2023《数字逻辑与处理器基础》处理器大作业

2023/05/08

一、 实验目的

1. 掌握 MIPS 单周期处理器的控制通路和数据通路的设计原理和 RTL 实现方法；
2. 掌握处理器与外部设备的通信原理和 RTL 实现方法。

二、 指令集

1. MIPS 指令集子集：

lw, sw, lui, add, addu, sub, subu, addi, addiu, and, or, xor, nor, andi, sll, srl, sra, slt, sltu, slti, sltiu, beq, j, jal, jr

2. MIPS 指令格式：

Instruction	OpCode[5:0]	Rs[4:0]	Rt[4:0]	Rd[4:0]	Shamt[4:0]	Funct[5:0]
lw rt, offset (rs)	0x23	rs	rt	offset		
sw rt, offset (rs)	0x2b	rs	rt	offset		
lui rt, imm	0x0f	0	rt	imm		
add rd, rs, rt	0	rs	rt	rd	0	0x20
addu rd, rs, rt	0	rs	rt	rd	0	0x21
sub rd, rs, rt	0	rs	rt	rd	0	0x22
subu rd, rs, rt	0	rs	rt	rd	0	0x23
addi rt, rs, imm	0x08	rs	rt	imm		
addiu rt, rs, imm	0x09	rs	rt	imm		
and rd, rs, rt	0	rs	rt	rd	0	0x24
or rd, rs, rt	0	rs	rt	rd	0	0x25
xor rd, rs, rt	0	rs	rt	rd	0	0x26
nor rd, rs, rt	0	rs	rt	rd	0	0x27
andi rt, rs, imm	0x0c	rs	rt	imm		
sll rd, rt, shamt	0	0	rt	rd	shamt	0
srl rd, rt, shamt	0	0	rt	rd	shamt	0x02
sra rd, rt, shamt	0	0	rt	rd	shamt	0x03
slt rd, rs, rt	0	rs	rt	rd	0	0x2a
sltu rd, rs, rt	0	rs	rt	rd	0	0x2b
slti rt, rs, imm	0x0a	rs	rt	imm		
sltiu rt, rs, imm	0x0b	rs	rt	imm		
beq rs, rt, label	0x04	rs	rt	offset		
j target	0x02	target				
jal target	0x03	target				
jr rs	0	rs	0			0x08

3. MIPS 指令集参考资料

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

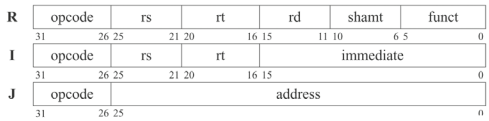
①

CORE INSTRUCTION SET

NAME, MNEMONIC		FORMAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >>> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

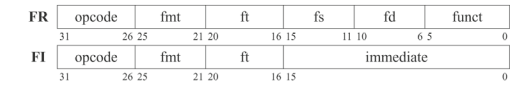


ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FORMAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS



PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

注：jal 指令的 R[31]=PC+8 是针对流水线处理器的情形，这是因为发生跳转时下一条指令(PC+4)已经在执行了，所以需要保存的是接下来的第二条指令(PC+8)。而对于单周期和多周期处理器来说，由于不存在指令同时执行的情况，因此需要保存的是下一条指令(PC+4)。

三、 实验内容

1. **MIPS 单周期 CPU**：*single-cycle* 文件夹中，是单周期处理器的 RTL 实现。请阅读各个基础功能模块的 Verilog 代码，理解每个模块的输入输出接口和基本功能。
   a) 根据对各个控制信号的理解，完成 MIPS 指令集子集与控制信号的真值表（如下表所示，填 0、1、2、x 等），并根据填写的真值表完成 *single-cycle* 文件夹中控制器模块 Control.v 的 Verilog 代码实现。

	PCSrc [1:0]	Branch	RegWrite	RegDst [1:0]	MemRead	MemWrite	MemtoReg [1:0]	ALUSrc1	ALUSrc2	ExtOp	LuOp
lw											
sw											
lui											
add											
addu											
sub											
subu											
addi											
addiu											
and											
or											
xor											
nor											
andi											
sll											
srl											
sra											
slt											
sltu											
slti											
sltiu											
beq											
j											
jal											
jr											

b) 阅读 MIPS Assembly 1 中的指令代码。这段程序运行足够长时间后，寄存器 $v0, $v1, $a0, $a1, $a2, $a3, $t0, $t1, $t2, $t3（分别对应 2-11 号寄存器）中的值应该是多少？

c) 将 Inst-q1.txt 中的代码（对应 MIPS Assembly 1）粘贴至 InstructionMemory.v 的相应位置，使用 ModelSim 或 Vivado 等仿真软件进行仿真，顶层仿真模块为 test_cpu.v。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中的计算结果是否与仿真结果一致，验证单周期处理器的功能正确性。

d) 面向“数逻实验课”所采用的 FPGA 板卡（参见其他说明-5），基于 Vivado 工具进行综合并开展静态时序分析。根据 Vivado 的资源和时序分析报告，分析说明 CPU 所可能达到的最高时钟频率、所使用的硬件资源开销，附上 Vivado 的综合分析资源和时序报告截图。

