CPU REPORT

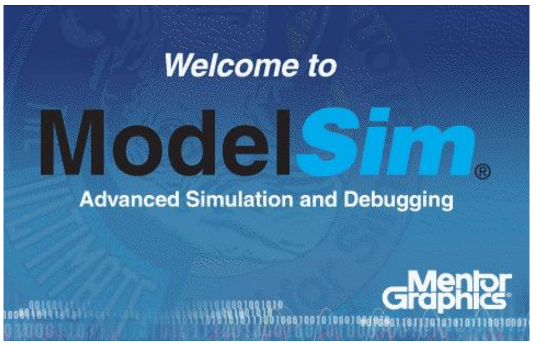
___________made by 徐世超

学号 5140309569

Mips32_cpu

Content

Introduction



这是一个基于 MIPS 指令集架构的 CPU。

该 CPU 采用 Verilog HDL 语言编写。

CPU 在 ModelSim SE-64 10.2c 上编译并仿真。

采用了 5 级流水用于增大吞吐量，并具有一些高级特征。

运行方法

将要运行的指令输入 test.s 文件中

在命令行中运行 assemble.py 文件

将输出文件命名为 data.data

随后就能在 modelsim 中以 mips32_cpu_test.v 为顶端文件进行仿真。

Features

五级流水

流水线是一种实现多条指令重叠执行的加速技术。

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

这个 CPU 将每条指令分割成五个阶段执行：

IF:该阶段从指令储存器中读取指令
ID:该阶段对指令进行译码并且读取寄存器
EX:该阶段执行指令所需操作
MEM:该阶段从数据存储器中读取操作数
WB:该阶段将指令结果写回寄存器

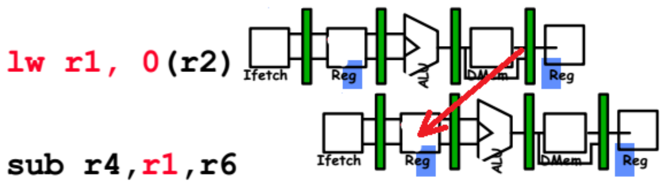
在多条指令需要运行的情况下，该 CPU 获得的加速比趋近于 5

支持 Forwarding

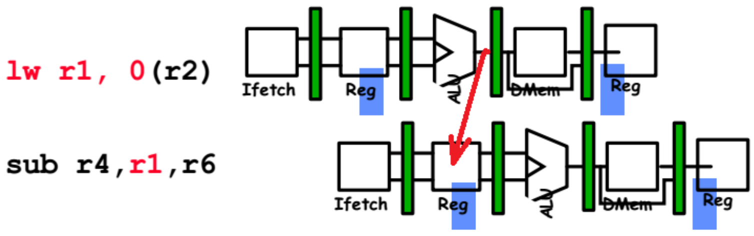
在流水线执行时，可能发生一条指令依赖于前一条指令的结果而导致某些指令读取的数据没有被更新，即 DATA HAZARD。
为了避免 DATA HAZARD，该 CPU 采用 forwarding 的方式将已经执行得出结果的数据前推到译码阶段。

该 CPU 一共增加了如下几条数据通路：

在执行阶段的相关数据前推到译码阶段



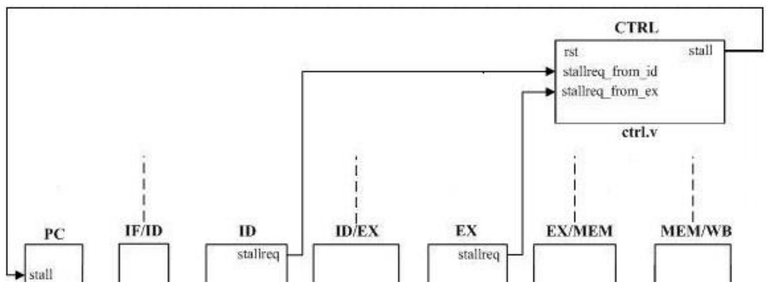
在访存阶段的相关数据前推到译码阶段



```
if((ex_write == 1'b1) && (ex_write_addr == reg1_addr))
begin reg1 <= ex_write_data; end
else if((mem_write == 1'b1) && (mem_write_addr == reg1_addr))
begin reg1 <= mem_write_data; end
```

