Verilog 流水线 CPU 设计文档

一、 CPU 设计方案综述

本 CPU 为 Verilog 实现的流水线 CPU（32 位），支持的指令集包含

{LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、SUBU、MULT、MULTU、DIV、DIVU、SLL、SRL、SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}，并进行了适当的指令扩展。

该流水线 CPU 采用五级流水结构，包括流水级寄存器、主要功能部件、功能控制器、冒险控制器等，采用分布式译码方式，对指令进行流水，并在各级进行译码处理。CPU 支持转发和必要的暂停，处理器顶层包含两个输入端口时钟信号 clk 和复位信号 reset。

二、 关键模块定义

1、 stageF

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
NPC[31:0]	I	输入跳转、分支指令下的下一指令地址

PC_en	I	PC 使能端，高电平有效
instr_F[31:0]	O	根据地址取到的当前指令
PC_F[31:0]	O	F 级当前指令地址
PC8_F[31:0]	O	F 级当前指令地址+8

2、 regD

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
D_en	I	D 级流水线寄存器使能信号
instr_F[31:0]	I	F 级指令
PC_F[31:0]	I	F 级当前指令地址
PC8_F[31:0]	I	当前指令地址+8
instr_D[31:0]	O	D 级指令
PC_D[31:0]	O	D 级当前指令地址
PC8_D[31:0]	O	D 级当前指令地址+8

3、 stageD

信号名	方向	描述
clk	I	时钟信号
reset	I	GRF 复位信号
A1_D[4:0]	I	地址输入信号，将对应地址寄存器的值输出至 RD1
A2_D[4:0]	I	地址输入信号，将对应地址寄存器的值输出至 RD2
A3_W[4:0]	I	地址输入信号，指定要进行写入的寄存器
PC_D[31:0]	I	D 级当前指令地址
PC_W[31:0]	I	W 级当前指令地址
imm16[15:0]	I	16 位立即数
addr26[25:0]	I	26 位地址
EXTOp	I	扩展的方式
RFWr	I	写使能信号
RFWD_W[31:0]	I	要写入寄存器的值
NPCOp[1:0]	I	控制 NPC 进行相应的操作： 00：当前为顺序执行指令，NPC 输出 PC+4 01：当前指令为 beq，作为决定是否跳转的条件之一

		10：当前指令为 jal，NPC 输出 PC31..28||instr_index||02 11：当前指令为 jr，NPC 输出 GRF[rs]
MF_RD1_Sel[1:0]	I	RD1 端口转发 MUX 信号
MF_RD2_Sel[1:0]	I	RD2 端口转发 MUX 信号
RD1_D[31:0]	O	转发后的输出，输出 A1 地址对应的寄存器的值
RD2_D[31:0]	O	转发后的输出，输出 A2 地址对应的寄存器的值
imm32_D[31:0]	O	数据输出信号，输出 EXT 扩展后的 32 位立即数
Next_PC[31:0]	O	下一指令地址

4、 regE

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
E_clr	I	E 级单独的清空信号，用于暂停清空
A3_D[4:0]	I	D 级写入地址
instr_D[31:0]	I	D 级指令
PC_D[31:0]	I	D 级当前指令地址

PC8_D[31:0]	I	D 级当前指令地址+8
RD1_D[31:0]	I	数据输入信号，输出 A1 地址对应的寄存器的值
RD2_D[31:0]	I	数据输入信号，输出 A2 地址对应的寄存器的值
imm32_D[31:0]	I	数据输入信号，输出 EXT 扩展后的 32 位立即数
A3_E[4:0]	O	E 级写入地址
RD1_E[31:0]	O	数据输出信号，输出 A1 地址对应的寄存器的值
RD2_E[31:0]	O	数据输出信号，输出 A2 地址对应的寄存器的值
imm32_E[31:0]	O	数据输出信号，输出 EXT 扩展后的 32 位立即数
instr_E[31:0]	O	E 级指令
PC_E[31:0]	O	E 级当前指令地址
PC8_E[31:0]	O	E 级当前指令地址+8

