数字逻辑与处理器基础大作业
多周期处理器设计

2021/5/10

一、学习目标

1. 掌握 MIPS 多周期处理器的功能模块和完整数据通路的设计原理和 RTL 实现方法；
2. 掌握 MIPS 多周期处理器中基于有限状态机的控制器的设计原理和 RTL 实现方法；
3. 掌握 MIPS 指令集的汇编代码与机器码的相互转换，能阅读并理解汇编程序。

二、MIPS 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, 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
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 指令参考资料

① **MIPS Reference Data**

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

R	opcode	rs	rt	rd	shamt	funct
	31–26	25–21	20–16	15–11	10–6	5–0
I	opcode	rs	rt	immediate		
	31–26	25–21	20–16	15–0		
J	opcode	address				
	31–26	25–0				

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

FR	opcode	fmt	ft	fs	fd	funct
	31–26	25–21	20–16	15–11	10–6	5–0
FI	opcode	fmt	ft	immediate		
	31–26	25–21	20–16	15–0		

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

© 2014 by Elsevier, Inc. All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 5th ed.

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. 附件中提供了去年 MIPS 单周期处理器的题目和实现代码，供各位同学学习和参考。

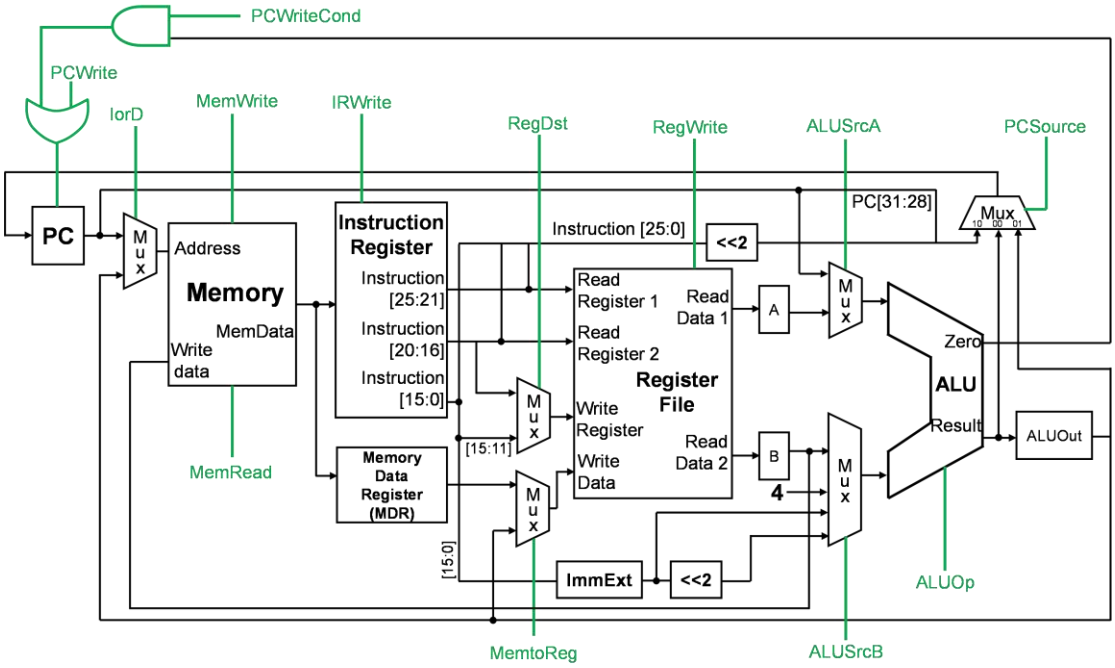
三、多周期处理器设计题目

1　数据通路设计

阅读 code 文件夹中的各个基础功能电路模块的 Verilog 代码，理解每个模块的输入输出接口和实现的基本功能。

1.1　将上述基础功能电路模块进行实例化和连接，完成多周期处理器的整体数据通路设计，即完成 MultiCycleCPU.v 中相关 Verilog 代码的编写。请写出数据通路中包含的寄存器（除寄存器堆之外）和多路选择器，并简要说明它们的功能。