支持流水线暂停

该 CPU 新增了 control 模块用于支持流水线暂停，在目前的版本中需要进行流水线暂停的指令会在译码阶段、执行阶段发出暂停请求。当 control 模块收到信号后，该 CPU 采用如下方法执行暂停：

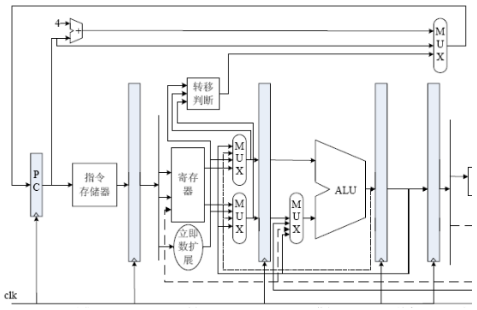


保持当前指令地址不变，同时保持该阶段以及之前的阶段的寄存器不变，在该阶段之后的指令不受影响继续运行。

延迟转移

在跳转指令改写指令地址时，会造成后面两条指令的执行无效。这会浪费两个周期，为了更好的利用每个周期，该 CPU 将是否转移的判断提前至译码阶段，

并且引入延迟槽，加入指令占用其中一个周期，使两个周期的浪费问题得到了一定的改善。



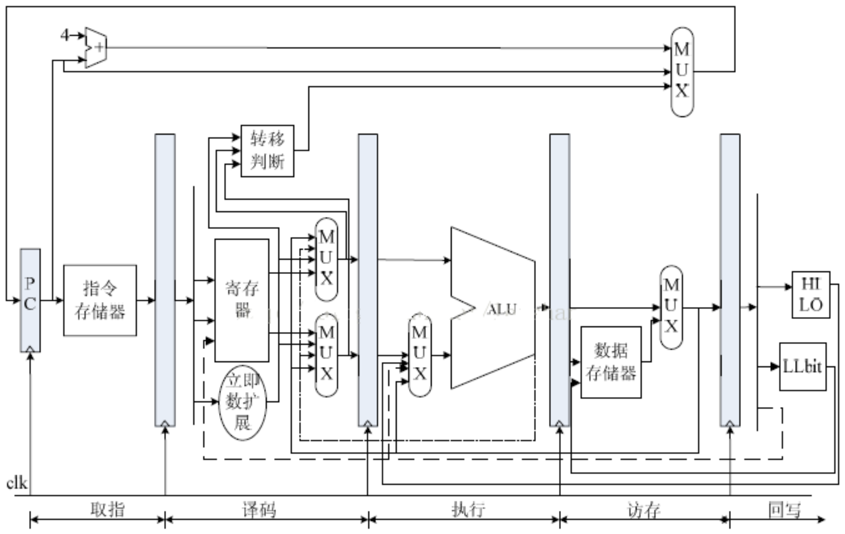
实现方法介绍

在译码阶段将跳转指令信号 stall_sign 和跳转地址的接口 b_addr 连入 pc 模块，

并将延迟槽信息 in_delayslot 传入 ID/EX 模块，在下一个周期传回 ID 模块，标注该指令是否为延迟槽指令

Design

CPU 设计原理简图



程序结构

pc_reg.v:	计算指令地址
if_id.v:	让上一个模块的结果在下一个时钟周期传入下一个模块
register.v:	指令储存器
id.v:	对指令进行译码
id_ex.v:	让上一个模块的结果在下一个时钟周期传入下一个模块
ex.v:	根据上一阶段的译码结果执行指定运算
ex_mem.v:	让上一个模块的结果在下一个时钟周期传入下一个模块
mem.v:	访问数据存储器
mem_wb.v:	让上一个模块的结果在下一个时钟周期传入下一个模块
control.v:	流水线暂停控制模块
div.v:	除法运算模块
hilo.v:hi lo	特殊寄存器的实现模块
cpu_mips32.v:	CPU 各模块连接部分

