Verilog 流水线处理器

20373864　　谭立德

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 MIPS-C4={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、 SUB、SUBU、 MULT、 MULTU、DIV、 DIVU、SLL、 SRL、 SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、 XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、 BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO、MTC0、MFC0、ERET}。为了实现这些功能，CPU 主要包含了 IM、GRF、DM、ALU、PC、CU 等主要模块，这些模块按照自顶向下的顶层设计逐级展开。

（二）关键模块定义

1. GRF （通用寄存器组，也称为寄存器文件、寄存器堆）

GRF 端口定义：

表 0 GRF 端口表

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，将 32 个寄存器中的值全部清零 1：复位 0：无效
We	I	写使能信号 1：可向 GRF 中写入数据 0：不可向 GEF 中写入数据
A1	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出至 RD1
A2	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出至 RD2
A3	I	5 位地址输入信号，指定 32 个寄存器中的一个作为写入的目标寄存器
WD3	I	32 位数据输入信号

RD1	O	输出指定的寄存器中的 32 位数据
RD2	O	输出指定的寄存器中的 32 位数据

GRF 模块功能定义：

表 1 GRF 功能表

序号	功能名称	描述
1	复位	Reset 信号有效时，所有寄存器储存的数值清零
2	读数据	读出 A1，A2 地址对应寄存器中所储存的数据到 RD1，RD2
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中

2. DM （数据存储器）：

DM 端口定义：

表 2 DM 端口表

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，将 32 个寄存器中的值全部清零 1：复位 0：无效
We	I	写使能信号 1：可向 DM 中写入数据 0：不可向 DM 中写入数据
A	I	5 位地址输入信号，指定中储存器上的地址，将其中存储的数据读出至 RD1
WD	I	32 位数据输入信号
RD	O	输出储存器指定地址上的 32 位数据

DM 模块功能定义：

表 3 DM 功能表

序号	功能名称	描述
1	复位	Reset 信号有效时，储存器储存的所有数值清零
2	读数据	读出 A 地址对应储存器中所储存的数据到 RD
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中

3. ALU （算术逻辑运算单元）：

ALU 端口定义：

表 4 ALU 端口表

信号名	方向	描述
SrcA	I	32 位运算数输入信号
SrcB	I	32 位运算数输入信号
ALU Control	I	3 位逻辑运算选择信号，选择进行哪种逻辑运算
Zero	O	输出比较两运算数比较的 1 位输出
ALU Result	O	输出对两运算数进行指定逻辑运算后的 32 位结果

ALU 模块功能定义：

表 5 ALU 功能表

序号	功能名称	描述
1	计算	根据控制信号进行对应的逻辑计算并输出
2	比较	判断两个输入是否相等

4. IM （指令存储器）：

IM 端口定义：

表 6 IM 端口表

信号名	方向	描述
PC	I	5 位输入地址信号
Instr	O	输出地址所储存 32 位指令

IM 模块功能定义：

表 7 IM 功能表

序号	功能名称	描述
1	读指令	根据输入输出对应 32 位指令

5. Control Unit （指令译码器）：

Control Unit 端口定义：

表 8 Control Unit 端口表

信号名	方向	描述
Opcode[5:0]	I	指令操作码
Funct[5:0]	I	指令功能码

Jump	0	跳转信号
ToHigh16	0	高位置位信号
ExtOp	0	位扩展方式
MemtoReg	0	读内存信号
MemWrite	0	内存写使能信号
Branch	0	分支信号
ALUCtrl[2:0]	0	ALU 控制信号
ALUSrc	0	ALU 操作数 2 的来源 0：寄存器 1：立即数
RegDst	0	寄存器写地址选择 0：Instr[20:16] 1：Instr[15:11]
RegWrite	0	寄存器写使能信号
DMop[1:0]	0	存储、读取方式控制信号