例：课件中的多周期处理器数据通路（仅供实现参考，注意该示例数据通路未包含此次大作业要求的全部指令、控制信号和功能电路模块）



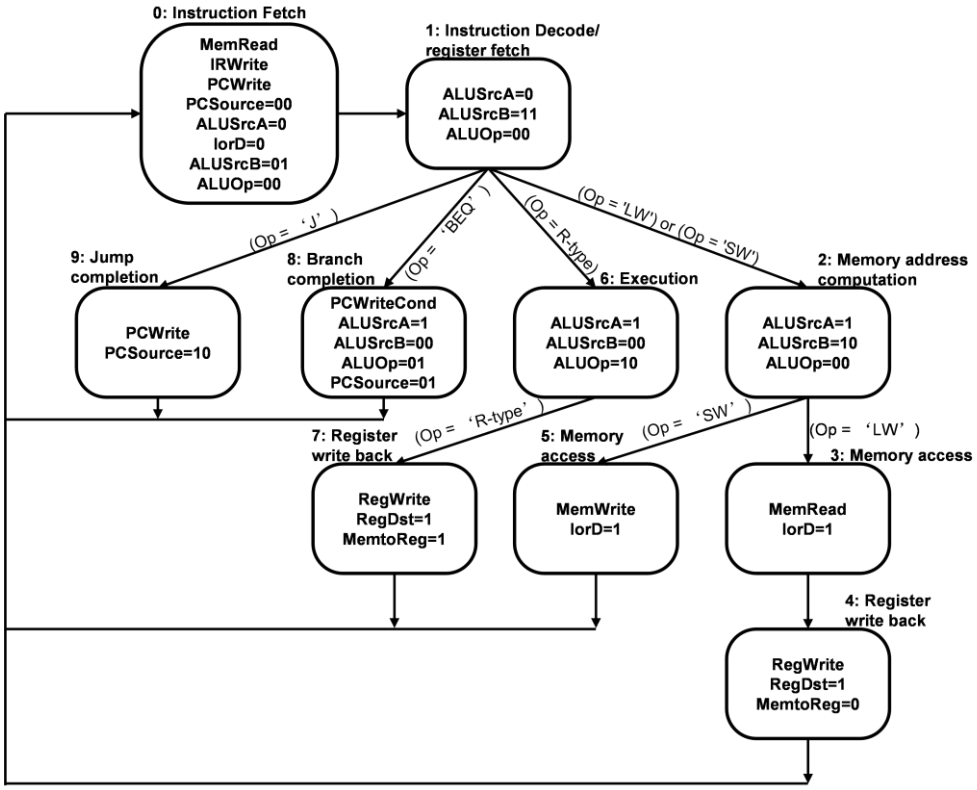
2　控制信号分析与有限状态机实现

2.1　基于上述设计的数据通路和 Controller.v 中的控制信号，请写出每个控制信号所实现的具体控制功能。

例：PCWrite：PC 寄存器的写使能信号；0-不能写 PC；1-允许写 PC。

2.2　依据对 MIPS 指令集和多周期处理器的理解，完成控制器 Controller 中有限状态机的设计，并画出状态转移图。要求状态转移列出具体的指令类型（合并的指令，比如 R-type，需要提前列出所有的 R-type 指令），并且状态图中列出最关键的控制信号。

例：课件中的状态转移图（仅供作图参考，注意该示例状态转移图未包含此次大作业要求的全部指令）



2.3 基于上述设计的有限状态机和控制信号分析，完成控制器 Controller.v 中相关 Verilog 代码的编写。

3 ALU 功能拓展

对于一个元素个数为 32 的全集，使用如下数据格式表征一个集合：对全集中的 32 个元素进行序号分配（0-31），使用 32-bit 寄存器 S 表示某集合，S 中的第 i 位表示该集合是否包含第 i 个元素（0 为不包含，1 为包含）。下面将修改 ALU 电路实现如下功能：输入集合 A 与集合 B（输入数据存储在寄存器中），计算两个集合的差，即 A-B。

3.1 假设该指令为 setsub，请说明该指令的类型和并自行定义对应的机器码字段内容（与现有指令不冲突即可）。

3.2 阅读 ALU.v、ALUControl.v 和 Controller.v，请写出你的设计思路，并完成 ALU 电路模块中相关的 Verilog 代码修改。

