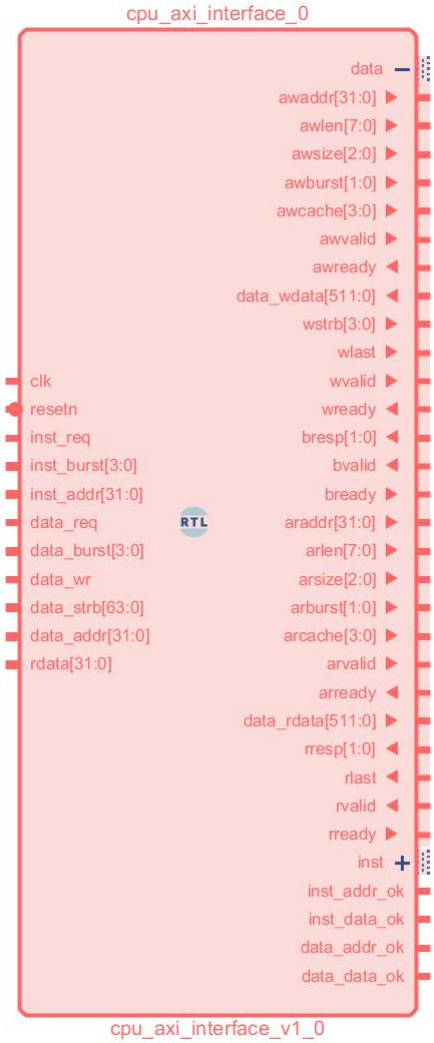


图 10 AXI 模块封装示意图

AXI总线桥模块是为了使CPU可以正常的和其他应用设备进行连接和数据交互。因为类SRAM接口的局限性太大，不管是从比赛测试方面还是从应用设备拓展方面都无法和AXI接口相提并论，所以为了方便地进行比赛的测试，也为了之后对CPU进行拓展，所以设计实现了类SRAM转AXI总线桥（简称AXI总线桥）。类SRAM接口在同一时间内只能同时进行写和读这两种操作中的一个操作。而AXI接口可以同一时间进行读写两种操作，所以转换桥不能只是简单的数据传输，还需要“握手机制”来实现两种接口间的转换。当类SRAM为写操作时，类SRAM接口方持续不断的向AXI接口方发送要写的数据，而AXI方面则是直接接收传来的数据并在地址握手后开始突发传输（进行连续传输），当AXI方面接收到内存发送过来的写回完成信号时，AXI在将信号发送回类SRAM方。类SRAM方在接收到AXI方发送的信号后才结束当次写操作。而如果类SRAM为读操作时，则向AXI方发送所读地址，AXI方接收到地址后将地

址发送给主存，然后开始接收主存发送过来的数据。当AXI数据接收完毕时，则将数据再发送回类SRAM方，最后类SRAM方接收成功后结束当次读操作。

6.2 类SRAM接口

为了配合 AXI 总线转接桥，我们在 IF 阶段增加了一个寄存器 req_en 用以表示允许向外发出请求。

7 其他优化

7.1 除法器

DIV 模块设计采用试商法实现除法运算，对于 32 位的除法，至少需要 32 个时钟周期才能得到除法结果。当流水线执行阶段的 EX 模块发现当前指令是除法指令时，首先暂停流水线，然后将被除数、除数等信息送到 DIV 模块，开始除法运算。DIV 模块在除法运算结束后，通知 EX 模块，并将除法结果送到 EX 模块，后者依据除法结果设置 HI、LO 寄存器的写信息，同时取消暂停流水线。