（三）重要机制实现方法

1. J 类型指令

根据输入判断和 ALU 模块协同工作算出跳转地址后跳转。

2. R 类型指令

根据输入判断和 ALU 模块协同工作算出结果后存储回寄存器堆中以实现指令 R 类型指令。

3. I 类型指令

根据输入判断和 ALU 模块和 DM 模块协同工作支持 I 类型指令。

4. 暂停&转发

见附页

二、测试方案

test_input.py > mipsDir

```python
# Coding in UTF-8
import os
import re
import shutil

################################################################################################
f = open("result.txt", "w")


def fileCmp(std_path, ise_path, std, ise, filename):  # file a,b
    stdText = std.read()
    iseText = ise.read()

    isSame = True

    stdLogs = re.findall("@[^\\n]*\\n?", stdText)
    iseLogs = re.findall("@[^\\n]*\\n?", iseText)

    for i in range(len(stdLogs)):
        if (stdLogs[i] != iseLogs[i]):
            isSame = False
            f.write(filename + ":\n")
            f.write("\tWrong: " + "At Line " + str(i) + " : " + "we want " +
                    stdLogs[i] + " " + " but we get " + iseLogs[i] + "\n")
            break
    if (isSame is True):
        print("\tAccepted")
        f.write("Accepted:  " + filename + "\n")
        flag = 1
    else:
```

```python
    else:
        print("\tWrongAnswer")
        flag = 0

    stdLog.close()
    iseLog.close()

    if (flag == 1):
        os.remove(std_path)
        os.remove(ise_path)

    return flag


################################################################################
# mipsDir = input(
#     "需要编译的mips程序绝对地址(e.g. D:/mips.asm): \n")  # Mips File For Mars to Compile

mipsDirs = []
for filename in os.listdir():
    if re.match(r"[\w]+\.asm", filename):
        mipsDirs.append(filename)
hexCodeDir = "code.txt"  # Hex Code For ISE

for mipsDir in mipsDirs:
    #  Dump Hex Code And Get Std Log
    # spMarsJarDir = input("修改版Mars绝对地址(e.g. D:/Mars.jar)\n")
    spMarsJarDir = "Mars_test.jar"  # 修改版Mars地址
    stdLogDir = mipsDir[:-4] + "_std_ans.txt"  # 标准输出
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
```

```
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
              " 100000 db nc mc CompactDataAtZero a dump .text HexText " + hexCodeDir)
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
              " 100000 db nc mc CompactDataAtZero >" + stdLogDir)

    # Prepare ISE exe
    testDir = input("工程文件夹地址(e.g. D:/test): \n")
    # testDir = "E:/ISE/Project_4"
    # tcl文件和prj文件需要放在工程同目录下
    tclFile = open(testDir + "/test.tcl", "w")
    # tcl文件声明了工程运行的参数
    tclFile.write("run 100us;\nexit")
    # prj文件声明了工程所含各模块的位置
    prjFile = open(testDir + "/test.prj", "w")
    for root, dirs, files in os.walk(testDir):
        for fileName in files:
            if re.match(r"[\w]+\.v", fileName):
                prjFile.write("Verilog work " + root + "/" + fileName + "\n")

    tclFile.close()
    prjFile.close()
    # Run ISE simulation
    iseCompileLogDir = "ise_log.txt"
    userLogDir = mipsDir[:-4] + "_ise_ans.txt"  # 我的输出

    ise_path = input("ISE安装文件夹(e.g. D:/Xilinx/14.7/ISE_DS/ISE):\n")
    os.environ['XILINX'] = ise_path
    # os.environ['XILINX'] = "D:/Xilinx/14.7/ISE_DS/ISE"  # ISE安装文件夹

```

```
    os.system(ise_path + "/bin/nt64/fuse -nodebug -prj " +
              testDir + "/test.prj" + " -o " + "testmips.exe mips_test>" +
              iseCompileLogDir)

    os.system("testmips.exe -nolog -tclbatch " + testDir + "/test.tcl" + ">" +
              userLogDir)
    # Mars Log Complete
    stdLog = open(stdLogDir, "r")  # 标准答案
    iseLog = open(userLogDir, "r")  # 你的答案

    fileCmp(stdLogDir, userLogDir, stdLog, iseLog, mipsDir)

    # os.remove("test/code.txt")
    os.remove("fuse.log")
    os.remove("fuse.xmsgs")
    os.remove("fuseRelaunch.cmd")
    os.remove("isim.wdb")
    os.remove("testmips.exe")
    os.remove("ise_log.txt")
    shutil.rmtree("isim")

f.close()

```

(1) CP0 设计

CP0 要干的事就是接收到中断异常时看看是否允许其发生，允许的话记录一下状态交给 handler 处理。

我们要实现 CP0 中的四个寄存器：SR，Cause，EPC，PrID。

SR 表示系统的状态，比如能不能发生异常

Cause 记录异常的信息，比如是否处于延迟槽以及异常的原因

EPC 记录发生异常的位置，便于处理完中断异常的时候返回

PrID 是一个可以随便定义的寄存器，表示你的 CPU 型号

SR 只要实现一部分：SR[15:10]表示允许发生的中断；SR[1]表示是否处于中断异常中（是的话就不能发生中断异常）；SR[0]表示是否允许中断。Cause 也只要实现一部分：Cause[31]表示延迟槽标记；Cause[15:10]表示发生了哪个中断；Cause[6:2]表示异常原因。 为了方便定义一些宏。

```

    `define IM SR[15:10]`define EXL SR[1]

    `define IE SR[0]

    `define BD Cause[31]

    `define hwint_pend Cause[15:10]

    `define ExcCode Cause[6:2]
```

异常和中断的条件：

```
    wire IntReq = (|(HWInt & `IM)) & !`EXL & `IE; // 允许当前中断 且 不在中断异常中 且


```

发生异常的处理方法：

```
    if (Req) begin // int|exc
     `ExcCode <= IntReq ? 5'b0 : ExcCodeIn;
```

```
3
4    `EXL <= 1'b1;
5
6     EPCreg <= tempEPC;

      `BD <= bdIn;end
```

