2024《数字逻辑与处理器基础》处理器大作业

MIPS 单周期处理器设计

2024/05/15

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. 利用 Vivado 对处理器进行时序和面积分析，掌握评估处理器性能的关键指标；
3. 添加简易神经网络运算单元，掌握从算法映射到硬件的加速原理；

二、 MIPS 指令集

1. MIPS 指令集子集

lw, sw, lui,

add, addu, sub, subu, addi, addiu, mul

and, or, xor, nor, andi, sll, srl, sra, slt, sltu, slti, sltiu,

beq, j, jal, jr, jalr

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
mul rd, rs, rt	0x1c	rs	rt	rd	0	0x02
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
jalr rd, rs	0	rs	0	rd	0	0x09

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

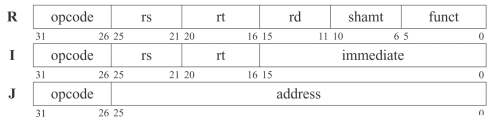
①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	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

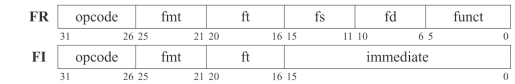


ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FOR-MAT	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

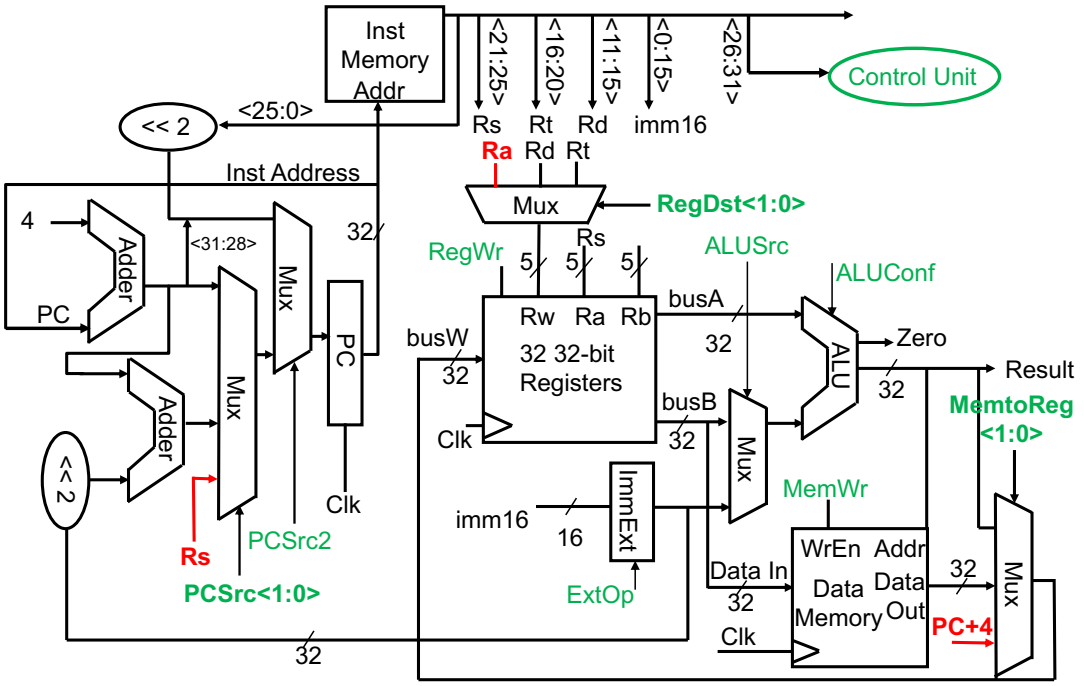
NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

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)。

4. 单周期处理器参考数据通路



注： 添加 jal、jr 等指令需要修改课件上的数据通路，这里给出一种修改方式，供大家参考。

三、 实验内容

1. **MIPS 单周期 CPU 设计** signle-cycle 文件夹中给出了单周期处理器的 RTL 实现。请阅读各个基础模块的 Verilog 代码，理解每个模块的接口和基本功能。
   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											
mul											
and											
or											
xor											
nor											
andi											
sll											
srl											
sra											
slt											
sltu											
slti											
sltiu											
beq											
j											
jal											
jr											