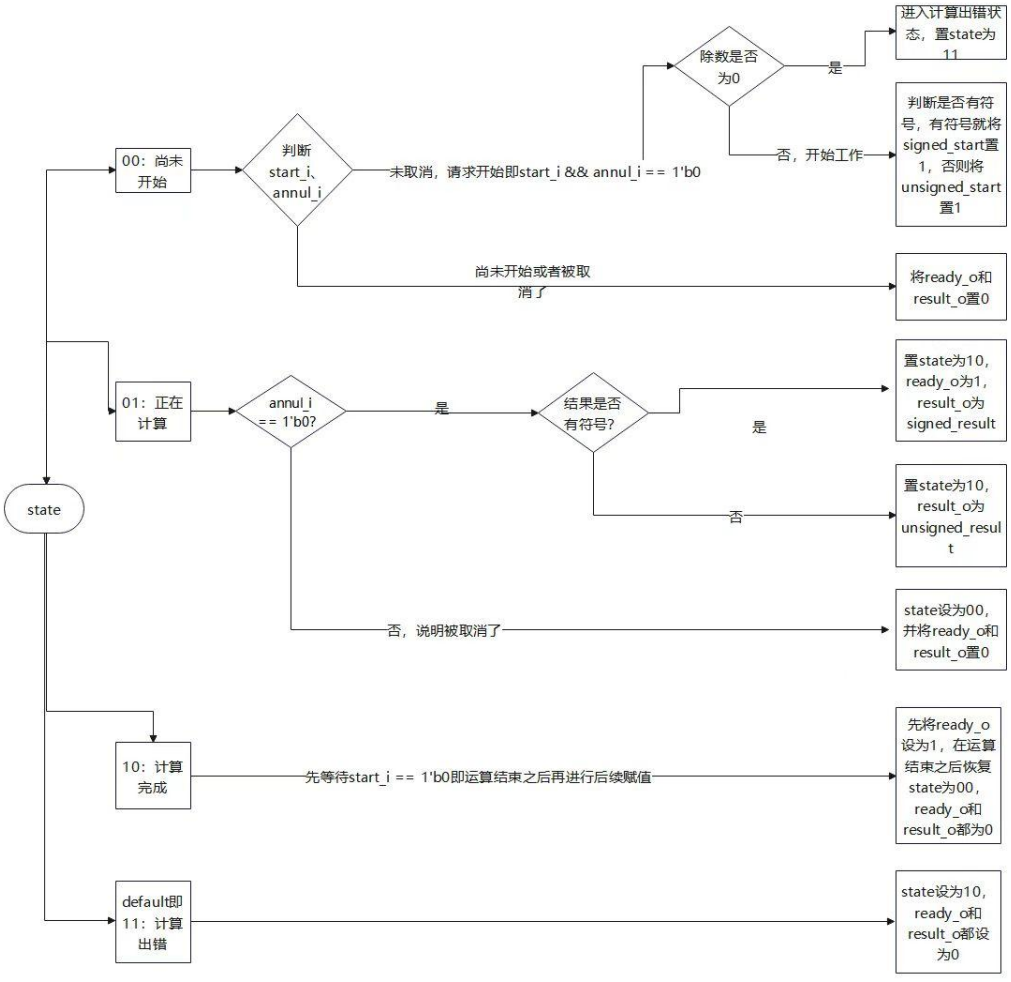


图 11 除法器运行流程图

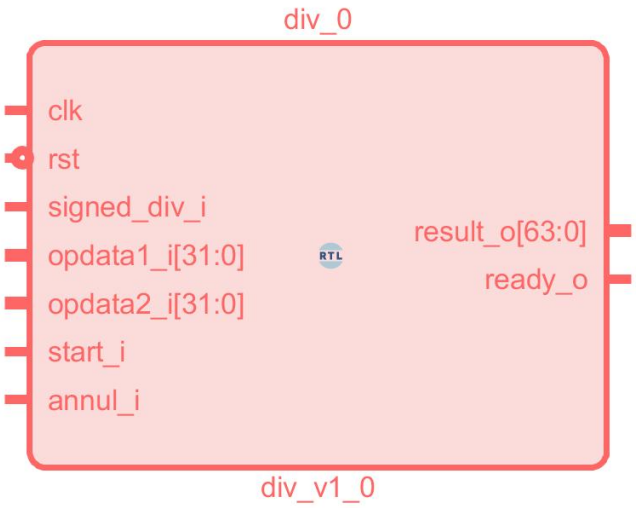


图 12 除法器模块封装示意图

7.2 CP0模块

CP0 是 MIPS 规范中必要的一个协处理器，它提供了操作系统所必须的功能抽象，例如异常处理、内存管理和资源访问控制等。CP0 模块有读、写端口以及处理异常相关的端口，读、写端口由使能、地址和数据三 种信号控制。CP0 通过读/写一个或一些内部寄存器改变一些 CPU 特性，实现配置 CPU 工作状态，控制读写缓存，高速缓存控制，异常检测与处理，存储管理单元控制：对系统的存储区域进行合理的管理、控制、分配。当把额外功能集成到 CPU 中，但又不方便当作外设访问时，常常在 CP0 中增加一些模块以实现这些功能。

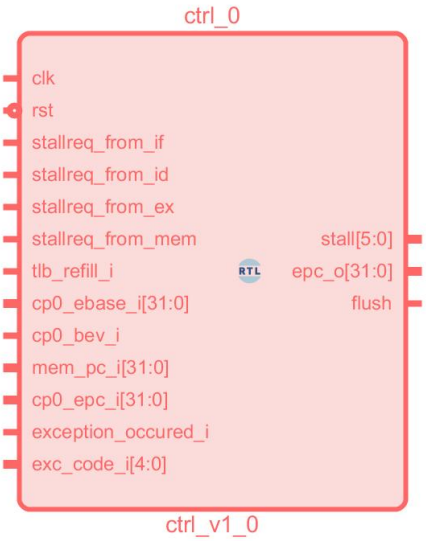


图 13 CP0 模块封装示意图

第三部分　测试结果

1　功能测试

1.1 仿真运行

```
        [6222000 ns] Test is running, debug_wb_pc = 0x00000000
----[6226735 ns] Number 8'd82 Functional Test Point PASS!!!
        [6232000 ns] Test is running, debug_wb_pc = 0x00000000
        [6242000 ns] Test is running, debug_wb_pc = 0xbfc00390
----[6248155 ns] Number 8'd83 Functional Test Point PASS!!!
        [6252000 ns] Test is running, debug_wb_pc = 0x00000000
        [6262000 ns] Test is running, debug_wb_pc = 0x00000000
        [6272000 ns] Test is running, debug_wb_pc = 0x00000000
----[6281625 ns] Number 8'd84 Functional Test Point PASS!!!
        [6282000 ns] Test is running, debug_wb_pc = 0xbfc00d50
        [6292000 ns] Test is running, debug_wb_pc = 0x00000000
        [6302000 ns] Test is running, debug_wb_pc = 0xbfc00390
----[6304555 ns] Number 8'd85 Functional Test Point PASS!!!