BD：如果异常发生在延迟槽，那么按照要求我们返回的时候要返回跳转指令。所以如果 BD 信号为真时应该输出上一条指令的 PC。

```
1     wire [31:2] tempEPC = (Req) ? (bdIn ? PC[31:2]-1 : PC[31:2])
2      : EPCreg;
3     assign EPCout = {tempEPC, 2'b0};
4
```

每个时钟上升沿都要更新 HWInt：

```
1     `hwint_pend <= HWInt;
```

退出异常的条件是识别到了 eret，我们直接把 EXLClr 接上 M_eret 就好

(2) Bridge 与 IO 设计

桥和 DM

我们的 CPU 把 DM 中一块特殊的区域用来作为与外设交互的接口，中间通过桥来连接。

```
wire selTC1 = (`RAddr >= `StartAddrTC1) && (`RAddr <= `EndAddrTC1),
     selTC2 = (`RAddr >= `StartAddrTC2) && (`RAddr <= `EndAddrTC2);wire TCwe1 = selTC1
     TCwe2 = selTC2 && PrWE;wire [31:0] TCout1, TCout2;wire IRQ1, IRQ2;assign PrRD = se
     selTC2 ? TCout2 :0;wire [5:0] HWInt = {3'b0, interrupt, IRQ2, IRQ1};
```

DM

注意 DM 写入的条件（WE 接口）为 M_WE & (!req)。

内部 always@(posedge clk) 中的也要改（考虑到外设）。

```
if (WE && (addr >= `StartAddrDM) && (addr <= `EndAddrDM)) begin
 // ...end
```

注意下一级寄存器（W_reg）传入 DM 数据时要判断是否是外设的数据。

```
W_REG W_reg(
 // ...
 .DM_in((M_ALUout >= 32'h0000_7f00) ? PrRD : M_DMout),
 // ...);
```

(3)异常中断测试

1.ADEL

(1).ktext 0x4180

```
mfc0    $k0, $14
addu    $k0, $k0, 4
mtc0    $k0, $14
eret
```

.text

```
ori  $28, $0, 0x0000
ori  $29, $0, 0x0000
lui  $8, 0x7000
lui  $9, 0xf000
lw $9,3($0)
sub $10, $8,$9
or   $10, $8, $9
```

(2).ktext 0x4180

```
mfc0    $k0, $14
addu    $k0, $k0, 4
mtc0    $k0, $14
eret
```

.text

```
ori  $28, $0, 0x0000
ori  $29, $0, 0x0000
lui  $8, 0x7000
lui  $9, 0xf000
lh $9,3($0)
sub $10, $8,$9
or   $10, $8, $9
```

(3).text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7fffffff
lui   $9, 0x1
add $10, $8,$9
lw $a0,0x1000($8)
or    $10, $8, $9
```

2.ADES

(1).text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7f00
lui   $9, 0xf000
sw $9,3($0)
sub $10, $8,$9
or    $10, $8, $9
```

(2).text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7f00
lui   $9, 0xf000
sh $9,1($0)
sub $10, $8,$9
or    $10, $8, $9
```

3.RI

在其他测试的机器码中插入 ffffffff

4.Ov

.ktext 0x4180

```
mfc0      $k0, $14
sub $8,$8,$8
```

```
                mtc0    $k0, $14
                eret
.text
                ori  $28, $0, 0x0000
                ori  $29, $0, 0x0000
                lui  $8, 0x7fff
                lui  $9, 0x7fff
                add $10, $8,$9
                or   $10, $8, $9
.text
                ori  $28, $0, 0x0000
                ori  $29, $0, 0x0000
                lui  $8, 0x7fff
                lui  $9, 0x7fff
                addi $10, $8,0x7fff0000
                or   $10, $8, $9
.text
                ori  $28, $0, 0x0000
                ori  $29, $0, 0x0000
                lui  $8, 0x7fff
                ori  $8, $8, 0xffff
                addi$10, $8, 1
ori $a0,$0,100
.text
                ori  $28, $0, 0x0000
                ori  $29, $0, 0x0000
                lui  $8, 0x7000
                lui  $9, 0xf000
                sub $10, $8,$9
                or   $10, $8, $9
```

```
.text
        ori   $28, $0, 0x0000
        ori   $29, $0, 0x0000
        ori   $8, 0x7f00
        lui   $9, 0xf000
        lw $9,0($8)
        sub $10, $8,$9
        or    $10, $8, $9
.text
        ori   $28, $0, 0x0000
        ori   $29, $0, 0x0000
        ori   $8, 0x7f00
        lui   $9, 0xf000
        sh $9,4($8)
        sub $10, $8,$9
        or    $10, $8, $9
```

5.延迟槽

（1）.ktext 0x4180

```
        mfc0 $1,$13
        sub $9,$9,$9
        eret
.text
        ori   $28, $0, 0x0000
        ori   $29, $0, 0x0000
        ori   $8, 0x7fffffff
        ori $9, 0x1000
        j eee
        add $10, $8,$9
        lw $a0,0x1000($8)
        eee:
```

