

# 同济大学计算机科学与技术系

## 实验指导



题目：CPU 改造、下板

专    业 计算机科学与技术

课    程 计算机系统试验

授课老师 郭玉臣

时    间 2022 年 2 月

# 目录

1	实验目的.....	3
2	实验内容.....	3
3	实验环境.....	4
3.1	软件环境.....	4
3.2	硬件环境.....	4
4	实验过程与方法 .....	4
4.1	指令设计和数据通路设计 .....	4
4.2	总体结构和数据通路设计 .....	12
4.3	模块设计 .....	13
4.4	测试和调试过程与方法 .....	20
4.5	仿真过程与方法 .....	20
4.6	下板验证过程与方法 .....	23
5	约束文件修改说明 .....	23
6	实验要求.....	24

# 1 实验目的

- 回顾 MIPS 五阶段流水线 CPU 的基本结构
- 学习教学用 CPU 各模块的编写以及模块之间的连接结构
- 加深对 Verilog 语言的理解以及对硬件开发的理解
- 为之后移植操作系统做准备工作

# 2 实验内容

主要的实验内容就是改造 CPU，支持到 89 条指令。参考《自己动手写 CPU》，改造 54 条指令 MIPS 流水线 CPU，支持到 **89 条指令，实现 CP0、异常处理**，需要新增的 35 条指令如下：

- 移动操作指令：

`movn、movz`

- 算术操作指令：

`clo、madd、maddu、msub、msubu`

- 转移指令：

`b、bal、bgezal、bgtz、blez、bltz、bltzal`

- 加载存储指令：

`ll、lwl、lwr、sc、swl、swr`

- 异常相关指令：

`tge、tgeu、tlt、tltu、tne、teqi、tgei、tgeiu、tlti、tltiu、tnei`

- 其他指令：

`nop、ssnop、sync、pref`

此次实验中实现的 MIPS CPU 为五级流水线 CPU，分别有取指、译码、执行、访存、回写五个阶段。采用**大端**模式。采用哈佛结构，有分开的指令存储器和数据存储器。

## 3 实验环境

### 3.1 软件环境

开发环境：建议 Vivado v2016.2(64 bit)、Vivado 2020.2，可自选

仿真环境：ModelSim PE 10.4c

测试环境：MARS 4.5

文档管理：office、notepad++等

### 3.2 硬件环境

计算机：自备

开发板：NEXYS 4 DDR Atrix-7

## 4 实验过程与方法

### 4.1 指令设计和数据通路设计

#### 1.movn

- 格式：movn rd,rs,rt
- 描述：if  $rt \neq 0$  then  $rd \leftarrow rs$ ，判断地址为 rt 的通用寄存器的值，如果不为零，那么将地址为 rs 的通用寄存器的值赋给地址为 rd 的通用寄存器；反之，保持地址为 rd 的通用寄存器不变。movn 即为 Move Conditional on Not Zero 的意思。
- 部件：PC, NPC, IMEM, RegFile

#### 2.movz

- 格式：movz rd,rs,rt
- 描述：if  $rt = 0$  then  $rd \leftarrow rs$ ，与 movn 相反，判断地址为 rt 的通用寄存器的值，如果为零，那么将地址为 rs 的通用寄存器的值赋给地址为 rd 的通用寄存器；反之，保持地址为 rd 的通用寄存器不变。movz 即为 Move Conditional on Zero 的意思。

- 部件：PC, NPC, IMEM, RegFile

### 3.clo

- 格式：clo rd,rs
- 描述： $rd \leftarrow \text{coun\_leading\_ones } rs$ ，对地址为 rs 的通用寄存器的值，从其最高位开始向最低位方向检查，直到遇到值为“0”的位，将该位之前“1”的个数保存到地址为 rd 的通用寄存器中，如果地址为 rs 的通用寄存器的所有位都为1(即 0xFFFFFFFF)，那么将 32 保存到地址为 rd 的通用寄存器中。

- 部件：PC, NPC, IMEM, RegFile

### 4.madd

- 格式：madd rs,rt
- 描述： $\{HI, LO\} \leftarrow \{HI, LO\} + rs \times rt$ ，将地址为 rs 的通用寄存器的值与地址为 rt 的通用寄存器的值作为有符号数进行乘法运算，运算结果与{HI, LO}相加，相加的结果保存到{HI, LO}中。此处{HI, LO}表示 HI、LO 寄存器连接形成的 64 位数，HI 是高 32 位，LO 是低 32 位。

- 部件：PC, NPC, IMEM, RegFile, HI,LO

### 5.maddu

- 格式：maddu rs,rt
- 描述： $\{HI, LO\} \leftarrow \{HI, LO\} + rs \times rt$ ，将地址为 rs 的通用寄存器的值与地址为 rt 的通用寄存器的值作为无符号数进行乘法运算，运算结果与{HI, LO}相加，相加的结果保存到{HI, LO}中。此处{HI, LO}表示 HI、LO 寄存器连接形成的 64 位数，HI 是高 32 位，LO 是低 32 位。

- 部件：PC, NPC, IMEM, RegFile, HI,LO

### 6.msub

- 格式：msub rs,rt
- 描述： $\{HI, LO\} \leftarrow \{HI, LO\} - rs \times rt$ ，将地址为 rs 的通用寄存器的值与地址为 rt 的通用寄存器的值作为有符号数进行乘法运算，运算使用{HI, LO}减去乘法结果，相减的结果保存到{HI, LO}中。

- 部件：PC, NPC, IMEM, RegFile, HI,LO

## 7.msubu

- 格式: msubu rs,rt
- 描述:  $\{HI, LO\} \leftarrow \{HI, LO\} - rs \times rt$ , 将地址为 rs 的通用寄存器的值与地址为 rt 的通用寄存器的值作为无符号数进行乘法运算, 运算使用 {HI, LO} 减去乘法结果, 相减的结果保存到 {HI, LO} 中。
- 部件: PC, NPC, IMEM, RegFile, HI, LO

## 8.b

- 格式: b offset
- 描述: 无条件转移, b 指令可以认为是 beq 指令的特殊情况, 当 beq 指令的 rs、rt 都等于 0 时, 即为 b 指令。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT18, ADD

## 9.bal

- 格式: bal offset
- 描述: 无条件转移, 并且将转移指令后面第 2 条指令的地址作为返回地址, 保存到通用寄存器 \$31。bal 指令是 bgezal 指令的特殊情况, 当 bgezal 指令的 rs 为 0 时, 就是 bal 指令。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT18, ADD

## 10.bgezal

- 格式: bgezal rs, offset
- 描述: if  $rs \geq 0$  then branch, 如果地址为 rs 的通用寄存器的值大于等于 0, 那么发生转移, 并且将转移指令后面第 2 条指令的地址作为返回地址, 保存到通用寄存器 \$31。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT18, ADD