表-1 MIPS 指令集子集与控制信号真值表

b) 阅读 MIPS Assembly 1-1 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, $t2, $t3 （分别对应 2-11 号寄存器）中的值应该是多少？

c) 将 Inst-q1-1.txt 中的代码粘贴至 InstructionMemory.v 的相应位置，Data-q1.txt 中的代码粘贴至 DataMemory.v 的相应位置；以 test_cpu.v 为顶层文件进行仿真。请给出 b 问中所有寄存器的仿真波形图，验证计算结果和仿真结果是否一致，验证单周期处理器的正确性。

MIPS Assembly 1-1	
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

MIPS Assembly 1-2	
	INIT:
0	addi $a0, $zero, 0
1	addi $a1, $zero, 32
2	addi $s0, $zero, 0
	LOOP:
3	lw $t0, 0($a0)
4	lw $t1, 4($a0)
5	mul $t2, $t0, $t1
6	add $s0, $s0, $t2
7	addi $a0, $a0, 8
8	beq $a0, $a1, L1
9	j LOOP
	L1:
10	sw $s0, 0($a1)
	END:
11	j END

d) 阅读 MIPS Assembly 1-2 中的指令代码。该代码实现了乘累加运算： $z = \Sigma_{k=0}^{3} x_k y_k$。这段程序运行足够长时间后，寄存器$a0, $a1, $s0 中的值应该是多少？

e) 将 Inst-q1-2.txt 中的代码（对应 MIPS Assembly 1-2）粘贴至 InstructionMemory.v 的相应位置，Data-q1.txt 中的代码粘贴至 DataMemory.v 的相应位置；使用 Vivado 等软件进行仿真，顶层仿真模块为 test_cpu.v。请给出 d 问中所有寄存器的仿真波形图，验证乘法指令的功能正确性。

f) 基于 Vivado 工具对处理器进行综合并开展静态时序分析。要求使用附件中的约束文件 (xdc_for_both.xdc)，FPGA 型号选择为：xc7a35tcsg324-1。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和硬件资源开销。请在实验报告中附上综合分析资源和时序报告截图。

2. 神经网络加速单元设计 相关研究工作表明，神经网络采用 8 位定点整数进行推理计算，在图像识别等应用中的精度损失通常可以忽略不计，而计算硬件开销显著下降。为此，在实验内容 1 实现的 MIPS 单周期处理器的基础上，我们添加一个低位宽 SIMD（Single Instruction Multiple Data, 单周期多数据）向量处理单元来加速神经网络中的矩阵乘法。如表-2 所示，要求在精简 MIPS 指令集中添加一条低位宽乘累加运算指令：mac。功能为：将 32 位寄存器$rs 和$rt 中存放的 2 个 8-bit 向量(x_0,x_1,x_2,x_3)和(y_0,y_1,y_2,y_3)进行乘累加运算(Multiply-accumulate Operations，MAC)，并将最终结果 z 写入寄存器$rd，运算表达式为：$z = \Sigma_{k=0}^{3} x_k y_k$。低位宽数据在 32-bit 寄存器中的存放格式见图-1。

a) 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 mac 指令相关控制逻辑的 RTL 实现，并在 ALU.v 文件中实现 2 个 8-bit 向量的 MAC 运算。(**提示：可参照图-1 和表-2 来编写代码**) 请在实验报告中简要写出你的设计思路，并粘贴关键代码。

b) 阅读 MIPS Assembly 2 中的指令代码。该代码实现了计算神经网络的矩阵乘法。即$Z = XY$。其中X为2×8的矩阵，Y为8×2的矩阵，Z为最终输出的运算结果，$Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix}$ 为最终输出的2×2矩阵，存储在寄存器$s0, $s1, $s2, $s3 中。这段程序运行足够长时间后，寄存器$s0, $s1, $s2, $s3 中的值应该是多少？