```
                or    $10, $8, $9
```

(2) .ktext 0x4180

```
                mfc0 $1,$13
                sub $9,$9,$9
                eret
.text
                ori   $8, 0x7fffffff
                ori $9, 0x1000
                ori $t1 0x00007f00
                ori $a0,0x0009
                ori $a3,0xfc01
                beq $9,$8,eee
                add $10,$8,$9
                ori $a1,2
                sw $a1,4($t1)
                eee:
                sw $a0,0($t1)
```

（3）.ktext 0x4180

```
                mfc0 $1,$13
                sub $9,$9,$9
                eret
.text
                ori   $8, 0x7fff0000
                ori $9, 0x7fff0000
                ori $a0,0x0009
                ori $a3,0xfc01
                beq $9,$8,eee
                add $10,$8,$9
                ori $a1,2
                sw $a1,4($t1)
```

```
                eee:
                sw $a0,0($t1)
```
（4）.ktext 0x4180
```
                mfc0 $1,$13
                sub $8,$8,$8
                sub $9,$9,$9
                eret
.text
                ori   $8, 0x7fff0000
                ori $9, 0x7fff0000
                ori $a0,0x0009
                ori $a3,0xfc01
                beq $9,$8,eee
                add $10,$8,$9
                ori $a1,2
                sw $a1,4($t1)
                eee:
                sw $a0,0($t1)
```
（5）.ktext 0x4180
```
                mfc0 $1,$13
                sub $9,$9,$9
                eret
.text
                ori   $28, $0, 0x0000
                ori   $29, $0, 0x0000
                ori   $8, 0x7fffffff
                ori $9, 0x1000
                j eee
                add $10, $8,$0
                lw $a0,0x1000($8)
```

```
eee:
or    $10, $8, $9
j end
add $10,$8,$9
end:
ori $a0,$0,0
```

(6) .ktext 0x4180

```
mfc0 $1,$13
sub $9,$9,$9
eret
```

.text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7fffffff
ori $9, 0x1000
j eee
add $10, $8,$0
lw $a0,0x1000($8)
eee:
or    $10, $8, $9
j end
sh $1,1($0)
end:
ori $a0,$0,0
```

(7) .text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7fffffff
ori $9, 0x1000
j eee
```

```
add $10, $8,$0
lw $a0,0x1000($8)
eee:
or    $10, $8, $9
j end
sw $1,1($0)
end:
ori $a0,$0,0
```

(8) .text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7fffffff
ori $9, 0x1000
j eee
add $10, $8,$0
lw $a0,0x1000($8)
eee:
or    $10, $8, $9
j end
lh $1,1($0)
end:
ori $a0,$0,0
```

(9) .text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7fffffff
ori $9, 0x1000
j eee
add $10, $8,$0
lw $a0,0x1000($8)
```

```
eee:
or    $10, $8, $9
j end
lw $1,1($0)
end:
ori $a0,$0,0
```

(10).text

```
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori   $8, 0x7fffffff
ori $9, 0x1000
j eee
add $10, $8,$0
lw $a0,0x1000($8)
eee:
or    $10, $8, $9
j end
lhu $1,1($0)
end:
ori $a0,$0,0
```

6.中断

（1）.ktext 0x4180

```
mfc0 $1,$13
sub $9,$9,$9
mtc0 $0,$12
eret
```

.text

```
ori   $8, 0x7fffffff
ori $9, 0x1000
ori $t1 0x00007f00
```

```
                ori $a0,0x0009
                ori $a3,0xfc01
                ori $a1,2
                sw $a1,4($t1)
                sw $a0,0($t1)
                mtc0 $a3,$12
                or   $10, $8, $9
                ori  $28, $0, 0x0000
                ori  $29, $0, 0x0000
                ori  $28, $0, 0x0010
                ori  $29, $0, 0x0111
                nop
                nop
                nop
```

（2）.ktext 0x4180

```
                mfc0 $1,$13
                sub $9,$9,$9
                mtc0 $0,$12
                eret
.text
                ori  $8, 0x7fffffff
                ori $9, 0x1000
                ori $t1 0x00007f00
                ori $a0,0x0009
                ori $a3,0xfc01
                ori $a1,2
                sw $a1,4($t1)
                sw $a0,0($t1)
                add $11,$8,$9
                mtc0 $a3,$12
```

```
or   $10, $8, $9
ori  $28, $0, 0x0000
ori  $29, $0, 0x0000
ori  $28, $0, 0x0010
ori  $29, $0, 0x0111
nop
nop
nop
```

（3）.ktext 0x4180

```
mfc0 $1,$13
sub $9,$9,$9
sub $29,$29,$29
mtc0 $0,$12
ori $v1,$0,0x00007f00
sw $a3,0($v1)
eret
```

.text

```
ori  $8, 0x7fffffff
ori $9, 0x1000
add $11,$8,$9
ori $t1 0x00007f00
ori $a0,0x0009
ori $a3,0xfc01
ori $a1,1
sw $a1,4($t1)
sw $a0,0($t1)
mtc0 $a3,$12
nop
ori $a0,1111
j eee
```