## 11.bgtz

- 格式: bgtz rs, offset

- 描述: if  $rs > 0$  then branch, 如果地址为  $rs$  的通用寄存器的值大于零, 那么发生转移。

- 部件: PC, NPC, IMEM, RegFile, ALU, EXT18, ADD

## 12.blez

- 格式: blez  $rs$ , offset

- 描述: if  $rs \leq 0$  then branch, 如果地址为  $rs$  的通用寄存器的值小于等于零, 那么发生转移。

- 部件: PC, NPC, IMEM, RegFile, ALU, EXT18, ADD

## 13.bltz

- 格式: bltz  $rs$ , offset

- 描述: if  $rs < 0$  then branch, 如果地址为  $rs$  的通用寄存器的值小于 0, 那么发生转移。

- 部件: PC, NPC, IMEM, RegFile, ALU, EXT18, ADD

## 14.bltzal

- 格式: bltzal  $rs$ , offset

- 描述: if  $rs < 0$  then branch, 如果地址为  $rs$  的通用寄存器的值小于 0, 那么发生转移, 并且将转移指令后面第 2 条指令的地址作为返回地址, 保存到通用寄存器 \$31。

- 部件: PC, NPC, IMEM, RegFile, ALU, EXT18, ADD

## 15.ll

- 格式: ll  $rt$ , offset(base)

- 描述: 从内存中指定的加载地址处, 读取一个字节, 然后符号扩展至 32 位, 保存到地址为  $rt$  的通用寄存器中。

- 部件: PC, NPC, IMEM, RegFile, ALU, EXT16, DMEM

## 16.lwl

- 格式: lwl rt, offset(base)
- 描述: 从内存中指定的加载地址处, 加载一个字的最高有效部分。lwl 指令对加载地址没有要求, 从而允许地址非对齐加载, 这是与 lh、lhu、lw 指令的不同之处。在大端模式、小端模式下, lwl 指令的效果不同。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT16, DMEM

## 17.lwr

- 格式: lwr rt, offset(base)
- 描述: 从内存中指定的加载地址处, 加载一个字的最低有效部分。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT16, DMEM

## 18.sc

- 格式: sc rt, offset(base)
- 描述: 如果 RMW 序列没有受到干扰, 也就是 LLbit 为 1, 那么将地址为 rt 的通用寄存器的值保存到内存中指定的存储地址处, 同时设置地址为 rt 的通用寄存器的值为 1, 设置 LLbit 为 0。如果 RMW 序列受到了干扰, 也就是 LLbit 为 0, 那么不修改内存, 同时设置地址为 rt 的通用寄存器的值为 0。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT16, DMEM

## 19.swl

- 格式: swl rt, offset(base)
- 描述: 将地址为 rt 的通用寄存器的高位部分存储到内存中指定的地址处, 存储地址的最后两位确定要存储 rt 通用寄存器的哪几个字节。swl 指令对存储地址没有对齐要求, 这是与 sh、sw 指令的不同之处。在大端模式、小端模式下, swl 指令的效果不同。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT16, DMEM

## 20.swr

- 格式: swr rt, offset(base)
- 描述: 将地址为 rt 的通用寄存器的低位部分存储到内存中指定的地址处。
- 部件: PC, NPC, IMEM, RegFile, ALU, EXT16, DMEM

## 21.tge

- 格式: tge rs, rt
- 描述: if  $GPR[rs] \geq GPR[rt]$  then trap, 将地址为 rs 的通用寄存器的值, 与地址为 rt 的通用寄存器的值作为有符号数进行比较, 如果前者大于等于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 22.tgeu

- 格式: tgeu rs, rt
- 描述: if  $GPR[rs] \geq GPR[rt]$  then trap, 将地址为 rs 的通用寄存器的值, 与地址为 rt 的通用寄存器的值作为无符号数进行比较, 如果前者大于等于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 23.tlt

- 格式: tlt rs, rt
- 描述: if  $GPR[rs] < GPR[rt]$  then trap, 将地址为 rs 的通用寄存器的值, 与地址为 rt 的通用寄存器的值作为有符号数进行比较, 如果前者小于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 24.tltu

- 格式: `tltu rs, rt`
- 描述: if  $GPR[rs] < GPR[rt]$  then trap, 将地址为 `rs` 的通用寄存器的值, 与地址为 `rt` 的通用寄存器的值作为无符号数进行比较, 如果前者小于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 25.tne

- 格式: `tne rs, rt`
- 描述: if  $GPR[rs] \neq GPR[rt]$  then trap, 将地址为 `rs` 的通用寄存器的值, 与地址为 `rt` 的通用寄存器的值进行比较, 如果不相等, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 26.teqi

- 格式: `teqi rs, immediate`
- 描述: if  $GPR[rs] = \text{sign\_extended}(\text{immediate})$  then trap, 将地址为 `rs` 的通用寄存器的值, 与指令中 16 位立即数符号扩展至 32 位后的值进行比较, 如果两者相等, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 27.tgei

- 格式: `tgei rs, immediate`
- 描述: if  $GPR[rs] \geq \text{sign\_extended}(\text{immediate})$  then trap, 将地址为 `rs` 的通用寄存器的值, 与指令中 16 位立即数符号扩展至 32 位后的值作为有符号数进行比较, 如果前者大于等于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU
- 数据通路:

## 28.tgeiu

- 格式: tgeiu rs, immediate
- 描述: if  $GPR[rs] \geq \text{sign\_extended}(\text{immediate})$  then trap, 将地址为 rs 的通用寄存器的值, 与指令中 16 位立即数符号扩展至 32 位后的值作为无符号数进行比较, 如果前者大于等于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 29.tlti

- 格式: tlti rs, immediate
- 描述: if  $GPR[rs] < \text{sign\_extended}(\text{immediate})$  then trap, 将地址为 rs 的通用寄存器的值, 与指令中 16 位立即数符号扩展至 32 位后的值作为有符号数进行比较, 如果前者小于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 30.tltiu

- 格式: tltiu rs, immediate
- 描述: if  $GPR[rs] < \text{sign\_extended}(\text{immediate})$  then trap, 将地址为 rs 的通用寄存器的值, 与指令中 16 位立即数符号扩展至 32 位后的值作为无符号数进行比较, 如果前者小于后者, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

## 31.tnei

- 格式: tnei rs, immediate
- 描述: if  $GPR[rs] \neq \text{sign\_extended}(\text{immediate})$  then trap, 将地址为 rs 的通用寄存器的值, 与指令中 16 位立即数符号扩展至 32 位后的值进行比较, 如果两者不相等, 那么引发自陷异常。
- 部件: PC, NPC, CP0, IMEM, RegFiles, ALU