温馨提示

在设计各个模块时务必要在原理图上标好各个部件的名称以及其宽度，这样写起来会轻松很多。

Reference

《计算机组成与设计 硬件/软件接口》
《计算机体系结构 量化研究方法》
《自己动手做 CPU》

附录

Instructions

ADD – *Add (with overflow)*

Description:	Adds two registers and stores the result in a register
Operation:	$d = $s + $t; advance_pc (4);
Syntax:	add $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0000

ADDI -- *Add immediate (with overflow)*

Description:	Adds a register and a sign-extended immediate value and stores the result in a register
Operation:	$t = $s + imm; advance_pc (4);
Syntax:	addi $t, $s, imm
Encoding:	0010 00ss ssst tttt iiii iiii iiii iiii

ADDIU -- *Add immediate unsigned (no overflow)*

Description:	Adds a register and a sign-extended immediate value and stores the result in a register
Operation:	$t = $s + imm; advance_pc (4);
Syntax:	addiu $t, $s, imm
Encoding:	0010 01ss ssst tttt iiii iiii iiii iiii

ADDU -- *Add unsigned (no overflow)*

Description:	Adds two registers and stores the result in a register
Operation:	$d = $s + $t; advance_pc (4);
Syntax:	addu $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0001

AND -- *Bitwise and*

Description:	Bitwise ands two registers and stores the result in a register
Operation:	$d = $s & $t; advance_pc (4);
Syntax:	and $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0100

ANDI -- *Bitwise and immediate*

Description:	Bitwise ands a register and an immediate value and stores the result in a register
Operation:	$t = $s & imm; advance_pc (4);
Syntax:	andi $t, $s, imm
Encoding:	0011 00ss ssst tttt iiii iiii iiii iiii

BEQ -- *Branch on equal*

Description:	Branches if the two registers are equal
Operation:	if $s == $t advance_pc (offset << 2)); else advance_pc (4);
Syntax:	beq $s, $t, offset
Encoding:	0001 00ss ssst tttt iiii iiii iiii iiii

BGEZ -- *Branch on greater than or equal to zero*

Description:	Branches if the register is greater than or equal to zero
Operation:	if $s >= 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bgez $s, offset
Encoding:	0000 01ss sss0 0001 iiii iiii iiii iiii

BGEZAL -- *Branch on greater than or equal to zero and link*

Description:	Branches if the register is greater than or equal to zero and saves the return address in $31
Operation:	if $s >= 0 $31 = PC + 8 (or nPC + 4); advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bgezal $s, offset
Encoding:	0000 01ss sss1 0001 iiii iiii iiii iiii

BGTZ -- *Branch on greater than zero*

Description:	Branches if the register is greater than zero
Operation:	if $s > 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bgtz $s, offset
Encoding:	0001 11ss sss0 0000 iiii iiii iiii iiii

BLEZ -- *Branch on less than or equal to zero*

Description:	Branches if the register is less than or equal to zero
Operation:	if $s <= 0 advance_pc (offset << 2)); else advance_pc (4);

Syntax:	blez $s, offset
Encoding:	0001 10ss sss0 0000 iiii iiii iiii iiii

BLTZ -- *Branch on less than zero*

Description:	Branches if the register is less than zero
Operation:	if $s < 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bltz $s, offset
Encoding:	0000 01ss sss0 0000 iiii iiii iiii iiii

BLTZAL -- *Branch on less than zero and link*

Description:	Branches if the register is less than zero and saves the return address in $31
Operation:	if $s < 0 $31 = PC + 8 (or nPC + 4); advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bltzal $s, offset
Encoding:	0000 01ss sss1 0000 iiii iiii iiii iiii

BNE -- *Branch on not equal*

Description:	Branches if the two registers are not equal
Operation:	if $s != $t advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bne $s, $t, offset
Encoding:	0001 01ss ssst tttt iiii iiii iiii iiii