```
ori   $29, $0, 0x0111
eee:
or    $10, $8, $9
ori   $28, $0, 0x0000
ori   $29, $0, 0x0000
ori $29,0x1000
lui $v1,1
j end
add $10,$8,$29
end:
lui $a0,1
lui $a2,2
```

7.乘除相关

（1）.ktext 0x4180

```
mfc0 $1,$13
sub $a0,$a0,$a0
eret
```

.text

```
ori   $8, 0x7fffffff
ori $9, 0x1000
div $8,$9
lui $a0,0x7f00
add $a0,$a0,$a0
mthi $a0
mflo $s7
ori $a1,1
ori $v0,1
addu $a1,$v0,$v0
nop
ori $s0,11
```

（2）.ktext 0x4180

```
mfc0 $1,$13
mtc0 $0,$12
mflo $s2
mfhi $s1
ori $v1,$0,0x00007f00
sw $a3,0($v1)
eret
```

.text

```
ori $t1,0x00007f00
ori $a0,0x0009
ori $a3,0xfc01
ori $a1,3
sw $a1,4($t1)
sw $a0,0($t1)
mtc0 $a3,$12
mult $t1,$a3
mthi $a3
mfhi $a2
ori $29,0x1000
lui $v1,1
```

（3）mult 中断

.ktext 0x4180

```
mfc0 $1,$13
mtc0 $0,$12
mflo $s2
mfhi $s1
ori $v1,$0,0x00007f00
sw $a3,0($v1)
eret
```

```
.text
        ori $t1,0x00007f00
        ori $a0,0x0009
        ori $a3,0xfc01
        ori $a1,3
        sw $a1,4($t1)
        sw $a0,0($t1)
        mtc0 $a3,$12
        mult $t1,$a3
        mthi $a3
        mfhi $a2
        ori $29,0x1000
        lui $v1,1
(4) .ktext 0x4180
        mfc0 $1,$13
        mtc0 $0,$12
        mflo $s2
        mfhi $s1
        ori $v1,$0,0x00007f00
        sw $a3,0($v1)
        eret
.text
        ori $t1,0x00007f00
        ori $a0,0x0009
        ori $a3,0xfc01
        ori $a1,4
        sw $a1,4($t1)
        sw $a0,0($t1)
        mtc0 $a3,$12
        mult $t1,$a3
```

```
                mthi $a3
                mfhi $a2
                ori $29,0x1000
                lui $v1,1
```

(5) .ktext 0x4180

```
                mfc0 $1,$13
                mtc0 $0,$12
                mflo $s2
                mfhi $s1
                ori $v1,$0,0x00007f00
                sw $a3,0($v1)
                eret
```

.text

```
                ori $t1,0x00007f00
                mult $t1,$t1
                ori $a0,0x0009
                ori $a3,0xfc01
                ori $a1,3
                sw $a1,4($t1)
                sw $a0,0($t1)
                mtc0 $a3,$12
                mult $t1,$a3
                #mthi $a3
                #mfhi $a2
                ori $29,0x1000
                lui $v1,1
```

（6）.ktext 0x4180

```
                mfc0 $1,$13
                mtc0 $0,$12
                mflo $s2
```