        [6312000 ns] Test is running, debug_wb_pc = 0xbfc00490
        [6322000 ns] Test is running, debug_wb_pc = 0x00000000
----[6326855 ns] Number 8'd86 Functional Test Point PASS!!!
        [6332000 ns] Test is running, debug_wb_pc = 0xbfc46a9c
        [6342000 ns] Test is running, debug_wb_pc = 0xbfc46c14
----[6348265 ns] Number 8'd87 Functional Test Point PASS!!!
        [6352000 ns] Test is running, debug_wb_pc = 0x00000000
        [6362000 ns] Test is running, debug_wb_pc = 0xbfc004f8
        [6372000 ns] Test is running, debug_wb_pc = 0xbfc0039c
----[6375135 ns] Number 8'd88 Functional Test Point PASS!!!
        [6382000 ns] Test is running, debug_wb_pc = 0x00000000
        [6392000 ns] Test is running, debug_wb_pc = 0x00000000
        [6402000 ns] Test is running, debug_wb_pc = 0x00000000
        [6412000 ns] Test is running, debug_wb_pc = 0x00000000
----[6413285 ns] Number 8'd89 Functional Test Point PASS!!!
================================================================
Test end!
----PASS!!!
$finish called at time : 6417357500 ps : File "C:/Users/GeYuYao/func_test_v0.01/soc_axi_func/testbench/mycpu_tb.v" Line 269
run: Time (s): cpu = 00:09:06 ; elapsed = 00:09:33 . Memory (MB): peak = 1515.672 ; gain = 353.043
```

