Verilog 单周期处理器

20373864　　谭立德

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 verilog 实现的单周期 MIPS - CPU，支持的指令集包含{ lw、sw、beq、addi、j、ori、lui、add、sub、and、or、addu、subu、lb、sb、lh、sh、jal 、jr、nop }。为了实现这些功能，CPU 主要包含了 IM、GRF、DM、ALU、PC、CU ，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	0	输出比较两运算数比较的 1 位输出
ALU Result	0	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	0	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。

二、测试方案

测试程序:（同 logisim 测试程序）

lui $t0,0x0004　# lui：立即数 0x0004 加载至 t0 寄存器的高位

lui $t1,0x0008　# lui：立即数 0x0008 加载至 t1 寄存器的高位

ori $t3,$zero,0x00002000　　# ori：zero 寄存器中的内容与立即数 0x00002000 进行或运算，储存在 t3 寄存器中

sw $t0,4($t3)　　# sw：把 t0 寄存器中值（1Word），存储到 t3 的值再加上偏移量 4,所指向的 RAM 中

sw $t0,8($t3)　　# sw：把 t0 寄存器中值（1Word），存储到 t3 的值再加上偏移量 8,所指向的 RAM 中

loop:add$t2,$t2,$t1　# add：t1 寄存器中的值加上 t2 寄存器中的值后存到 t2 寄存器中

lw $t4,4($t3)　　# lw：把 t3 寄存器的值+4 当作地址读取存储器中的值存入 t4

lui $t5,0x0004　# lui：立即数 0x0004 加载至 t5 寄存器的高位

sub $t7,$t6,$t5　# sub：t6 寄存器中的值减去 t5 寄存器中的值后存到 t7 寄存器中

sub $t0,$t0,$t5　# sub：t0 寄存器中的值减去 t5 寄存器中的值后存到 t0 寄存器中

add $t6,$t6,$t0　# add：t6 寄存器中的值加上 t0 后存到 t6 寄存器中

beq $t0,$t1,loop　　# beq：判断 t0 的值和 t1 的值是否相等，相等转 loop

add $t0,$t0,$t5　# add：t0 寄存器中的值加上 t5 后存到 t0 寄存器中

lui $v0,0x0001　# lui：立即数 0x0001 加载至 v0 寄存器的高位 lui$v1,0x0002 # lui：立即数 0x0002 加载至 v1 寄存器的高位 add $v0,$v0,$v1　　# add：v0 寄存器中的值加上 v1 后存到 v0 寄存器中

add $v1,$v0,$v1　　# add：v0 寄存器中的值加上 v1 后存到 v1 寄存器中

ori $a0,$v0,0xffff　　# ori：v0 寄存器中的内容与立即数 0xffff 进行或运算，储存在 a0 寄存器中

sub $a1,$a0,0x0000ffff　# sub：a0 寄存器中的值减去立即数 0x0000ffff 后存到 a1 寄存器中

loop2: sub $a2,$v1,$v0　# sub：v1 寄存器中的值减去 v0 中的值后存到 a2 寄存器中

add $a1,$a2,$a1　　# add：a2 寄存器中的值加上 a1 后存到 a1 寄存器中

beq $a1,$v1,loop2　# beq：判断 a1 的值和 v1 的值是否相等，相等转 loop2

对应机器码：3c080004　3c090008　340b2000　ad680004　01495020　8d6c0004　3c0d0004 01cd7822　010d4020　01c87020　1109fff9　010d4020　3c020001　3c030002　00431020　00431820　3444ffff　3c010000　3421ffff　00812822　00623022　00c52820　10a3fffd

```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
```

```python
        flag = 1
    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
```

```
    stdLogDir = mipsDir[:-4] + "_std_ans.txt"  # 标准输出
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
              " nc mc CompactDataAtZero a dump .text HexText " + hexCodeDir)
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
              " 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 10us;\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
```

```
    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()
```

三、思考题

（一）根据你的理解，在下面给出的 DM 的输入示例中，地址信号 addr 位数为什么是[11:2]而不是[9:0]？这个 addr 信号又是从哪里来的？

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

思考 Verilog 语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。

解：

地址为[11:2]时，可以截取传入地址的第 11 位到第 2 位信号。由于 DM 传入地址位字节地址，而存储器是按字存储，相差 4 倍，因此后两位必然为 0，要取第 11 到 2 位作为字地址。addr 信号位寻址结果，因此来自 ALU 运算结果。

（二）在相应的部件中，reset 的优先级比其他控制信号（不包括 clk 信号）都要高，且相应的设计都是同步复位。清零信号 reset 所驱动的部件具有什么共同特点？

解：

数据需要初始加载或者在运算过程中负责储存。

（三）C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分 。

解：

有符号溢出，就是计算结果符号位被进位（或借位）覆盖导致符号错误。在 add、addi 指令中，发生这种情况将会产生 IntegerOverflow 异常信号（错误算术运算）导致程序终止。但是若忽略溢出，二进制补码的加法方式决定有符号加法和无符号加法是等价的，不会发生异常就等价于无符号加法。

（四）根据自己的设计说明单周期处理器的优缺点。

解：

优点：数据通路简单，没有延时槽，没有数据冲突和控制冲突。

缺点：和实际硬件不同，数据存储器和指令存储器分开，而在实际的体系结构中不可能。而且，寻址需要另立加法器，而算术器件是比较耗费晶体管和时间的。此外，由于不同指令由于关键路径不同而延迟时间不同，导致时钟周期由最慢的指令决定，这严重降低执行效率。

附 1：

主译码器真值表：

指令	Opcode	Reg Write	Ext Op	Reg Dst	ToHigh 16	ALU Src	Branch	Mem Write	Memto Reg	R	Jump	DM op	ALU Control	…
R	000000	1	x	01	0	0	0	0	0	1	0	xx	…	
lw	100011	1	1	00	0	1	0	0	1	0	0	00	010	
sw	101011	0	1	x	0	1	0	1	x	0	0	00	010	
beq	000100	0	1	x	0	0	1	0	x	0	0	xx	110	
addi	001000	1	1	00	0	1	0	0	0	0	0	xx	010	
j	000010	0	x	x	0	x	x	0	x	0	1	xx	xxx	
ori	001101	1	0	00	0	1	0	0	0	0	0	xx	001	
lui	001111	1	0	00	1	1	0	0	0	0	0	xx	xxx	
lb	100000	1	1	00	0	1	0	0	1	0	0	01	010	
sb	101000	0	1	x	0	1	0	1	x	0	0	01	010	
lh	100001	1	1	00	0	1	0	0	1	0	0	10	010	
sh	101001	0	1	x	0	1	0	1	x	0	0	10	010	
jal	000011	1	x	10	0	x	x	0	x	0	1	xx	xxx	

附 2：

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)	xxx

附 3：

000	001	010	011	100	101	110	111
与	或	加				减	