```
                mfhi $s1
                ori $v1,$0,0x00007f00
                sw $a3,0($v1)
                eret
.text
                ori $t1,0x00007f00
                mult $t1,$t1
                ori $a0,0x0009
                ori $a3,0xfc01
                ori $a1,8
                sw $a1,4($t1)
                sw $a0,0($t1)
                mtc0 $a3,$12
                ori $29,0x1000
                mthi $a3
                ori $a0,0x000b
                sw $a0,0($t1)
                mthi $a3
                mfhi $a2
                ori $29,0x1000
                mult $t1,$t1
                div $a2,$a2
                lui $a0,1
                110@00003000: $ 9 <= 00007f00
                150@00003008: $ 4 <= 00000009
                170@0000300c: $ 7 <= 0000fc01
                190@00003010: $ 5 <= 00000008
                270@00003020: $29 <= 00001000
                310@00003028: $ 4 <= 0000000b
                370@00003034: $ 6 <= 0000fc01
```

390@00003038: $29 <= 00001000

550@00004180: $ 1 <= 00000000

590@00004188: $18 <= 3f010000

610@0000418c: $17 <= 00000000

630@00004190: $ 3 <= 00007f00

870@00003044: $ 4 <= 00010000

三、思考题

1、我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？(Tips：什么是接口？和我们到现在为止所学的有什么联系？)

解：

硬件

目标：CPU 与外设(被控对象)在硬件上连接构成一个有机整体

方法：I/0 接口电路(接口、接口控制器)

软件

目标：控制设备工作方式，完成信息传送

方法：接口控制程序(或驱动程序)

硬软件接口就是计算机硬件与软件的交互手段，人类与电脑等信息机器或人类与程序之间的接口称为用户界面。电脑等信息机器硬件组件间的接口叫硬件接口。电脑等信息机器软件组件间的接口叫软件接口。硬件接口指的是两个硬件设备之间的连接方式。硬件接口既包括物理上的接口，还包括逻辑上的数据传送协议。软件不同部分之间的交互接口。通常就是所谓的 API－－应用程序编程接口，其表现的形式是源代码。

2、BE 部件对所有的外设都是必要的吗？

解：

不是，timer 只能用 lw,sw,用不到 BE 了。

3、请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图。

解：

异：

模式 0 计时结束后，一直保持中断，直到 en 或 IM 被修改

模式 1 计时结束后，中断一个周期，再重新计数

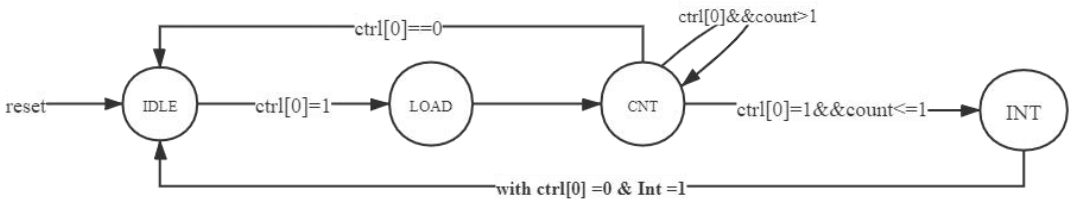
可以理解为中断保持的逻辑不同

同：

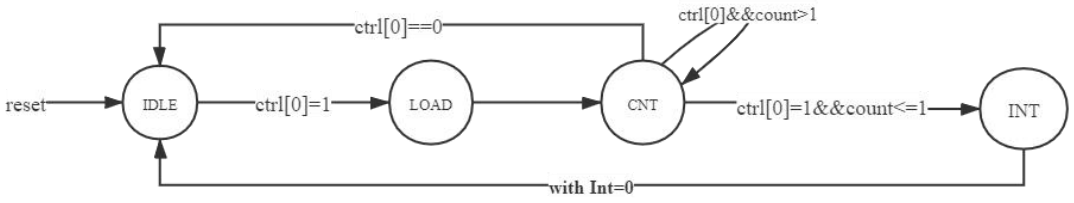
都在 CNT 到 0 的时候产生了中断

2.绘制状态转移图

模式 0：



模式 1：



4、请开发一个主程序以及定时器的 exception handler。整个系统完成如下功能：

（1）定时器在主程序中被初始化为模式 0；

（2）定时器倒计数至 0 产生中断；

（3）handler 设置使能 Enable 为 1 从而再次启动定时器的计数器。(2) 及 (3) 被无限重复。

（4）主程序在初始化时将定时器初始化为模式 0，设定初值寄存器的初值为某个值，如 100 或 1000。(注意，主程序可能需要涉及对 CP0.SR 的编程，推荐阅读过后文后再进行。)

解：

```
    .ktext 0x4180
        mfc0 $1,$13
        mflo $s2
        mfhi $s1
        ori $v1,$0,0x00007f00
        sw $a0,0($v1)
        eret
    .text
        ori $t1,0x00007f00
        mult $t1,$t1
        ori $a0,0x0009
        ori $a3,0xfc01
        ori $a1,2
        sw $a1,4($t1)
        sw $a0,0($t1)
```