图 14　功能测试仿真运行结果

1.2 上板测试

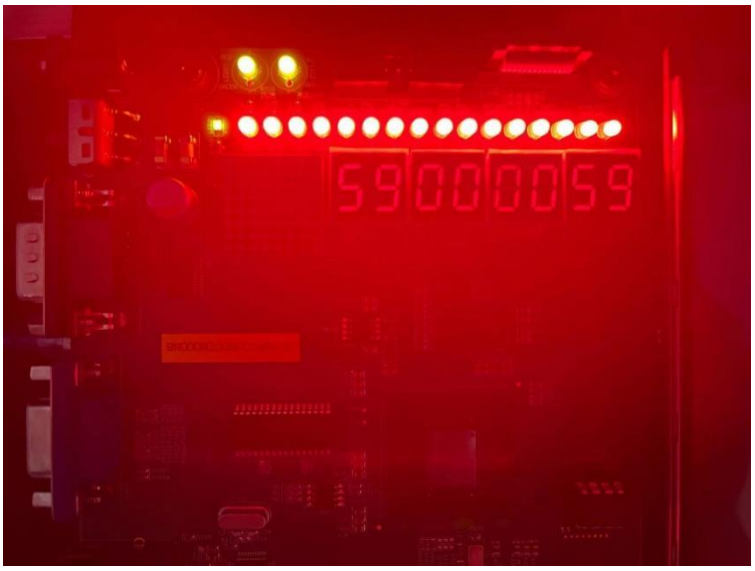


图 15　功能测试上板运行结果（59 为十六进制表示）

2 记忆游戏

2.1 上板测试

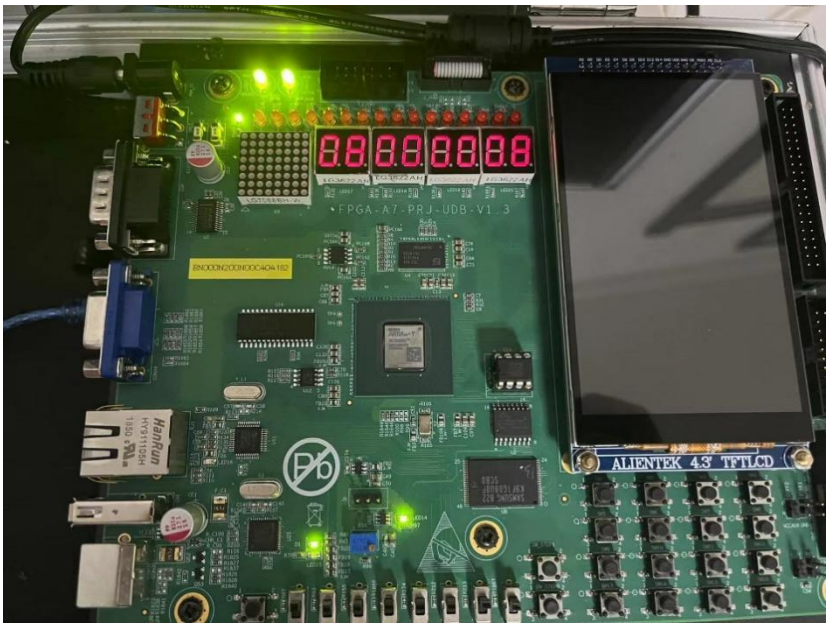


图 16 记忆游戏测试通过实拍

如下表所示，AXI 接口的 CPU 全部通过 89 个功能点测试，上板时任意切换种子均通过。

表3-1 功能测试分数计算

myCPU接口类型	89个功能点通过数	记忆游戏
sram接口	0	-
axi接口	89	1

3 性能测试

如下表所示，在cpu_clk为97MHz,sys_clk为100MHz时，对应测试程序的结果，最终得分为55.849。

表3-2 性能测试分数计算

序号	测试程序	myCPU	gs132	T_{gs132}/T_{mycpu}
		上板计时(16进制)	上板(16进制)	
		数码管显示	数码管显示	
cpu_clk : sys_clk		97MHz : 100MHz	50MHz : 100MHz	-
1	bitcount	525B4	13CF7FA	61.57986198
2	bubble_sort	1A466F	7BDD47E	75.42607611
3	coremark	57AEB1	10CE6772	49.06805308
4	crc32	3292B6	AA1AA5C	53.81647607
5	dhrystone	F38DD	1FC00D8	33.37269859
6	quick_sort	1E9378	719615A	59.43803373
7	select_sort	1B23C5	6E0009A	64.84967579
8	sha	1C8341	74B8B20	65.49862919
9	stream_copy	1BFAE	853B00	76.18616826