### 32.nop

- 描述：空指令。

### 33.ssnop

- 描述：一种特殊类型的空指令，此次实验中实现的 MIPS CPU 中与 nop 的作用相同，所以可以按照 nop 指令的处理方式来处理 ssnop 指令。

### 34.sync

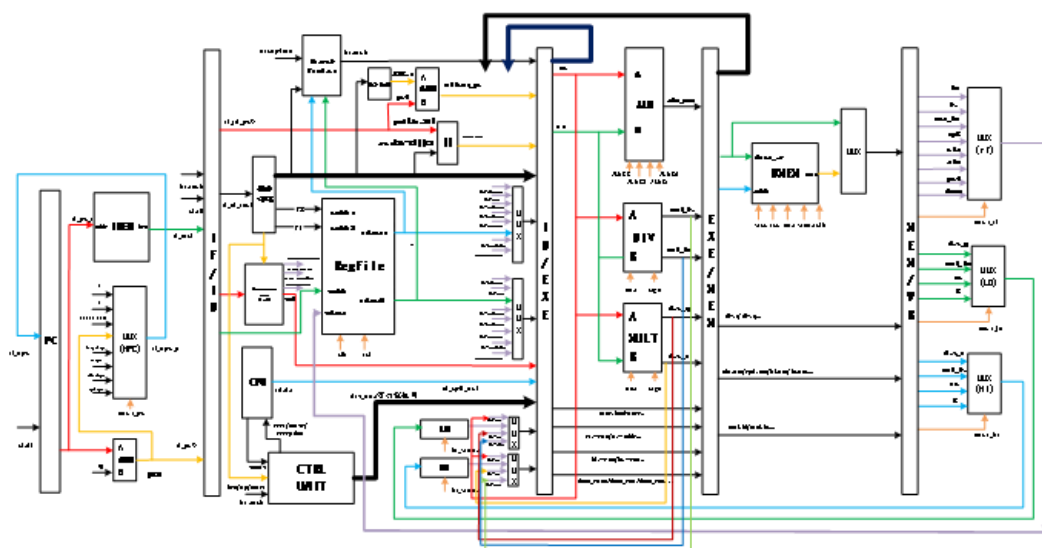
- 描述：用于保证加载、存储操作的顺序。此次实验中实现的 MIPS CPU 中，售加载、存储操作是严格按照指令顺序执行的，所以可将 sync 指令当作 nop 指令处理。

### 35.pref

- 描述：用于缓存预取。此次实验中实现的 MIPS CPU 中没有实现缓存，所以可将 pref 指令当作 nop 指令处理。

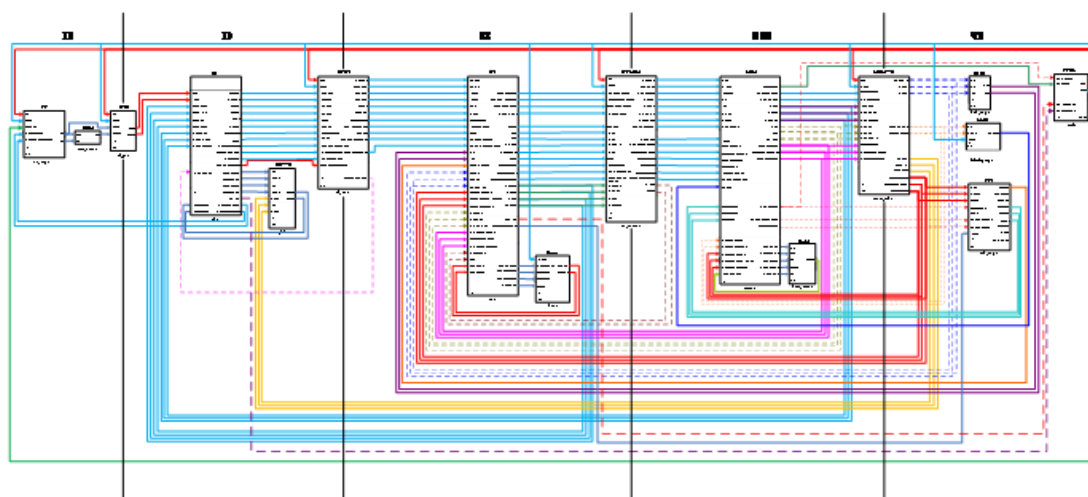
## 4.2 总体结构和数据通路设计

新增指令之前，54 条 MIPS CPU 的总体结构和数据通路设计如下（供参考）：



在设计 89 条 MIPS CPU 之前，在 54 条 MIPS CPU 的基础上修改完成了数据通

路的设计，描述其总体结构。总体结构和数据通路设计如下（供参考）：

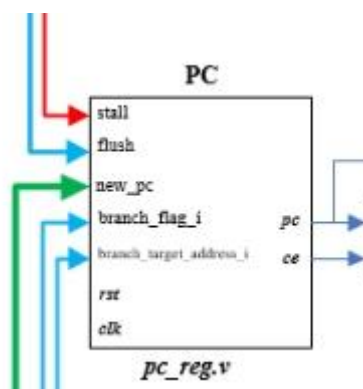


### 4.3 模块设计

逐个模块完成设计和程序编写，最终完成整个系统的设计。

#### 1. PC 寄存器模块

- 解释说明：可以看作 IF 段的流水寄存器。
- 模块设计



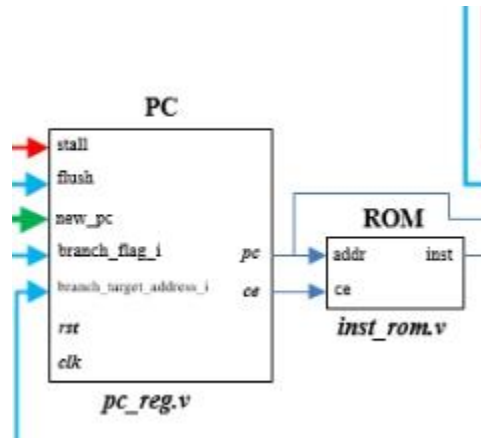
- 接口定义

```

1. module pc_reg(
2.     input wire clk,
3.     input wire rst,
4.     //来自控制模块的信息
5.     input wire[5:0] stall,
6.     input wire flush,
7.     input wire[`RegBus] new_pc,
8.     //来自译码阶段的信息
9.     input wire branch_flag_i,
10.    input wire[`RegBus] branch_target_address_i,
11.    output reg[`InstAddrBus] pc,
12.    output reg ce
13. );
    
```