3.3 假设输入集合 A 为 0xABCD1234，输入集合 B 为 0xCDEF3456，请自行设计测试指令序列并修改指令存储器文件(InstAndDataMemory.v)，使用 ModelSim 或 Vivado 等仿真软件进行仿真，给出仿真结果示例以说明 ALU 功能拓展的正确性。

4 汇编程序分析-1

阅读 InstAndDataMemory.v 中的指令代码，结合注释理解存储器中的指令程序，对应的 MIPS 汇编指令如下所示：

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

4.1 这段程序运行足够长时间后，寄存器$a0 到$a3，$t0 到 t2，$v1 到$v2 中的值应该是多少？写出计算过程。

4.2 使用 ModelSim 或 Vivado 等仿真软件进行仿真，顶层仿真模块为 test_cpu.v。请给出仿真截图，验证计算结果和所设计的多周期处理器的功能正确性。

5 汇编程序分析-2

阅读并理解下面这段汇编程序：

MIPS Assembly	
0	addi $a0, $zero, 5
1	xor $v0, $zero, $zero
2	jal sum
	Loop:
3	beq $zero, $zero, Loop
	sum:
4	addi $sp, $sp, -8
5	sw $ra, 4($sp)
6	sw $a0, 0($sp)
7	slti $t0, $a0, 1
8	beq $t0, $zero, L1
9	addi $sp, $sp, 8
10	jr $ra
	L1:
11	add $v0, $a0, $v0
12	addi $a0, $a0, -1
13	jal sum
14	lw $a0, 0($sp)

```
15        lw $ra, 4($sp)
16        addi $sp, $sp, 8
17        add $v0, $a0, $v0
18        jr $ra
```

5.1 如果第一行的 5 是任意正整数 n，这段程序能实现什么功能？Loop，sum，L1 各有什么作用？为每一句代码添加注释。

5.2 将这段汇编翻译成机器码并写出，完成 InstAndDataMemory.v 的修改。

5.3 使用 ModelSim 或 Vivado 等仿真软件进行仿真。运行足够长时间后，寄存器$a0，$v0 的值是多少？和你预期的程序功能是否一致？

5.4 观察、描述并解释 PC（只考虑 PC 寄存器写入时的变化）、$a0、$v0、$sp、$ra 如何变化。

6 异常处理（附加思考题，3 分，注：大作业总分 15 分）

试添加电路模块和异常处理指令，并修改数据通路，以实现对算术溢出的异常处理，具体要求如下：

6.1 1）增加 EPC 寄存器，保存异常指令的<u>下一条</u>指令地址（注：由于本次大作业没有与操作系统进行交互，因此无法修改异常指令，异常处理完成后直接跳过异常指令）；2）增加 ErrorTarget 寄存器，当发生溢出异常时，保存算术运算的目标寄存器地址；3）增加异常处理程序的入口地址为 0x7c，该地址存储了一条异常处理指令 errproc，该指令将 0xffffffff 写入目标寄存器地址（缓存在 ErrorTarget 寄存器中）；4）完成异常处理后，继续执行后续的指令程序。根据上述要求，请写出需要添加的控制信号和具体的控制功能。

6.2 请完成异常处理指令 errproc 的设计（包括指令格式和状态机），并画出执行该指令时的状态转移图（不需要画出其他指令，但是需要包括所有的关键控制信号）。

6.3 完成数据通路的修改及电路 Verilog 代码 RTL 实现，自行设计汇编程序，给出异常处理的仿真结果并说明算术溢出异常处理的功能正确性。

四、提交内容

1. 大作业实验报告，要求回答上述所有题目（附加思考题为加分题，大作业总分超过 15 分按 15 分计），并附上相应的仿真截图，格式为 word 或者 pdf 版本。
2. 所有的 Verilog 源代码文件，统一放在 code 目录下。
3. 所有文件打包成 zip 压缩文件，提交至网络学堂课程作业区。

五、考核要求

1. 请在 6 月 10 日 23:59 之前提交大作业，晚交将直接影响最终成绩（直接在 15 分总分上扣除），最终扣分规则以网络学堂公告为准。

2. 6 月 10 日之后（暂定 6 月 11 日周五）将安排随机抽查，要求与助教当面解释代码设计思路，具体时间安排以网络学堂公告为准。
3. 要求所有代码设计与实验报告独立完成，严禁抄袭，如发现抄袭将上报至学校教务处！