10	stringsearch	2267AE	50A1BCC	37.49770752

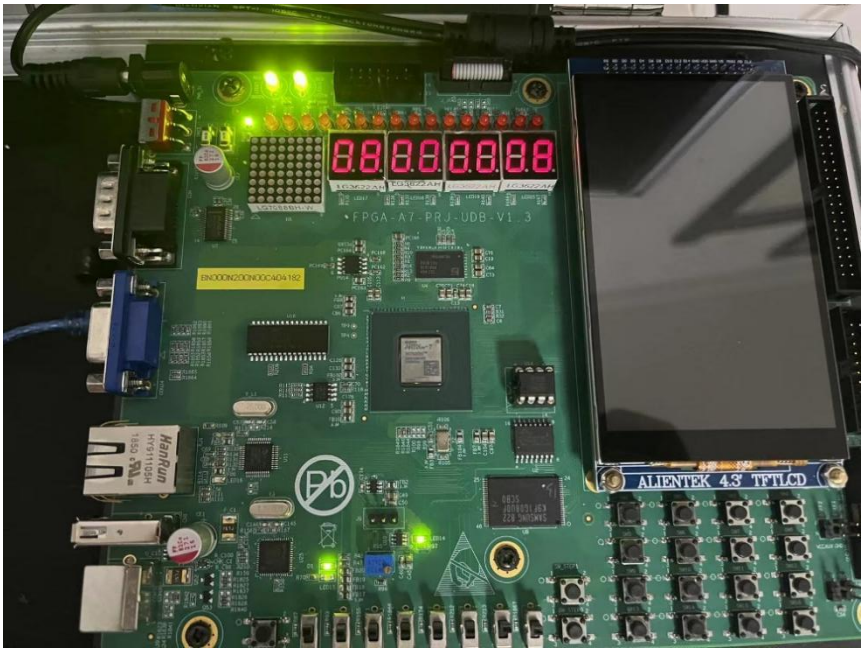


图 17 性能测试上板实拍

4 系统测试

```
C:\WINDOWS\system32\cmd.exe - python  term.py -s com4 -b 57600
F:\HardwareDesign\lx2022_hhhcpu\system_test_v0.01\supervisor-mips32\term>python term.py -s com4 -b 57600
MONITOR for MIPS32 - initialized.
>> g
>>addr: 0x8000300c

elapsed time: 4.424s
>> g
>>addr: 0x8000303c

elapsed time: 0.901s
>> g
>>addr: 0x800030c4

elapsed time: 8.855s
>> g
>>addr: 0x8000315c
OK
elapsed time: 0.000s
>> g
>>addr: 0x80003180

elapsed time: 10.883s
>> g
>>addr: 0x800031b4

elapsed time: 5.442s
>> g
>>addr: 0x800031fc

elapsed time: 6.349s
>> g
>>addr: 0x80003228

elapsed time: 8.162s
>>
```

图 18 系统测试通过截图

附录

A. 声明

本报告中CPU源代码参考《自己动手写CPU》中关于OpenMIPS设计以及本校2021年完成的CPU设计。

B. 致谢

感谢本组四位同学几个月以来的奋斗和支持，也特别感谢吴磊老师、龚怿学长在修改方向上指导与帮助，感谢林达华和郭志坚同学在仿真和上板过程中给予的帮助和支持，也感谢北方工业大学信息学院和龙芯官方对本次比赛的大力支持！