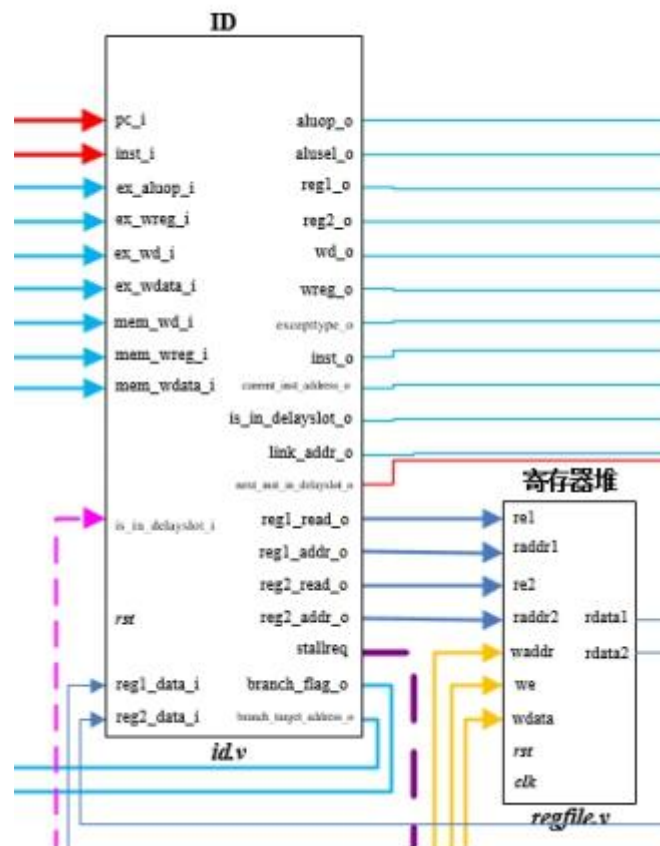
## 2. IF 模块

- 解释说明：IF 级部件，主要包含指令存储器等。
- 模块设计



## 3. ID 模块

- 解释说明：ID 级部件，主要包含控制单元模块、通用寄存器堆等。
- 模块设计



- 接口定义

```
1. module id(
2.     input wire rst,
```

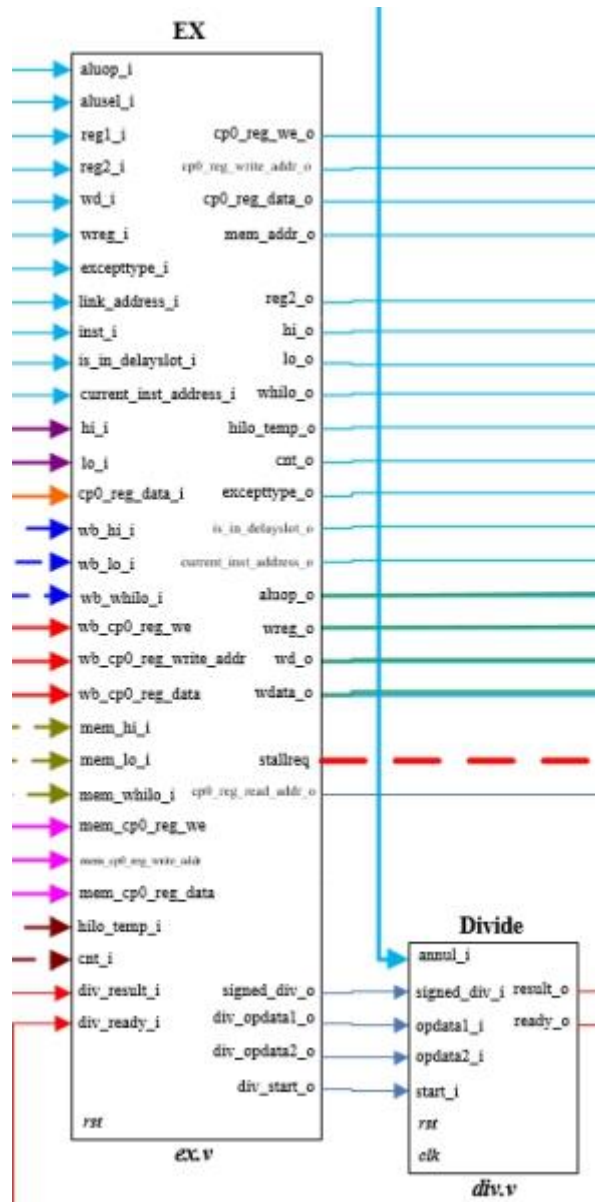
```

3.  input wire[`InstAddrBus] pc_i,
4.  input wire[`InstBus] inst_i,
5.      //处于执行阶段的指令的一些信息，用于解决 load 相关
6.      input wire[`AluOpBus] ex_aluop_i,
7.  //处于执行阶段的指令要写入的目的寄存器信息
8.  input wire ex_wreg_i,
9.  input wire[`RegBus] ex_wdata_i,
10. input wire[`RegAddrBus] ex_wd_i,
11. //处于访存阶段的指令要写入的目的寄存器信息
12. input wire mem_wreg_i,
13. input wire[`RegBus] mem_wdata_i,
14. input wire[`RegAddrBus] mem_wd_i,
15. input wire[`RegBus] reg1_data_i,
16. input wire[`RegBus] reg2_data_i,
17. //如果上一条指令是转移指令，那么下一条指令在译码的时候 is_in_delayslot 为 true
18. input wire is_in_delayslot_i,
19. //送到 regfile 的信息
20. output reg reg1_read_o,
21. output reg reg2_read_o,
22. output reg[`RegAddrBus] reg1_addr_o,
23. output reg[`RegAddrBus] reg2_addr_o,
24. //送到执行阶段的信息
25. output reg[`AluOpBus] aluop_o,
26. output reg[`AluSelBus] alusel_o,
27. output reg[`RegBus] reg1_o,
28. output reg[`RegBus] reg2_o,
29. output reg[`RegAddrBus] wd_o,
30. output reg wreg_o,
31. output wire[`RegBus] inst_o,
32. output reg next_inst_in_delayslot_o,
33. output reg branch_flag_o,
34. output reg[`RegBus] branch_target_address_o,
35. output reg[`RegBus] link_addr_o,
36. output reg is_in_delayslot_o,
37.     output wire[31:0] excepttype_o,
38.     output wire[`RegBus] current_inst_address_o,
39. output wire stallreq
40. );

```

## 4.EXE 模块

- 解释说明：EXE 级部件，除法器模块。
- 模块设计



## • 接口定义

```

1. module ex(
2.   input wire rst,
3.   //送到执行阶段的信息
4.   input wire[`AluOpBus]      aluop_i,
5.   input wire[`AluSelBus]     alusel_i,
6.   input wire[`RegBus]        reg1_i,
7.   input wire[`RegBus]        reg2_i,
8.   input wire[`RegAddrBus]    wd_i,
9.   input wire                  wreg_i,
10.  input wire[`RegBus]         inst_i,
11.  input wire[31:0]            excepttype_i,
12.  input wire[`RegBus]         current_inst_address_i,
13.  //HI、LO寄存器的值
14.  input wire[`RegBus]         hi_i,
15.  input wire[`RegBus]         lo_i,
16.  //回写阶段的指令是否要写 HI、LO，用于检测 HI、LO 的数据相关
17.  input wire[`RegBus]         wb_hi_i,
18.  input wire[`RegBus]         wb_lo_i,
19.  input wire                  wb_whileo_i,
20.  //访存阶段的指令是否要写 HI、LO，用于检测 HI、LO 的数据相关
21.  input wire[`RegBus]         mem_hi_i,
22.  input wire[`RegBus]         mem_lo_i,
23.  input wire                  mem_whileo_i,