5、 stageE

信号名	方向	描述
RD1_E[31:0]	I	数据输入信号，输出 A1 地址对应的寄存器的值
RD2_E[31:0]	I	数据输入信号，输出 A2 地址对应的寄存器的值
imm32_E[31:0]	I	数据输入信号，输出 EXT 扩展后的 32 位立即数

RFWD_M[31:0]	I	M 级转发来源
RFWD_W[31:0]	I	W 级转发来源
MF_ALUA_Sel[31:0]	I	ALU.A 转发 MUX 控制信号
MF_ALUB_Sel[31:0]	I	ALU.B 转发 MUX 控制信号
ALUOp[1:0]	I	ALU 控制信号 00：A+B 01：A-B 10：A|B 11：B
Bsel	I	ALUB 端口功能 MUX 控制信号
C_E[31:0]	O	E 级 ALU 计算结果

6、 regM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
instr_E[31:0]	I	E 级指令

PC_E[31:0]	I	E 级当前指令地址
PC8_E[31:0]	I	E 级 PC+8
C_E[31:0]	I	E 级 ALU 计算结果
RD2_E[31:0]	I	E 级 rt 寄存器的值
A3_E[4:0]	I	E 级写入地址
A3_M[4:0]	O	M 级写入地址
C_M[31:0]	O	M 级 ALU 计算结果
RD2_M[31:0]	O	M 级 rt 寄存器的值
instr_M[31:0]	O	M 级指令
PC_M[31:0]	O	M 级当前指令地址
PC8_M[31:0]	O	M 级 PC+8

7、 stageM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
PC_M[31:0]	I	M 级当前指令地址

DMWr	I	DM 写使能
A_M[31:0]	I	DM 写入地址
WD_M[31:0]	I	DM 写入值
RFWD_W[31:0]	I	W 级转发来源
MF_DMWD_Sel	I	DM.WD 端口转发 MUX 控制信号
WDSel_M[1:0]	I	M 级写入值的控制信号
C_M[31:0]	I	M 级 ALU 计算结果
PC8_M[31:0]	I	M 级 PC+8
D_M[31:0]	O	M 级 DM 读出值
RFWD_M[31:0]	O	M 级写入值

8、 regW

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
instr_M[31:0]	I	M 级指令
PC8_M[31:0]	I	M 级 PC+8

D_M[31:0]	I	M 级 DM 读出值
C_M[31:0]	I	M 级 ALU 计算结果
A3_M[4:0]	I	M 级写入地址
A3_W[4:0]	O	W 级写入
instr_W[31:0]	O	W 级指令
D_W[31:0]	O	W 级 DM 读出值
C_W[31:0]	O	W 级 ALU 计算结果
PC8_W[31:0]	O	W 级 PC+8

9、stageW

信号名	方向	描述
D_W[31:0]	I	W 级 DM 读出值
C_W[31:0]	I	W 级 ALU 计算结果
PC8_W[31:0]	I	W 级 PC+8
WDSel_W[1:0]	I	W 级写入值选择信号
RFWD_W[31:0]	O	W 级写入值

10、controller

信号名	方向	描述
instr[31:0]	I	指令
NPCOp[1:0]	O	NPC 控制信号
RFWr	O	GRF 使能信号
EXTOp	O	EXT 控制信号
ALUOp[1:0]	O	ALU 控制信号
DMWr	O	DM 使能信号
A3Sel[1:0]	O	A3 选择信号
WDSel[1:0]	O	WD 端口选择信号
BSel	O	B 选择信号
imm16[15:0]	O	16 位立即数
addr26[25:0]	O	26 位立即数
rs[4:0]	O	rs 寄存器地址
rt[4:0]	O	rt 寄存器地址
rd[4:0]	O	rd 寄存器地址