c) 将 Inst-q2.txt 的代码粘贴至 InstructionMemory.v 的相应位置，将 Data-q2-q3.txt 的代码粘贴至 DataMemory.v 的相应位置，以 test_cpu.v 为顶层模块进行仿真。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中计算结果是否与仿真结果一致，验证 mac 指令的功能正确性。

d) 请简要分析，实验内容 2 引入的低位宽单周期 mac 指令与实验内容 1 通过组合精简指令实现乘累加运算这两种实现方式的区别与特点。

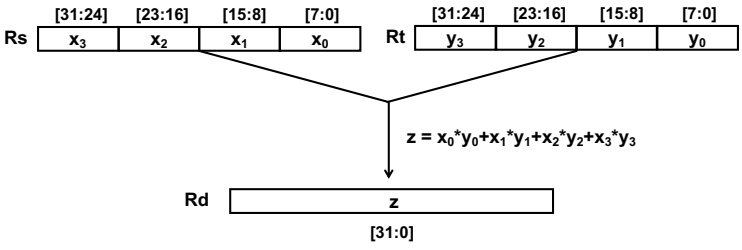


图-1 mac 指令计算示意图

MIPS Assembly 2	
0	lw $t0 0($zero)
1	lw $t1 4($zero)
2	lw $t2 8($zero)
3	lw $t3 12($zero)
4	lw $t4 16($zero)
5	lw $t5 20($zero)
6	lw $t6 24($zero)
7	lw $t7 28($zero)
8	mac $s0 $t0 $t4
9	mac $a0 $t1 $t5
10	add $s0 $s0 $a0
11	mac $s1 $t0 $t6
12	mac $a0 $t1 $t7
13	add $s1 $s1 $a0
14	mac $s2 $t2 $t4
15	mac $a0 $t3 $t5
16	add $s2 $s2 $a0
17	mac $s3 $t2 $t6
18	mac $a0 $t3 $t7
19	add $s3 $s3 $a0
20	sw $s0 32($zero)
21	sw $s1 36($zero)
22	sw $s2 40($zero)
23	sw $s3 44($zero)
	END:
24	j END

Instruction	OpCode[5:0]	Rs[4:0]	Rt[4:0]	Rd[4:0]	Shamt[4:0]	Funct[5:0]
mac rd, rs, rt	0	rs	rt	rd	0	0x2d
relu rd, rs	0	rs	0	rd	0	0x2e

表-2 待添加的神经网络计算指令

3. 激活函数计算

神经网络一般使用非线性激活函数对矩阵乘法结果进行处理。在推理任务中最常见的激活函数是 ReLU 函数，其运算表达式为：$ReLU(z) = \max(z, 0)$。然而，MIPS 精简指令集不能在单个周期内完成该运算。 如表-2 所示，要求添加一条指令：relu。功能为：对寄存器\$rs，\$rd，若\$rs/\$rd 的值大于 0，则保持原值，否则将 0 写入寄存器\$rs/\$rd。

a) 在实验内容 2 的基础上添加 relu 指令数据通路与控制逻辑的 RTL 实现。**(提示：该指令 1 个周期内需要从\$rs, \$rd 读取数据，再写入\$rs 和\$rd，因此需要在寄存器堆中增加一个写端口和写使能信号，并且可能需要修改 ALU 的输出端口)**。请在实验报告中简要写出你的设计思路，并粘贴关键代码。

b) 阅读 MIPS Assembly 3 中的指令代码。该代码实现了计算神经网络的矩阵乘法与激活函数运算。即$Z = ReLU(XY)$。其中X为2×8的矩阵，Y为8×2的矩阵，$Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix}$ 为最终输出的2×2矩阵， 4 个元素$Z_{11}, Z_{12}, Z_{21}, Z_{22}$分别存储在寄存器\$s0, \$s1, \$s2, \$s3 中。这段程序运行足够长时间后，寄存器\$s0, \$s1, \$s2, \$s3 中的值应该是多少？

c) 将 Inst-q3.txt 的代码粘贴至 InstructionMemory.v 的相应位置，将 Data-q2-q3.txt 的代码粘贴至 DataMemory.v 的相应位置。以 test_cpu.v 为顶层模块进行仿真。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中计算结果是否与仿真结果一致，验证 relu 指令的功能正确性。

d) 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和硬件资源开销。请在实验报告中附上综合分析资源和时序报告截图。