```

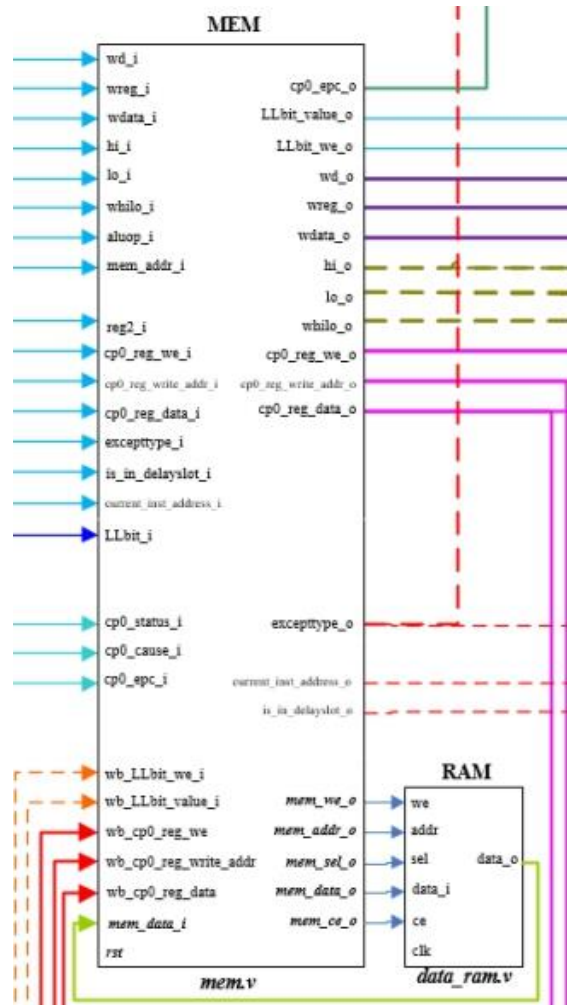
```

24. input wire[`DoubleRegBus]    hilo_temp_i,
25. input wire[1:0]              cnt_i,
26. //与除法模块相连
27. input wire[`DoubleRegBus]    div_result_i,
28. input wire                   div_ready_i,
29. //是否转移、以及 link address
30. input wire[`RegBus]          link_address_i,
31. input wire                   is_in_delayslot_i,
32. //访存阶段的指令是否要写 CP0, 用来检测数据相关
33.     input wire               mem_cp0_reg_we,
34. input wire[4:0]              mem_cp0_reg_write_addr,
35. input wire[`RegBus]          mem_cp0_reg_data,
36. //回写阶段的指令是否要写 CP0, 用来检测数据相关
37.     input wire               wb_cp0_reg_we,
38. input wire[4:0]              wb_cp0_reg_write_addr,
39. input wire[`RegBus]          wb_cp0_reg_data,
40. //与 CP0 相连, 读取其中 CP0 寄存器的值
41. input wire[`RegBus]          cp0_reg_data_i,
42. output reg[4:0]              cp0_reg_read_addr_o,
43. //向下一流水级传递, 用于写 CP0 中的寄存器
44. output reg                   cp0_reg_we_o,
45. output reg[4:0]              cp0_reg_write_addr_o,
46. output reg[`RegBus]          cp0_reg_data_o,
47. output reg[`RegAddrBus]      wd_o,
48. output reg                   wreg_o,
49. output reg[`RegBus] wdata_o,
50. output reg[`RegBus]          hi_o,
51. output reg[`RegBus]          lo_o,
52. output reg                   whilo_o,
53. output reg[`DoubleRegBus]    hilo_temp_o,
54. output reg[1:0]              cnt_o,
55. output reg[`RegBus]          div_opdata1_o,
56. output reg[`RegBus]          div_opdata2_o,
57. output reg                   div_start_o,
58. output reg                   signed_div_o,
59. //下面新增的几个输出是为加载、存储指令准备的
60. output wire[`AluOpBus]       aluop_o,
61. output wire[`RegBus]         mem_addr_o,
62. output wire[`RegBus]         reg2_o,
63. output wire[31:0]            excepttype_o,
64. output wire                  is_in_delayslot_o,
65. output wire[`RegBus]         current_inst_address_o,
66. output reg tallreq
67.
68. );

```

## 5.MEM 模块

- 解释说明：MEM 级部件，主要包含数据存储器等模块。
- 模块设计



## • 接口定义

```

1. module mem(
2.     input wire          rst,
3.     //来自执行阶段的信息
4.     input wire[`RegAddrBus] wd_i,
5.     input wire          wreg_i,
6.     input wire[`RegBus]  wdata_i,
7.     input wire[`RegBus]  hi_i,
8.     input wire[`RegBus]  lo_i,
9.     input wire          whilo_i,
10.    input wire[`AluOpBus] aluop_i,
11.    input wire[`RegBus]  mem_addr_i,
12.    input wire[`RegBus]  reg2_i,
13.    //来自 memory 的信息
14.    input wire[`RegBus]  mem_data_i,
15.    //LLbit_i 是 LLbit 寄存器的值
16.    input wire          LLbit_i,
17.    //但不一定是最新值, 回写阶段可能要写 LLbit, 所以还要进一步判断
18.    input wire          wb_LLbit_we_i,
19.    input wire          wb_LLbit_value_i,
20.    //协处理器 CP0 的写信号
21.    input wire          cp0_reg_we_i,
22.    input wire[4:0]     cp0_reg_write_addr_i,
23.    input wire[`RegBus] cp0_reg_data_i,
24.    input wire[31:0]    excepttype_i,
25.    input wire          is_in_delayslot_i,
26.    input wire[`RegBus] current_inst_address_i,
27.    //CP0 的各个寄存器的值, 但不一定是最新的值, 要防止回写阶段指令写 CP0
28.    input wire[`RegBus] cp0_status_i,
29.    input wire[`RegBus] cp0_cause_i,
30.    input wire[`RegBus] cp0_epc_i,