A3[4:0]	O	当前 GRF 写入地址
Tuse_rs[1:0]	O	D 级 rs 的 Tuse
Tuse_rt[1:0]	O	D 级 rt 的 Tuse
Tnew_E[1:0]	O	rs 或 rt 的 Tnew_E
Tnew_M[1:0]	O	rs 或 rt 的 Tnew_M

11、ATcontroller

信号名	方向	描述
rs_D[4:0]	I	D 级 rs 地址
rt_D[4:0]	I	D 级 rt 地址
rs_E[4:0]	I	E 级 rs 地址
rt_E[4:0]	I	E 级 rt 地址
rt_M[4:0]	I	M 级 rt 地址
A3_E[4:0]	I	E 级 A3 地址
A3_M[4:0]	I	M 级 A3 地址
A3_W[4:0]	I	M 级 A3 地址
Tuse_rs[1:0]	I	D 级 rs 的 Tuse

Tuse_rt[1:0]	I	D 级 rt 的 Tuse
Tnew_E[1:0]	I	rs 或 rtE 级 Tnew
Tnew_M[1:0]	I	rs 或 rt 的 Tnew
MF_RD1_Sel[1:0]	O	CMPA 端口的转发 MUX 的选择信号
MF_RD2_Sel[1:0]	O	CMPB 端口的转发 MUX 的选择信号
MF_ALUA_Sel[1:0]	O	ALUA 端口的转发 MUX 的选择信号
MF_ALUB_Sel[1:0]	O	ALUB 端口的转发 MUX 的选择信号
MF_DMWD_Sel	O	DMWD 端口的转发 MUX 的选择信号
PC_en	O	PC 使能信号
D_en	O	D 级使能
E_clr	O	E 级流水线寄存器清除信号

各级流水寄存器储存值如下

D	E	M	W
instr_D	instr_E	instr_M	instr_W
	RD1_E_raw	C_M	C_W
	RD2_E_raw	RD2_M	D_W

	imm32_E		
	A3_E	A3_M	A3_W
PC_D	PC_E	PC_M	PC_W
PC8_D	PC8_E	PC8_M	PC8_W
	(WDSel_E)	(WDSel_M)	(WDSel_W)

三、 转发与暂停处理

转发

转发点共有五个：CMP.A,CMP.B,ALU.A,ALU.B,DM.WD

最后一级 W 级对 GRF 采用内部转发

对其他为 RFWD_W:C_W,D_W,PC8_W

M 级为 RFWD_M：PC8_M,C_E

E 级为 PC8_E

转发点	0	1	2
GRF.RD1	RD1_D_raw	PC8_E	RFWD_M
GRF.RD2	RD2_D_raw	PC8_E	RFWD_M
ALU.A	RD1_E	RFWD_M	RFWD_W
ALU.B	RD2_E	RFWD_M	RFWD_W

DM.WD	RD2_M	RFWD_W	

转发的判断条件是：当前使用的寄存器地址与转发来源寄存器地址相同，且不为 0，则按照优先等级进行转发。

如果转发来源不进行写入的话，令转发来源寄存器地址为 0（从而实现不转发）。

暂停

在 D 级进行 Tuse 和各级 Tnew 的比较如果 Tuse<Tnew，则进行暂停：

PC_en 无效，保持原值不变，D 级流水线寄存器使能信号无效，保持原值不变。E_clr 有效，清除原流水线寄存器的值，实现暂停。

T 矩阵：

指令	Tuse_rs	Tuse_rt(不用为 3)	Tnew_E（不用为 0)	Tnew_M
addu	01	01	01	00
add	01	01	01	00
subu	01	01	01	00
sub	01	01	01	00
and	01	01	01	00

or	01	01	01	00
xor	01	01	01	00
nor	01	01	01	00
slt	01	01	01	00
sltu	01	01	01	00
sll	11	01	01	00
sllv	01	01	01	00
srl	11	01	01	00
srlv	01	01	01	00
sra	11	01	01	00
srav	01	01	01	00
mfhi	11	11	01	00
mflo	11	11	01	00
mthi	01	01	00	00
mtlo	01	01	00	00
mult	01	01	00	00
multu	01	01	00	00