DIV -- *Divide*

Description:	Divides $s by $t and stores the quotient in $LO and the remainder in $HI
Operation:	$LO = $s / $t; $HI = $s % $t; advance_pc (4);
Syntax:	div $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1010

DIVU -- *Divide unsigned*

Description:	Divides $s by $t and stores the quotient in $LO and the remainder in $HI
Operation:	$LO = $s / $t; $HI = $s % $t; advance_pc (4);
Syntax:	divu $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1011

J -- *Jump*

Description:	Jumps to the calculated address
Operation:	PC = nPC; nPC = (PC & 0xf0000000) | (target << 2);
Syntax:	j target
Encoding:	0000 10ii iiii iiii iiii iiii iiii iiii

JAL -- *Jump and link*

Description:	Jumps to the calculated address and stores the return address in $31
Operation:	$31 = PC + 8 (or nPC + 4); PC = nPC; nPC = (PC & 0xf0000000) | (target << 2);
Syntax:	jal target
Encoding:	0000 11ii iiii iiii iiii iiii iiii iiii

JR -- *Jump register*

Description:	Jump to the address contained in register $s
Operation:	PC = nPC; nPC = $s;
Syntax:	jr $s

Encoding:	0000 00ss sss0 0000 0000 0000 0000 1000

LB -- *Load byte*

Description:	A byte is loaded into a register from the specified address.
Operation:	$t = MEM[$s + offset]; advance_pc (4);
Syntax:	lb $t, offset($s)
Encoding:	1000 00ss ssst tttt iiii iiii iiii iiii

LUI -- *Load upper immediate*

Description:	The immediate value is shifted left 16 bits and stored in the register. The lower 16 bits are zeroes.
Operation:	$t = (imm << 16); advance_pc (4);
Syntax:	lui $t, imm
Encoding:	0011 11-- ---t tttt iiii iiii iiii iiii

LW -- *Load word*

Description:	A word is loaded into a register from the specified address.
Operation:	$t = MEM[$s + offset]; advance_pc (4);
Syntax:	lw $t, offset($s)
Encoding:	1000 11ss ssst tttt iiii iiii iiii iiii

MFHI -- *Move from HI*

Description:	The contents of register HI are moved to the specified register.
Operation:	$d = $HI; advance_pc (4);
Syntax:	mfhi $d
Encoding:	0000 0000 0000 0000 dddd d000 0001 0000

MFLO -- *Move from LO*

Description:	The contents of register LO are moved to the specified register.
Operation:	$d = $LO; advance_pc (4);
Syntax:	mflo $d
Encoding:	0000 0000 0000 0000 dddd d000 0001 0010

MULT -- *Multiply*

Description:	Multiplies $s by $t and stores the result in $LO.
Operation:	$LO = $s * $t; advance_pc (4);
Syntax:	mult $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1000

MULTU -- *Multiply unsigned*

Description:	Multiplies $s by $t and stores the result in $LO.
Operation:	$LO = $s * $t; advance_pc (4);
Syntax:	multu $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1001

NOOP -- *no operation*

Description:	Performs no operation.
Operation:	advance_pc (4);
Syntax:	noop
Encoding:	0000 0000 0000 0000 0000 0000 0000 00 00

Note: The encoding for a NOOP represents the instruction SLL $0, $0, 0 which has no side effects. In fact, nearly every instruction that has $0 as its destination register will have no side effect and can thus be considered a NOOP instruction.

OR -- *Bitwise or*

Description:	Bitwise logical ors two registers and stores the result in a register
Operation:	$d = $s | $t; advance_pc (4);
Syntax:	or $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0101

ORI -- *Bitwise or immediate*

Description:	Bitwise ors a register and an immediate value and stores the result in a register
Operation:	$t = $s | imm; advance_pc (4);
Syntax:	ori $t, $s, imm
Encoding:	0011 01ss ssst tttt iiii iiii iiii iiii

SB -- *Store byte*

Description:	The least significant byte of $t is stored at the specified address.
Operation:	MEM[$s + offset] = (0xff & $t); advance_pc (4);
Syntax:	sb $t, offset($s)
Encoding:	1010 00ss ssst tttt iiii iiii iiii iiii