```
mtc0 $a3,$12
mult $t1,$a3
ori $29,0x1000
lui $v1,1
lui $v0,2
lui $s1,3
lui $s0,4
```

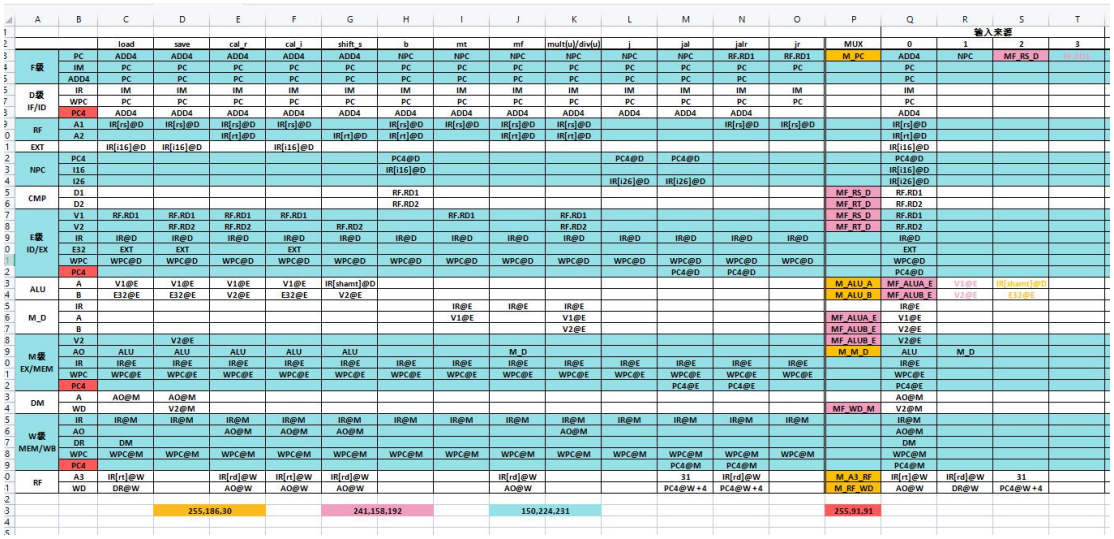
5、请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

解：

设备实际上包括两部分接口控制器（也称为接口芯片）和设备主体，设备主体不直接与主机连接，而是通过接口控制器与主机连接。鼠标和键盘的输入信号相当于中断，当键盘、鼠标有信息时，产生一个中断然后中断例程会从端口读入数据到寄存器。CPU 接收到中断请求之后进入中断处理程序获得鼠标键盘的信息。

附 1：

数据通路



																输入来源			
		load	save	cal_r	cal_i	shift_s	b	mt	mf	mult(u)/div(u)	j	jal	jalr	jr	MUX	0	1	2	3
F级	PC	ADD4	ADD4	ADD4	ADD4	ADD4	NPC	NPC	NPC	NPC	NPC	NPC	RF.RD1	RF.RD1	M_PC	ADD4	NPC	MF_RS_D	M_RT_D
	IM	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC		PC			
	ADD4	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC			PC			
D级 IF/ID	IR	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM		IM			
	WPC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC		PC			
	PC4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4			ADD4			
RF	A1	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D		IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D			IR[rs]@D	IR[rs]@D		IR[rs]@D			
	A2			IR[rt]@D		IR[rt]@D	IR[rt]@D		IR[rt]@D	IR[rt]@D						IR[rt]@D			
EXT		IR[i16]@D	IR[i16]@D		IR[i16]@D											IR[i16]@D			
NPC	PC4						PC4@D				PC4@D	PC4@D				PC4@D			
	I16						IR[i16]@D									IR[i16]@D			
	I26										IR[i26]@D	IR[i26]@D				IR[i26]@D			
CMP	D1						RF.RD1								MF_RS_D	RF.RD1			
	D2						RF.RD2								MF_RT_D	RF.RD2			
E级 ID/EX	V1	RF.RD1	RF.RD1	RF.RD1	RF.RD1			RF.RD1		RF.RD1					MF_RS_D	RF.RD1			
	V2		RF.RD2	RF.RD2		RF.RD2				RF.RD2					MF_RT_D	RF.RD2			
	IR	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D		IR@D			
	E32	EXT	EXT		EXT											EXT			
	WPC	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D		WPC@D			
	PC4											PC4@D	PC4@D			PC4@D			
ALU	A	V1@E	V1@E	V1@E	V1@E	IR[shamt]@D									M_ALU_A	MF_ALUA_E	V1@E	IR[shamt]@D	
	B	E32@E	E32@E	V2@E	E32@E	V2@E									M_ALU_B	MF_ALUB_E	V2@E	E32@E	
M_D	IR							IR@E	IR@E	IR@E						IR@E			
	A							V1@E		V1@E					MF_ALUA_E	V1@E			
	B									V2@E					MF_ALUB_E	V2@E			
M级 EX/MEM	V2		V2@E												MF_ALUB_E	V2@E			
	AO	ALU	ALU	ALU	ALU	ALU			M_D						M_M_D	ALU	M_D		
	IR	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E		IR@E			
	WPC	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E		WPC@E			
	PC4											PC4@E	PC4@E			PC4@E			
DM	A	AO@M	AO@M													AO@M			
	WD		V2@M												MF_WD_M	V2@M			
W级 MEM/WB	IR	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M		IR@M			
	AO			AO@M	AO@M	AO@M				AO@M						AO@M			
	DR	DM														DM			
	WPC	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M		WPC@M			
	PC4											PC4@M	PC4@M			PC4@M			
RF	A3	IR[rt]@W		IR[rd]@W	IR[rt]@W	IR[rd]@W			IR[rd]@W			31	IR[rd]@W		M_A3_RF	IR[rt]@W	IR[rd]@W	31	
	WD	DR@W		AO@W	AO@W	AO@W			AO@W			PC4@W+4	PC4@W+4		M_RF_WD	AO@W	DR@W	PC4@W+4	
			255,186,30			241,158,192			150,224,231						255,91,91				

附 2：

冲突策略矩阵

	Tuse	
指令	rs	rt
addu	1	1
subu	1	1