div	01	01	00	00
divu	01	01	00	00
addi	01	11	01	00
addiu	01	11	01	00
ori	01	11	01	00
andi	01	11	01	00
xori	01	11	01	00
slti	01	11	01	00
sltiu	01	11	01	00
lui	11	11	01	00
lw	01	11	10	01
lh	01	11	10	01
lhu	01	11	10	01
lb	01	11	10	01
lbu	01	11	10	01
sw	01	10	00	00
sh	01	10	00	00

sb	01	10	00	00
beq	00	00	00	00
bne	00	00	00	00
blez	00	00	00	00
bltz	00	00	00	00
bgez	00	00	00	00
bgtz	00	00	00	00
jal	11	11	00	00
Jalr	00	11	00	00
j	11	11	00	00
jr	00	11	00	00
nop	不处理即可			

四、 控制信号取值表

指令	NPCOp[1:0]	RFWr	EXTOp	ALUOp[3:0]	DMWr	A3Sel[1:0]	WDSel[1:0]	BSel	VSel	DAOp[1:0]	SSel	MDOp[3:0]	start	CSel	MD_instr
addu (000000/100001)	00	1	x	0000	0	00	00	0	x	xx	x	xxxx	0	0	0
add (000000/100000)	00	1	x	0000	0	00	00	0	x	xx	x	xxxx	0	0	0
subu (000000/100011)	00	1	x	0001	0	00	00	0	x	xx	x	xxxx	0	0	0
sub (000000/100010)	00	1	x	0001	0	00	00	0	x	xx	x	xxxx	0	0	0
and (000000/100100)	00	1	x	0010	0	00	00	0	x	xx	x	xxxx	0	0	0
or	00	1	x	0011	0	00	00	0	x	xx	x	xxxx	0	0	0

(000000/100101)															
xor (000000/100110)	00	1	x	0100	0	00	00	0	x	xx	x	xxxx	0	0	0
nor (000000/100111)	00	1	x	0101	0	00	00	0	x	xx	x	xxxx	0	0	0
slt (000000/101010)	00	1	x	0111	0	00	00	0	x	xx	x	xxxx	0	0	0
sltu (000000/101011)	00	1	x	1000	0	00	00	0	x	xx	x	xxxx	0	0	0
sll (000000/000000)	00	1	x	1001	0	00	00	0	0	xx	x	xxxx	0	0	0
sllv (000000/000100)	00	1	x	1001	0	00	00	0	1	xx	x	xxxx	0	0	0

srl (000000/000010)	00	1	x	1010	0	00	00	0	0	xx	x	xxxx	0	0	0
srlv (000000/000110)	00	1	x	1010	0	00	00	0	1	xx	x	xxxx	0	0	0
sra (000000/000011)	00	1	x	1011	0	00	00	0	0	xx	x	xxxx	0		0
srav (000000/000111)	00	1	x	1011	0	00	00	0	1	xx	x	xxxx	0	x0	0
mfhi (000000/010000)	00	1	x	xxxx	0	00	00	x	x	xx	x	0000	0	1	1
mflo (000000/010010)	00	1	x	xxxx	0	00	00	x	x	xx	x	0001	0	1	1
mthi	00	0	x	xxxx	0	xx	xx	x	x	xx	x	0010	0	x	1

(000000/010001)															
mtlo (000000/010011)	00	0	x	xxxx	0	xx	xx	x	x	xx	x	0011	0	x	1
mult (000000/011000)	00	0	x	xxxx	0	xx	xx	x	x	xx	x	0100	1	x	1
multu (000000/011001)	00	0	x	xxxx	0	xx	xx	x	x	xx	x	0101	1	x	1
div (000000/011010)	00	0	x	xxxx	0	xx	xx	x	x	xx	x	0110	1	x	1
divu (000000/011011)	00	0	x	xxxx	0	xx	xx	x	x	xx	x	0111	1	x	1
addi (001000)	00	1	1	0000	0	00	01	1	x	xx	x	xxxx	0	0	0