```

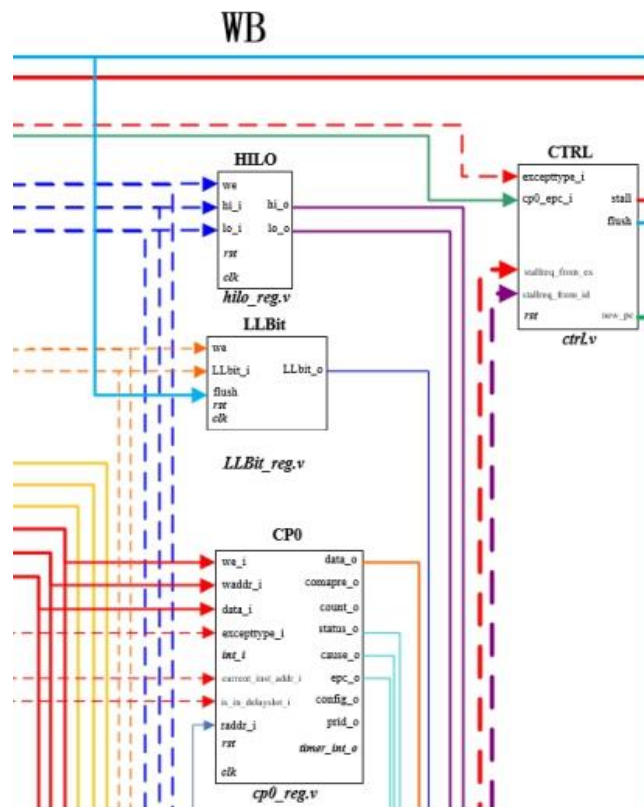
```

31. //回写阶段的指令是否要写 CP0，用来检测数据相关
32.     input wire                wb_cp0_reg_we,
33.     input wire[4:0]          wb_cp0_reg_write_addr,
34.     input wire[`RegBus]      wb_cp0_reg_data,
35. //送到回写阶段的信息
36.     output reg[`RegAddrBus]   wd_o,
37.     output reg                wreg_o,
38.     output reg[`RegBus]       wdata_o,
39.     output reg[`RegBus]       hi_o,
40.     output reg[`RegBus]       lo_o,
41.     output reg                whileo_o,
42.     output reg                LLbit_we_o,
43.     output reg                LLbit_value_o,
44.     output reg                cp0_reg_we_o,
45.     output reg[4:0]           cp0_reg_write_addr_o,
46.     output reg[`RegBus]       cp0_reg_data_o,
47. //送到 memory 的信息
48.     output reg[`RegBus]       mem_addr_o,
49.     output wire               mem_we_o,
50.     output reg[3:0]           mem_sel_o,
51.     output reg[`RegBus]       mem_data_o,
52.     output reg                mem_ce_o,
53.     output reg[31:0]          excepttype_o,
54.     output wire[`RegBus]      cp0_epc_o,
55.     output wire               is_in_delayslot_o,
56.     output wire[`RegBus]      current_inst_address_o
57. );

```

## 6.WB 模块

- 解释说明：WB 级部件，主要包含 HI 寄存器、LO 寄存器、CP0 寄存器等模块。
- 模块设计



## 4.4 测试和调试过程与方法

对于 CPU 系统的测试和调试，主要使用的方法是利用 Vivado 自带的仿真工具进行仿真，观察相关的信号的变化，进行 Debug，最终测试和调试出正确的程序。具体地说，可以使用下面这种方法快速的添加仿真中的信号线。

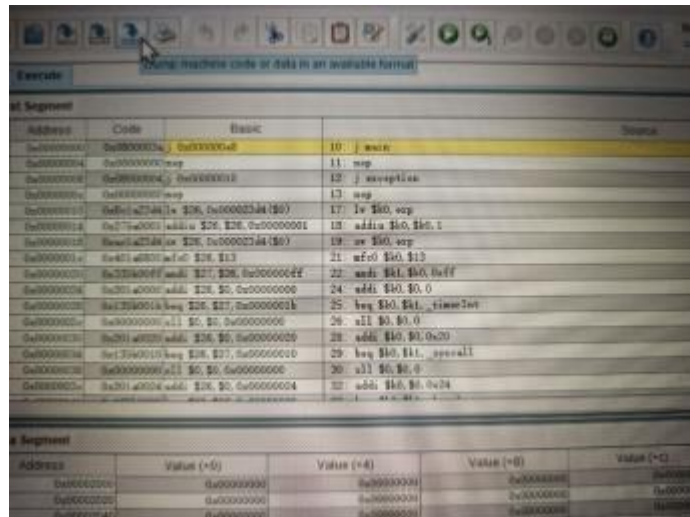
```
assign pc=new1_tb.cpu.inst_addr;
assign inst=new1_tb.cpu.inst;
assign h1=new1_tb.cpu.data_ram0.data_mem0[32'h00000800];
assign h2=new1_tb.cpu.data_ram0.data_mem1[32'h00000800];
assign h3=new1_tb.cpu.data_ram0.data_mem2[32'h00000800];
assign h4=new1_tb.cpu.data_ram0.data_mem3[32'h00000800];
assign h5=new1_tb.cpu.data_test;
} //assign h5=new1_tb.cpu.data_ram0.addr;
//assign h6=new1_tb.cpu.data_ram0.data_addr;
//assign h7=new1_tb.cpu.data_ram0.sel;
//assign h8=new1_tb.cpu.data_ram0.data_i;
//assign h9=new1_tb.cpu.opermips0.cp0_reg0.timer_int_o;
//assign h10=new1_tb.cpu.opermips0.cp0_reg0.excepttype_i;
//assign h11=new1_tb.cpu.opermips0.regfile1.regs[2];
} //assign h12=new1_tb.cpu.opermips0.regfile1.regs[15];
```