MIPS Assembly 1	
0	addi $a0, $zero, 12123
1	addiu $a1, $zero, -12345
2	sll $a2, $a1, 16
3	sra $a3, $a2, 16
4	sw $a0 0($zero)
5	beq $a3, $a1, L1
6	lui $a0, 22222
	L1:
7	add $t0, $a2, $a0
8	sra $t1, $t0, 8
9	addi $t2, $zero, -12123
10	slt $v0, $a0, $t2
11	sltu $v1, $a0, $t2
12	lw $t3, 0($zero)
	Loop:
13	j Loop

2. **实现乘法指令：**我们现在需要计算函数$f(x,y)=g(x)-h(y)$，其中$g(x)=(x^4+x^3+x^2+x+1), h(y)=(y^2+y)$。为了计算这个函数，我们决定在该单周期 CPU 中额外支持乘法指令 mul rd, rs, rt。该指令为 R 型指令，实现的功能为实现 rd = rs * rt（rs 和 rt 是有符号 32 位整数，使用补码表示，不必考虑溢出，直接取乘法结果的低 32 位即可）指令格式如下所示：

Instruction	OpCode[5:0]	Rs[4:0]	Rt[4:0]	Rd[4:0]	Shamt[4:0]	Funct[5:0]
mul rd, rs, rt	0x1c	rs	rt	rd	0	0x2

   a) 在前一题的基础上，在 Control.v 和 ALUControl.v 文件中（如有需要，也可以在其他文件中进行相应修改）补充 mul 指令的相关控制逻辑的 RTL 实现，并在 ALU.v 文件中实现 32 位乘法。请简要写出你的设计思路，并把新增的关键代码粘贴在实验报告中。

   b) 阅读 MIPS Assembly 2 中的指令代码。该代码实现了计算$f(x,y)$的功能。这段程序运行足够长时间后，寄存器$s0, $s1, $s2 中的值应该是多少？

   c) 将 Inst-q2.txt 中的代码（对应 MIPS Assembly 2）粘贴至 InstructionMemory.v 的相应位置，使用 ModelSim 或 Vivado 等仿真软件进行仿真，顶层仿真模块为 test_cpu.v。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中的计算结果是否与仿真结果一致，验证乘法指令的功能正确性。

MIPS Assembly 2		
0	addi $a0 $zero 5	# x = 5
1	jal g_x	# calc x^4 + x^3 + x^2 + x + 1
2	addi $s0 $v0 0	# $s0 = g(x)
3	addi $a0 $zero 7	# y = 7
4	jal h_y	# calc y^2 + y
5	addi $s1 $v0 0	# $s1 = h(y)
6	sub $s2 $s0 $s1	# $s2 = f(x, y)

```
    loop:
7       j loop                  # end
    g_x:
8       addi $t0 $zero 1        # partial sum $t0 = 1
9       addi $t1 $a0 0          # $t1 = x
10      add $t0 $t0 $t1         # $t0 = 1 + x
11      mul $t1 $t1 $a0         # $t1 = x^2
12      add $t0 $t0 $t1         # $t0 = 1 + x + x^2
13      mul $t1 $t1 $a0         # $t1 = x^3
14      add $t0 $t0 $t1         # $t0 = 1 + x + x^2 + x^3
15      mul $t1 $t1 $a0         # $t1 = x^4
16      add $t0 $t0 $t1         # $t0 = 1 + x + x^2 + x^3 + x^4
17      addi $v0 $t0 0          # $v0 = $t0
18      jr $ra                  # return g(x)
    h_y:
19      add $t0 $zero $a0       # partial sum $t0 = y
20      mul $t1 $a0 $a0         # $t1 = y^2
21      add $t0 $t0 $t1         # $t0 = y + y^2
22      addi $v0 $t0 0          # $v0 = $t0
23      jr $ra                  # return h(y)
```