addiu (001001)	00	1	1	0000	0	00	01	1	x	xx	x	xxxx	0	0	0
ori (001101)	00	1	0	0011	0	00	01	1	x	xx	x	xxxx	0	0	0
andi (001100)	00	1	0	0010	0	00	01	1	x	xx	x	xxxx	0	0	0
xori (001110)	00	1	0	0100	0	00	01	1	x	xx	x	xxxx	0	0	0
slti (001010)	00	1	1	0111	0	00	01	1	x	xx	x	xxxx	0	0	0
sltiu (001011)	00	1	1	1000	0	00	01	1	x	xx	x	xxxx	0	0	0
lui (001111)	00	1	0	0110	0	01	00	1	x	xx	x	xxxx	0	0	0
lw (100011)	00	1	1	0000	0	01	01	1	x	00	x	xxxx	0	0	0
lh	00	1	1	0000	0	01	01	1	x	01	1	xxxx	0	0	0

(100001)															
lhu (100101)	00	1	1	0000	0	01	01	1	x	01	0	xxxx	0	0	0
lb (100000)	00	1	1	0000	0	01	01	1	x	10	1	xxxx	0	0	0
lbu (100100)	00	1	1	0000	0	01	01	1	x	10	0	xxxx	0	0	0
sw (101011)	00	0	1	0000	1	xx	xx	1	x	00	x	xxxx	0	0	0
sh (101001)	00	0	1	0000	1	xx	xx	1	x	01	x	xxxx	0	0	0
sb (101000)	00	0	1	0000	1	xx	xx	1	x	10	x	xxxx	0	0	0
beq (000100)	01	0	x	xxxx	0	xx	xx	x	x	xx	x	xxxx	0	x	0
bne (000101)	01	0	x	xxxx	0	xx	xx	x	x	xx	x	xxxx	0	x	0

blez (000110)	01	0	x	xxxx	0	xx	xx	x	x	xx	x	xxxx	0	x	0
bltz (000001)	01	0	x	xxxx	0	xx	xx	x	x	xx	x	xxxx	0	x	0
bgez (000001/00001)	01	0	x	xxxx	0	xx	xx	x	x	xx	x	xxxx	0	x	0
bgtz (000111)	01	0	x	xxxx	0	xx	xx	x	x	xx	x	xxxx	0	x	0
jal (000011)	10	1	x	xx	0	10	10	x	x	xx	x	xxxx	0	x	0
Jalr (000000/001001)	11	1	x	xx	0	00	10	x	x	xx	x	xxxx	0	x	0
j （000010）	10	0	x	xx	0	xx	xx	x	x	xx	x	xxxx	0	x	0
jr (000000/001000)	11	0	x	xx	0	xx	xx	x	x	xx	x	xxxx	0	x	0