在调试中，可以使用“Find Value”等工具和人工排查来进行 Debug 工作：

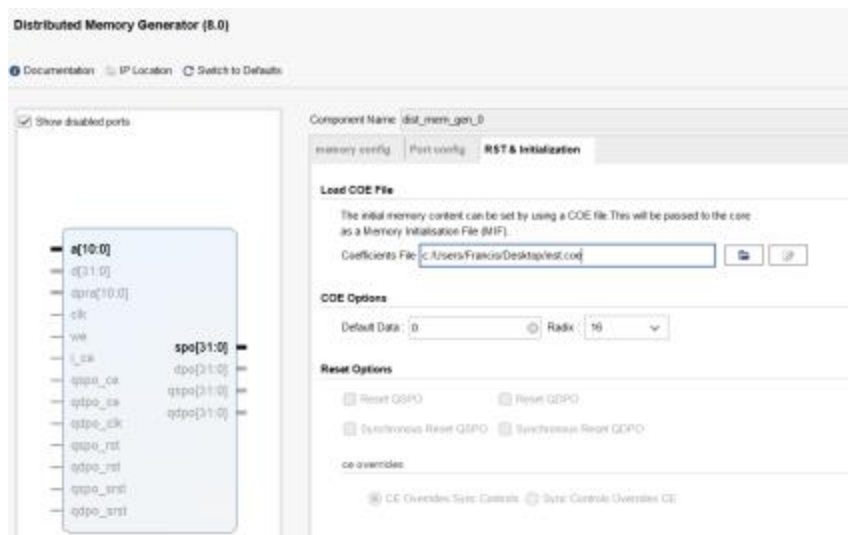


## 4.5 仿真过程与方法

1.使用 Mars4\_5 将测试汇编程序转换成 coe 文件。



2.配置 IP 核。



3.编写 tb 文件。

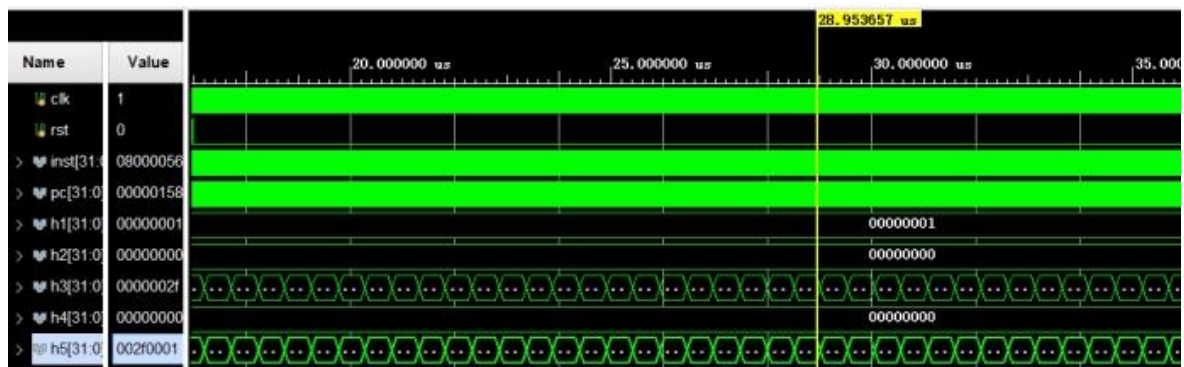
```

openmips_min_soc_tb.v
C:/Users/Francis/Desktop/cz/project_1/src/openmips_min_soc_tb.v

113 ○ $fwrite(fp_w, "regfile24: %h\n", new1_tb.cpu.openmips0.regfile1.regs[24]);
114 ○ $fwrite(fp_w, "regfile25: %h\n", new1_tb.cpu.openmips0.regfile1.regs[25]);
115 ○ $fwrite(fp_w, "regfile26: %h\n", new1_tb.cpu.openmips0.regfile1.regs[26]);
116 ○ $fwrite(fp_w, "regfile27: %h\n", new1_tb.cpu.openmips0.regfile1.regs[27]);
117 ○ $fwrite(fp_w, "regfile28: %h\n", new1_tb.cpu.openmips0.regfile1.regs[28]);
118 ○ $fwrite(fp_w, "regfile29: %h\n", new1_tb.cpu.openmips0.regfile1.regs[29]);
119 ○ $fwrite(fp_w, "regfile30: %h\n", new1_tb.cpu.openmips0.regfile1.regs[30]);
120 ○ $fwrite(fp_w, "regfile31: %h\n", new1_tb.cpu.openmips0.regfile1.regs[31]);
121 ○ $fwrite(fp_w, "pc: %h\n", new1_tb.pc);
122 ○ $fwrite(fp_w, "inst: %h\n", new1_tb.inst);
123 ○ end
124 ○ end
125 ○ assign pc=new1_tb.cpu.inst_addr;
126 ○ assign inst=new1_tb.cpu.inst;
127 ○ assign h1=new1_tb.cpu.data_ram0.data_mem0[32'h00000000];
128 ○ assign h2=new1_tb.cpu.data_ram0.data_mem1[32'h00000000];
129 ○ assign h3=new1_tb.cpu.data_ram0.data_mem2[32'h00000000];
130 ○ assign h4=new1_tb.cpu.data_ram0.data_mem3[32'h00000000];
131 ○ assign h5=new1_tb.cpu.data_test;
132 ○ //assign h5=new1_tb.cpu.data_ram0.addr;
133 ○ //assign h6=new1_tb.cpu.data_ram0.data_addr;
134 ○ //assign h7=new1_tb.cpu.data_ram0.sel;
135 ○ //assign h8=new1_tb.cpu.data_ram0.data_i;
136 ○ //assign h9=new1_tb.cpu.openmips0.cp0_reg0.timer_int_o;
137 ○ //assign h10=new1_tb.cpu.openmips0.cp0_reg0.excepttype_i;
138 ○ //assign h11=new1_tb.cpu.openmips0.regfile1.regs[0];
139 ○ //assign h12=new1_tb.cpu.openmips0.regfile1.regs[15];
140 ○ endmodule