SLL -- *Shift left logical*

Description:	Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in.
Operation:	$d = $t << h; advance_pc (4);
Syntax:	sll $d, $t, h
Encoding:	0000 00ss ssst tttt dddd dhhh hh00 0000

SLLV -- *Shift left logical variable*

Description:	Shifts a register value left by the value in a second register and places the result in a third register. Zeroes are shifted in.
Operation:	$d = $t << $s; advance_pc (4);
Syntax:	sllv $d, $t, $s
Encoding:	0000 00ss ssst tttt dddd d--- --00 0100

SLT -- *Set on less than (signed)*

Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d = 1; advance_pc (4); else $d = 0; advance_pc (4);
Syntax:	slt $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 1010

SLTI -- *Set on less than immediate (signed)*

Description:	If $s is less than immediate, $t is set to one. It gets zero otherwise.
Operation:	if $s < imm $t = 1; advance_pc (4); else $t = 0; advance_pc (4);
Syntax:	slti $t, $s, imm
Encoding:	0010 10ss ssst tttt iiii iiii iiii iiii

SLTIU -- *Set on less than immediate unsigned*

Description:	If $s is less than the unsigned immediate, $t is set to one. It gets zero otherwise.
Operation:	if $s < imm $t = 1; advance_pc (4); else $t = 0; advance_pc (4);
Syntax:	sltiu $t, $s, imm
Encoding:	0010 11ss ssst tttt iiii iiii iiii iiii

SLTU -- *Set on less than unsigned*

Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d = 1; advance_pc (4); else $d = 0; advance_pc (4);
Syntax:	sltu $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 1011

SRA -- *Shift right arithmetic*

Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. The sign bit is shifted in.
Operation:	$d = $t >> h; advance_pc (4);
Syntax:	sra $d, $t, h
Encoding:	0000 00-- ---t tttt dddd dhhh hh00 0011

SRL -- *Shift right logical*

Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in.
Operation:	$d = $t >> h; advance_pc (4);
Syntax:	srl $d, $t, h
Encoding:	0000 00-- ---t tttt dddd dhhh hh00 0010

SRLV -- *Shift right logical variable*

Description:	Shifts a register value right by the amount specified in $s and places the value in the destination register. Zeroes are shifted in.
Operation:	$d = $t >> $s; advance_pc (4);
Syntax:	srlv $d, $t, $s
Encoding:	0000 00ss ssst tttt dddd d000 0000 0110

SUB -- *Subtract*

Description:	Subtracts two registers and stores the result in a register
Operation:	$d = $s - $t; advance_pc (4);
Syntax:	sub $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0010

SUBU -- *Subtract unsigned*

Description:	Subtracts two registers and stores the result in a register
Operation:	$d = $s - $t; advance_pc (4);
Syntax:	subu $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0011

SW -- *Store word*

Description:	The contents of $t is stored at the specified address.
Operation:	MEM[$s + offset] = $t; advance_pc (4);
Syntax:	sw $t, offset($s)
Encoding:	1010 11ss ssst tttt iiii iiii iiii iiii

SYSCALL -- *System call*

Description:	Generates a software interrupt.
Operation:	advance_pc (4);
Syntax:	syscall
Encoding:	0000 00-- ---- ---- ---- ---- --00 1100

The syscall instruction is described in more detail on the System Calls page.

XOR -- *Bitwise exclusive or*

Description:	Exclusive ors two registers and stores the result in a register
Operation:	$d = $s ^ $t; advance_pc (4);
Syntax:	xor $d, $s, $t

Encoding:	0000 00ss ssst tttt dddd d--- --10 0110

XORI -- *Bitwise exclusive or immediate*

Description:	Bitwise exclusive ors a register and an immediate value and stores the result in a register
Operation:	$t = $s ^ imm; advance_pc (4);
Syntax:	xori $t, $s, imm
Encoding:	0011 10ss ssst tttt iiii iiii iiii iiii