五、 测试方案

先进行新增指令的功能性测试，然后进行数据冒险的测试。

```
#Rtype

ori $1,$1,5 #101

ori $2,$2,3 #011

add $3,$1,$2

sub $3,$1,$2

and $3,$1,$2

or $3,$1,$2

xor $3,$1,$2

nor $3,$1,$2

lui $4,0xffff

lui $5,0x7fff

slt $3,$2,$1

slt $3,$1,$2

slt $3,$4,$5
```

```
sltu $3,$4,$5

sll $3,$1,1

sllv $3,$1,$2

srl $3,$1,1

srlv $3,$1,$2

sra $3,$4,1

sra $3,$5,1

srav $3,$4,$2

srav $3,$5,$2

ori $4,$4,0xfffe #-2

mult $4,$2 #-6

mflo $3

mfhi $4

multu $4,$2#6

mflo $3

mfhi $5

div $1,$4

mfhi $3
```

```
mflo $5

divu $1,$4

mfhi $3

mflo $5

mthi $1

mtlo $1

mfhi $1

mflo $1

#Itype

ori $1,$1,5

addi $1,$1,1

addiu $1,$1,1

ori $2,$2,1

andi $1,$1,3

xori $2,$2,2

lui $3,0xffff

slti $2,$2,0xffff

sltiu $2,$2,0xffff
```

```
#Btype

ori $1,$1,1

lui $2,0xf000

lui $3,0x7000

bne $1,$2,test1

lui $2,0xf000

lui $2,0xf000

test1:

lui $2,0xf000

lui $2,0xf000

blez $2,test2

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000

test2:

bltz $0,test3

lui $2,0xf000

lui $2,0xf000
```

```
lui $2,0xf000

test3:

bgtz $1,test4

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000

test4:

bgtz $0,test5

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000

test5:

lui $2,0xf000

bne $1,$3,test6

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000

test6:
```

```
blez $0,test7

lui $2,0xf000

lui $2,0xf000

test7:

bltz $2,test8

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000

test8:

blez $1,test9

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000

test9:

bltz $1,test10

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000
```

```
test10:
bgtz $1,test11
lui $2,0xf000
lui $2,0xf000
lui $2,0xf000
test11:
bgez $0,test12
lui $2,0xf000
lui $2,0xf000
lui $2,0xf000
test12:
bgez $1,test13
lui $2,0xf000
lui $2,0xf000
lui $2,0xf000
test13:
bgez $2,test14
lui $2,0xf000
```

```
lui $2,0xf000

lui $2,0xf000

test14:

lui $2,0xf000

lui $2,0xf000

lui $2,0xf000

  #ls

  lui $1,0x1234

  lui $3,0xffff

  ori $1,$1,0x5678

  ori $3,$3,0xffff

  sw $1,0($zero)

  sh $1,6($zero)

  sb $1,8($zero)

  sh $3,10($zero)

  lw $2,0($zero)

  lh $1,10($zero)

  lhu $1,10($zero)
```

lb $1,10($zero)

lbu $1,10($zero)

期望结果为：mars 中的运行结果

将指令分为 R_ALU 型，R_MDU 型，I 型，B 型，以及 J 型，两两测试数据冒险下的转发和暂停功能，两类指令之间分别插入 0,1,2 条 nop 指令，即可实现覆盖。

在保证功能性的前提下，可以从每类指令中选取一条作为代表，特别的，J 型（不用包括 j 指令）,R_MDU 类中取出四条，进行测试。共计 8*7*3=168 种情况，真正达到覆盖性测试。

测试结果为：mars 中运行结果。

利用自动对拍机对拍。

六、 思考题

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

   因为 MDU 运算出结果需要经过多于 1 个的周期，如果整合进 ALU 会导致 E 级其他计算指令的时间消耗增加，减小了时钟频率。

   因为乘法可能溢出 32 位，除法会产生商和余数，需要用两个寄存器存储结果。

2. 参照你对延迟槽的理解，试解释“乘除槽”。

因为在跳转指令刚进去 D 级的时候，才知道下一个 PC 地址，所以会自然地执行跳转指令的下一条，产生延迟槽；在乘除指令进入 E 级之后，由于没有产生结果，会并行的执行其他非乘除族指令，产生乘除槽。

3. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。(Hint: 考虑 C 语言中字符串的情况)。

   当访问目标不是字的倍数，而又是字节的倍数的时候，按字节访问更有优势。

4. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

   需要写入 GRF 的指令和需要读出 GRF 的指令产生的冲突。

   具体是 R_ALU 型，I 型，R_MDU 型，B 型，J 型（不含 j 指令）之间的冲突。

   设置转发机制。

   对于转发处理不了的，通过 Tuse 和 Tnew 解决除 R_MDU 型指令的冲突。

   对于 R_MDU 型指令，通过 start|busy 来判断是否需要暂停。

   测试样例为以上五类指令之间的两两组合，中间插入 0、1、2 条 nop。

5. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

   将指令分类。

在译码的时候可以通过分类信号产生其他信号，降低了复杂性。

数据冲突的解决可以按指令类别进行，降低了测试的复杂性。