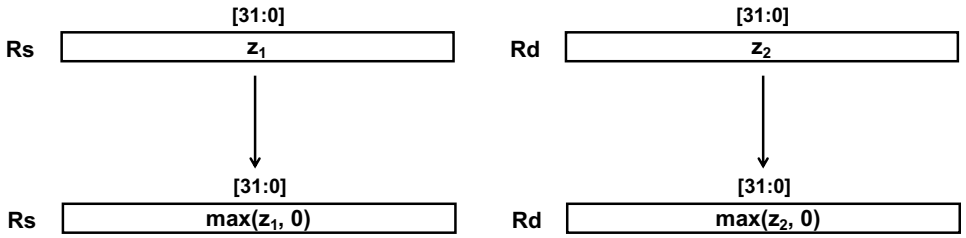


图-2 relu 指令计算示意图

MIPS Assembly 3	
0	lw $t0 0($zero)
1	lw $t1 4($zero)
2	lw $t2 8($zero)
3	lw $t3 12($zero)
4	lw $t4 16($zero)
5	lw $t5 20($zero)
6	lw $t6 24($zero)
7	lw $t7 28($zero)
8	mac $s0 $t0 $t4
9	mac $a0 $t1 $t5
10	add $s0 $s0 $a0
11	mac $s1 $t0 $t6
12	mac $a0 $t1 $t7
13	add $s1 $s1 $a0
14	relu $s1 $s0
15	mac $s2 $t2 $t4
16	mac $a0 $t3 $t5
17	add $s2 $s2 $a0
18	mac $s3 $t2 $t6
19	mac $a0 $t3 $t7
20	add $s3 $s3 $a0
21	relu $s3 $s2
22	sw $s0 32($zero)
23	sw $s1 36($zero)
24	sw $s2 40($zero)
25	sw $s3 44($zero)
	END:
26	j END

四、 实验结果与提交材料

1. **根据实验内容 1 给出的程序作为测试样例，对本次实验给出的 MIPS 指令集子集进行功能性仿真与验证；** 提交实验报告并说明你的设计思路、代码逻辑和验证结果，并附上仿真波形图、源代码、测试代码和 xdc 约束文件；这部分所有文件要求统一存放在 single-cycle 文件夹下；
2. **根据实验内容 2、3 给出的程序作为测试样例，对添加的神经网络计算指令进行功能性仿真与验证；** 提交实验报告并说明你的设计思路、代码逻辑和验证结果，并附上仿真波形图、源代码、测试代码和 xdc 约束文件；为方便批阅，这部分所有文件要求统一存放在 accelerator 文件夹下，和实验内容 1 区分开；
3. **对实验内容 1 和实验内容 2、3 设计的处理器（代码存放在不同文件夹下），要求分别基于 Vivado 工具进行综合并开展静态时序分析**，给出截图，粘贴在实验报告的对应位置，并从时序表现和面积开销等方面分析硬件性能。

五、 其他说明

1. 处理器大作业**提交截止时间为春季学期第 19 周周一，即 7 月 1 日 23 点 59 分（毕业生按教学办通知及时提交）**，迟交将按时间长短相应扣分，如有特殊情况请及时联系老师和助教。
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（包括不恰当地使用 ChatGPT）！如果被发现抄袭往年“版本”或互相抄袭，会被要求向助教、老师单独解释自己写的代码，如果说不清楚我们将会上报系教学办处理
3. 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（czh23@mails.tsinghua.edu.cn），或在答疑时间进行答疑。
4. 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/