andi	1	
ori	1	
lw	1	
sw	1	2
beq	0	0
jr	0	
j	-	
jal	-	
lui	-	

		Tnew		
指令	功能部件	E	M	W
addu	ALU	1	0	0
subu	ALU	1	0	0
andi	ALU	1	0	0
ori	ALU	1	0	0
lw	DM	2	1	0
sw	-			
beq	-			
jal	PC	0	0	0
lui	EXT	0	0	0
j	-			
jr	-			

rs策略矩阵

S：暂停

F：转发

Tuse\Tnew	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

rt策略矩阵

S：暂停

F：转发

Tuse\Tnew	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

241,158,192

s_s/s_v/mf -> cal_r

转发：

	0	1	2	3	4	5		ID/EX		EX/MEM				MEM/WB				
	输入0	AO_M	M4	PC8_E	PC8_M	PC8_W		Tnew		Tnew				Tnew				
								jal	jalr	cal_r	cal_i	jal	jalr	cal_r	cal_i	load	jal	jalr
	流水级	源寄存器	涉及指令		MUX	控制信号	输入0	0/31	0/rd	0/rd	O/rt	0/31	0/rd	0/rd	O/rt	O/rt	0/31	0/rd
	D	rs	c_r,c_i,ld,st,b,jr,jalr,s_s		MF_RS_D	F_RS_Dsel	RF.RD1	PC8_E	PC8_E	AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	D	rt	c_r,st,b,s_s		MF_RT_D	F_RT_Dsel	RF.RD2	PC8_E	PC8_E	AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	E	rs	c_r,c_i,ld,st,s_s		MF_ALUA_E	F_ALUA_Esel	V1@E			AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	ALU_E	rt	c_r,st,s_s		MF_ALUB_E	F_ALUB_Esel	V2@E			AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	M	rt	st		MF_WD_M	F_WD_Msel	V2@M							M4	M4	M4	PC8_W	PC8_W
	DM_M																	

暂停：

	IF/ID当前指令			ID/EX			EX/MEM
	指令类型	源寄存器	Tuse	Tnew			Tnew
				cal_r	cal_i	load	load
				1/rd	1/rt	2/rt	1/rt
	beq	rs/rt	0	暂停	暂停	暂停	暂停
	cal_r	rs/rt	1			暂停	
	cal_i	rs	1			暂停	
	load	rs	1			暂停	
	store	rs	1			暂停	
	store	rt	2				
	jr	rs	0	暂停	暂停	暂停	暂停
	jalr	rs	0	暂停	暂停	暂停	暂停

附 3：

控制器

指令	Opcode	RegWrite	DMop_R	DMop_S	ALU_Control	CMP_type	NPC_sel	EXTop	start	PC_sel	ALU_Asel	ALU_Bsel	M_D_sel	A3_RFsel	RF_WDsel
R	000000	1/...	xxx	000	...	xxx	xxx	xx	0	00/...	0	0	0	01/...	00/...
load	.....	1	...	000	0010	xxx	xxx	01	0	00	0	1	0	00	01
save	.....	0	xxx	...	0010	xxx	xxx	01	0	00	0	1	0	xx	xx
cal_i	.....	1	xxx	000	...	xxx	xxx	...	0	00	0	1	0	00	00
sll_s	.....	1	xxx	000	...	xxx	xxx	xx	0	00	1	0	0	01	00
b	.....	0	xxx	000	xxxx	...	010	xx	0	01	x	x	x	xx	xx
addi	001000	1	xxx	000	0010	xxx	xxx	01	0	00	0	1	0	00	00
j	000010	0	xxx	000	xxxx	xxx	001	xx	0	01	x	x	x	xx	xx
jal	000011	1	xxx	000	xxxx	xxx	001	xx	0	01	x	x	x	10	10
R_jr					xxxx				0	10	x	x			
R_jalr					xxxx				0	10	x	x		01	10
R_mt		0			xxxx				0		x	x	x		
R_mf		1			xxxx				0		x	x	1	01	00
R_mult(u)/R_div		0			xxxx				1		x	x	x		

ALU译码器真值表：R类

Funct	ALUControl
100000(add)	010（加）
100010(sub)	110（减）
100100(and)	000（与）
100101(or)	001（或）
101010(slt)	111（小于置位）
100001(addu)	010
100011(subu)	110
001001(jr)	

0000	0001	0010	0011	0100	0101	0110	0111
与	或	加	异或	或非	小于置1	减	符号）小于置1
1000	1001	1010	1011				
lui	逻辑左移	逻辑右移	算数右移				

附 4：

多路选择器控制信号

				优先级		
				低	中	高
功能MUX	控制信号	转发MUX	控制信号	0	1	2
M_PC	PC_sel	MF_RS_D	F_RS_Dsel	RF.RD1	AO@M	M_RF_WD
M_ALU_A	ALU_Asel	MF_RT_D	F_RT_Dsel	RF.RD2	AO@M	M_RF_WD
M_ALU_B	ALU_Bsel	MF_ALUA_E	F_ALUA_Esel	V1@E	AO@M	M_RF_WD
M_M_D	M_Dsel	MF_ALUB_E	F_ALUB_Esel	V2@E	AO@M	M_RF_WD
M_A3_RF	A3_RF_sel	MF_WD_M	F_WD_Msel	V2@M	M_RF_WD	
M_RF_WD	RF_WDsel					