```

4.使用 Vivado 的仿真工具进行仿真。

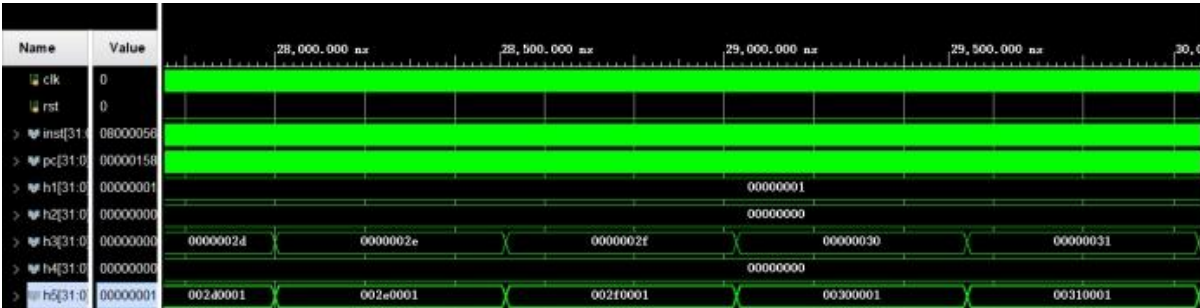


## 5.分析验证

可以看到，当执行完 11 个测试函数后，结果正确，DMEM 的 0 号单元低位为 0x0001。

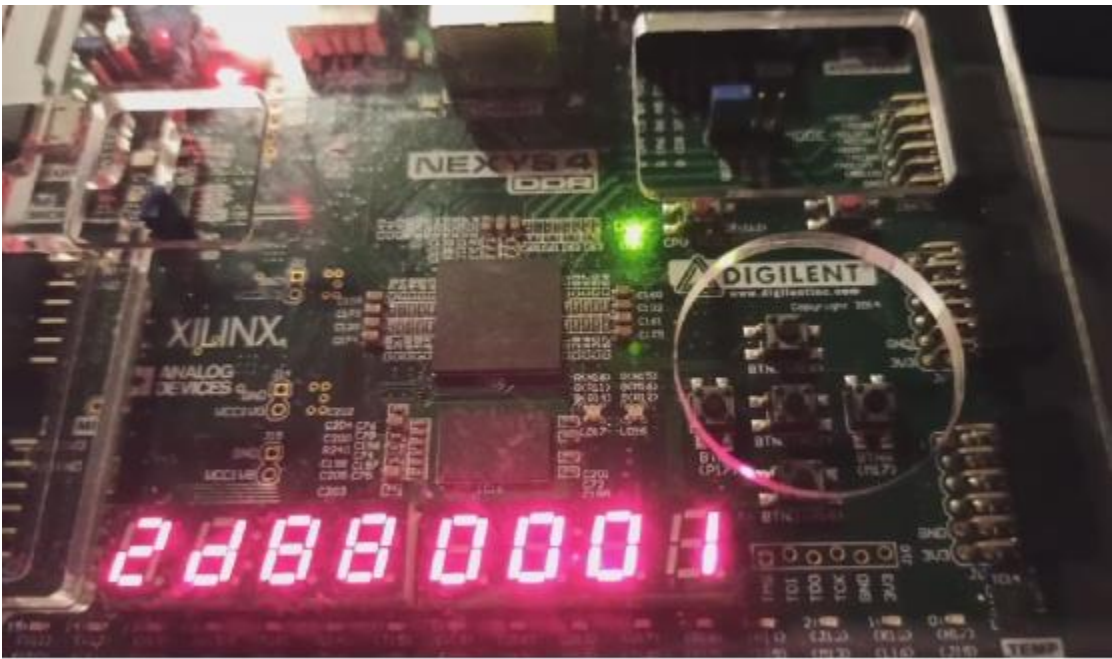


然后程序开始测试时钟中断，可以看到，DMEM 的 0 号单元高位不断增加，表示时钟中断次数的计数。



#### 4.6 下板验证过程与方法

配置约束文件，连接开发板，然后进行综合、生成 bit 文件，下板，观察结果，可以看到数码管低半字显示 0x0001，高半字随着时钟中断而计数，验证结果正确。

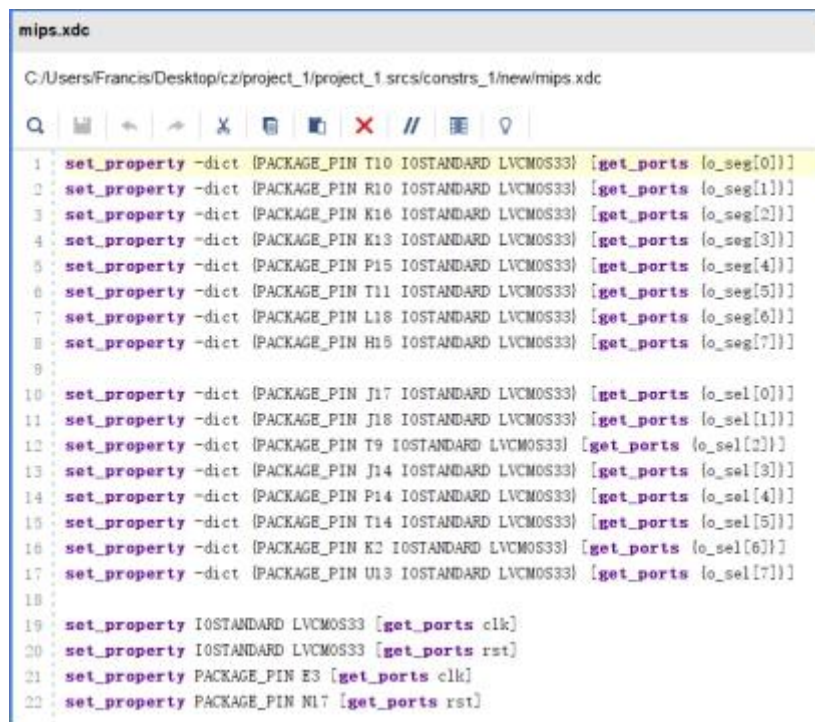


测试参照附件测试程序及说明。

### 5 约束文件修改说明

这次实验中与开发连接的物理接口主要有时钟，复位信号和七段数码管，这

在之前的 CPU 上已经使用到了，只需要将之前多余的接口删去，即可生成这次实验中所需要得约束文件。



```
mips.xdc
C:/Users/Francis/Desktop/cz/project_1/project_1/srcs/constrs_1/new/mips.xdc

1 set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMS33} [get_ports {o_seg[0]}]
2 set_property -dict {PACKAGE_PIN R10 IOSTANDARD LVCMS33} [get_ports {o_seg[1]}]
3 set_property -dict {PACKAGE_PIN K16 IOSTANDARD LVCMS33} [get_ports {o_seg[2]}]
4 set_property -dict {PACKAGE_PIN K13 IOSTANDARD LVCMS33} [get_ports {o_seg[3]}]
5 set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMS33} [get_ports {o_seg[4]}]
6 set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMS33} [get_ports {o_seg[5]}]
7 set_property -dict {PACKAGE_PIN L18 IOSTANDARD LVCMS33} [get_ports {o_seg[6]}]
8 set_property -dict {PACKAGE_PIN H15 IOSTANDARD LVCMS33} [get_ports {o_seg[7]}]
9
10 set_property -dict {PACKAGE_PIN J17 IOSTANDARD LVCMS33} [get_ports {o_sel[0]}]
11 set_property -dict {PACKAGE_PIN J18 IOSTANDARD LVCMS33} [get_ports {o_sel[1]}]
12 set_property -dict {PACKAGE_PIN T9 IOSTANDARD LVCMS33} [get_ports {o_sel[2]}]
13 set_property -dict {PACKAGE_PIN J14 IOSTANDARD LVCMS33} [get_ports {o_sel[3]}]
14 set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMS33} [get_ports {o_sel[4]}]
15 set_property -dict {PACKAGE_PIN T14 IOSTANDARD LVCMS33} [get_ports {o_sel[5]}]
16 set_property -dict {PACKAGE_PIN K2 IOSTANDARD LVCMS33} [get_ports {o_sel[6]}]
17 set_property -dict {PACKAGE_PIN U13 IOSTANDARD LVCMS33} [get_ports {o_sel[7]}]
18
19 set_property IOSTANDARD LVCMS33 [get_ports clk]
20 set_property IOSTANDARD LVCMS33 [get_ports rst]
21 set_property PACKAGE_PIN E3 [get_ports clk]
22 set_property PACKAGE_PIN N17 [get_ports rst]
```

## 6 实验要求

- 1、完成实验，通过所给测试文件测试；
- 2、撰写实验报告