3. **实现设备**：为了加速计算，减轻 CPU 的负担，我们计划把计算 g(x)的操作使用外部设备实现（Device.v）。注意设备是通过内存总线挂载到 CPU 上的，所以 CPU 并不需要额外的指令，只需要通过 sw 写入操作数，设备在收到写入指令和数据时初始化有限状态机，并独立于 CPU 完成后续运算。设备中包含：操作数寄存器 reg_op、结果寄存器 reg_ans、工作状态寄存器 reg_start 和有限状态机。其中 reg_op、reg_ans、reg_start 均为 32bit 的寄存器，并映射在 CPU 的内存地址空间中，CPU 可以通过使用 sw 和 lw 访问对应的地址写入或读取寄存器的值，通过内存的地址区别访问的是内存还是该设备。对于该设备的说明如下：
   - 该设备为同步设备，即与 CPU 使用相同的时钟信号。
   - reg_op 用于存储 x 的值。reg_op 对应内存地址 0x40000000。
   - reg_ans 存储设备计算完的 g(x)结果。reg_ans 对应内存地址 0x40000004。
   - reg_start 的值为 0 时，设备不工作；值为任意非 0 值时，设备读取 reg_op 中的值 x 并开始计算 g(x)。计算完成后，设备将计算结果存储到 reg_ans 并将 reg_start 的值重置为 0。reg_start 对应内存地址 0x40000008。
   - 该设备使用 5 个周期计算 g(x)。CPU 需在 5 个周期后访问 reg_ans 的值获取计算结果（提前访问将获取到错误的计算结果）。每个周期进行的操作如下所示，其中 t 是设备内部的寄存器，不对外暴露。
     i. Cycle 1：检测到 reg_start 非零，开始计算，初始化 reg_ans = 1; t = reg_op
     ii. Cycle 2：reg_ans = reg_ans + t; t = t * reg_op
     iii. Cycle 3：reg_ans = reg_ans + t; t = t * reg_op
     iv. Cycle 4：reg_ans = reg_ans + t; t = t * reg_op
     v. Cycle 5：reg_ans = reg_ans + t; reg_start = 0

   a) 在前两题的基础上，修改 CPU.v 中 135-139 行的代码，将设备连接至 CPU 的

内存总线上。之后，请在 Device.v 文件中实现该设备的功能（如有需要，也可以在等其他文件中进行相应修改）。请简要写出你的设计思路，并把关键代码粘贴在实验报告中。

b) 阅读 MIPS Assembly 3 中的指令代码。该代码使用设备实现了 g(x) 的计算（蓝色部分代码）。这段程序运行足够长时间后，寄存器$s0, $s1, $s2 中的值应该是多少？

c) 将 Inst-q3.txt 中的代码（对应 MIPS Assembly 3）粘贴至 InstructionMemory.v 的相应位置，使用 ModelSim 或 Vivado 等仿真软件进行仿真，顶层仿真模块为 test_cpu.v。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中的计算结果是否与仿真结果一致，验证乘法指令的功能正确性。

d) 请比较修改 CPU 以支持新功能（第 2 问）与通过在内存总线上挂载设备实现新功能（第 3 问）这两种实现方式的区别与特点。

MIPS Assembly 3		
0	addi $a0 $zero 5	# x = 5
1	lui $t0 0x4000	
2	addi $t0 $t0 0	# $t0 = address of reg_op
3	sw $a0 0($t0)	# set reg_op = 5
4	lui $t1 0x4000	
5	addi $t1 $t0 8	# $t1 = address of reg_start
6	addi $a1 $zero 1	# $a1 = 1
7	sw $a1 0($t1)	# set reg_start = 1
8	addi $a0 $zero 7	# y = 7
9	jal h_y	# calc y^2 + y
10	addi $s1 $v0 0	# $s1 = h(y)
11	lui $t2 0x4000	
12	addi $t2 $t2 4	# $t2 = address of reg_ans
13	lw $s0 0($t2)	# $s0 = g(x)
14	sub $s2 $s0 $s1	# $s2 = f(x, y)
	loop:	
15	j loop	
	h_y:	
16	add $t0 $zero $a0	# partial sum $t0 = y
17	mul $t1 $a0 $a0	# $t1 = y^2
18	add $t0 $t0 $t1	# $t0 = y + y^2
19	addi $v0 $t0 0	# $v0 = $t0
20	jr $ra	# return h(y)

四、 实验结果与提交材料

1. 请完成上述实验内容 1、2、3，并**撰写实验报告**（word 或者 pdf 版本）。实验报告不必很长，简明扼要阐述清楚，回答对应问题即可。
2. 将**实验报告以及所有代码文件**（包括所有 Verilog 源代码文件、testbench 测试代码文件和 xdc 约束文件）的压缩包提交至网络学堂。

五、 其他说明

1. 处理器大作业第一部分的提交截止时间为十五周周末，即 **6 月 4 日 23 点 59 分**，该 DDL 不会延长，迟交将按时间长短相应扣分，如有特殊请联系助教和老师商量。
2. 我们鼓励讨论，但是要求所有代码与实验报告均独立完成，严禁抄袭，包括使用 ChatGPT！如发现抄袭现象，将上报学校教务处进行处理。
3. 如果被发现抄袭往年“版本”或互相抄袭，会被要求向助教单独解释自己写的代码，如果说不清楚我们将会提交给系里处理。
4. 如对本次处理器大作业有任何问题或建议，请发送邮件至助教邮箱，或在答疑时间进行答疑。
   a) 钟凯 zhongk19@mails.tsinghua.edu.cn
   b) 陈一鸣 cym21@mails.tsinghua.edu.cn
   c) 杨昕昊 yxh21@mails.tsinghua.edu.cn
5. 针对部分没有选修“数逻实验课”的同学，我们提供了数逻实验课的相关资料（如下清华云盘链接所示），供各位同学**自行学习** Verilog 语法和相关工具的使用。
   链接：https://cloud.tsinghua.edu.cn/f/cec824dad54b4dcf9c2a/