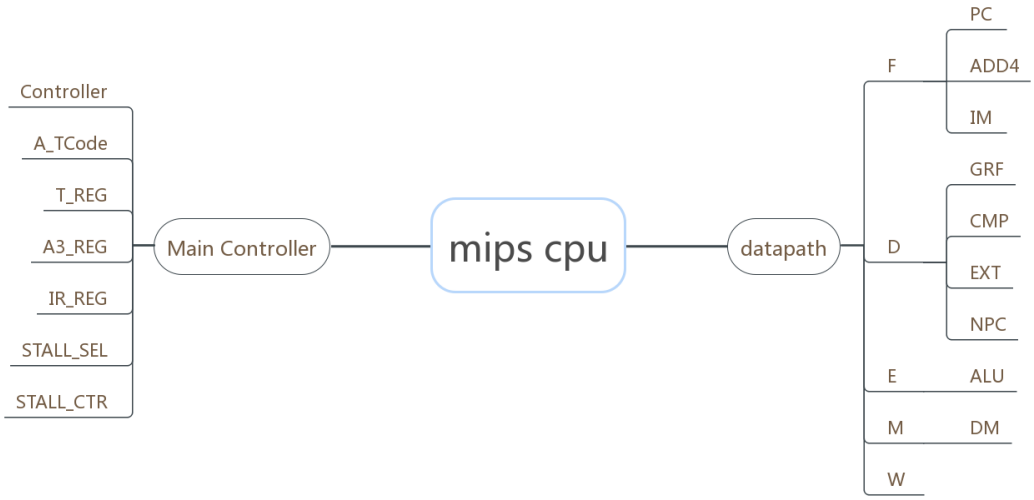
MIPS 流水线 CPU

一、 支持指令

MIPS-lite2={ addu, subu, ori, lw, sw, beq, lui, j, jal, jr, nop}

二、 模块总览



三、 数据通路设计

(一) F 级部件

进行取指令的工作和 pc 的更新。

1. PC（程序计数器）

信号名	方向及位数	功能描述
npc	input [31:0]	读入下一个指令的地址
pc	output [31:0]	输出当前指令地址
en	input	pc 使能信号
clk	input	时钟信号
reset	input	复位信号

2. ADD4

信号名	方向及位数	功能描述
pc	input [31:0]	读入下一个指令的地址
pc4	output [31:0]	输出 pc+4

pc8	output [31:0]	输出 pc+8

3. IM（指令存储器）

信号名	方向及位数	功能描述
IMAddr	input [31:0]	读取指令的地址
IMOut	output [31:0]	取出 pc 对应的指令
寄存器 IMData[1023:0]	reg [31:0]	实际的指令地址从 0 开始存储 读入的 pc 要为(IMAddr-0x00003000)/4

(二) D 级部件

进行指令的解码工作，计算下一个 pc 地址，同时从寄存器堆中读入操作数，给立即数做拓展。

1. NPC

信号名	方向及位数	功能描述
PC_B	input [31:0]	分支指令的地址偏移量
PC_J	input [31:0]	j 指令跳转地址
PC_JR	input [31:0]	jr 指令跳转地址
able	input	b 指令是否跳转（1：跳转）
NPCOp	input [1:0]	下一个 pc 值的选择信号 2'b00:PC+4 2'b01:br 分支指令 2'b10:j 指令 2'b11:jr 指令
NPCOut	output [31:0]	输出下一个指令地址

NPCOp 功能选择信号定义：

功能编号	功能代号	功能描述
2'b00	NPC_add4	NPC = PC+4
2'b01	NPC_br	NPC = PC+4 or PC+4+ PC_branch/4
2'b10	NPC_j	NPC = {pc[31:28],PC_jump[25:0],2'b0}
2'b11	NPC_jr	NPC = PC_jr

2. GRF

信号名	方向	位数	功能描述
RA	in	[4:0]	读取寄存器 A 的下标
RB	in	[4:0]	读取寄存器 B 的下标
RW	in	[4:0]	写入寄存器的下标
WD	in	[31:0]	写入寄存器的数据

WE	in	1	写使能端口 在时钟上升沿到来时，如果 WE 为 1，则将 WD 的数据写入 RW 对应下标的寄存器 如果 WE 为 0，则数据无法写入
Clk	in	1	时钟信号
Reset	in	1	复位信号 1：复位有效，所有寄存器被置为 0x00000000 0：复位信号无效
busA	out	[31:0]	A 寄存器读出的数据
busB	out	[31:0]	B 寄存器读出的数据

3. CMP

信号名	方向	位数	功能描述
CMPA	in	[15:0]	比较数字 A
CMPB	out	[31:0]	比较数字 B
CMPOp	in	[1:0]	0 为 beq 相等比较
CMPOut	out	1	输出 1 则为需要跳转

4. EXT

信号名	方向	位数	功能描述
Imm16	in	[15:0]	16 位立即数
Imm32	out	[31:0]	32 位拓展结果
ExtOp	in	1	控制立即数做拓展方式 0：0 拓展 1：符号拓展

5. D 级流水寄存器

```
REG #(32) REG_IR_D(F_IR,D_IR,clk,reset,D_en);
REG #(32) REG_PC_D(F_PC,D_PC,clk,reset,D_en);
REG #(32) REG_PC4_D(F_PC4,D_PC4,clk,reset,D_en);
REG #(32) REG_PC8_D(F_PC8,D_PC8,clk,reset,D_en);
```

(三) E 级部件

进行运算操作。

1. ALU

信号名	方向	位数	功能描述
A	in	[31:0]	读入的运算数 A
B	in	[31:0]	读入的运算数 B
Result	out	[31:0]	运算结果
ALUCtr	in	[3:0]	运算功能选择信号

ALUCtr 信号功能定义

功能编号	功能代号	功能	功能描述
4'b0000	ALU_addu	无符号加	result = A + B
4'b0001	ALU_subu	无符号减	result = A - B
4'b0010	ALU_and	按位与	result = A & B
4'b0011	ALU_or	按位或	result = A | B
4'b0100	ALU_sll16	左移 16 位	result = B<<16
4'b0101	ALU_jal	输出 B	result = B

2. E 级流水寄存器

```
REG #(32) REG_IR_E(D_IR,E_IR,clk,E_clr|reset,en);
REG #(32) REG_PC_E(D_PC,E_PC,clk,E_clr|reset,en);
REG #(32) REG_PC8_E(D_PC8,E_PC8,clk,E_clr|reset,en);
REG #(32) REG_RS_E(D_RS,E_RS,clk,E_clr|reset,en);
REG #(32) REG_RT_E(D_RT,E_RT,clk,E_clr|reset,en);
REG #(32) REG_EXT_E(D_EXTOUT,E_EXTOUT,clk,E_clr|reset,en);
```

(四) M 级部件

1. DM

信号名	方向	位数	功能描述
DataIn	in	[31:0]	写入的数据
DataOut	out	[31:0]	读出的数据
Addr	in	[31:0]	数据的写入地址
WrEn	in	1	是否写入数据到 Addr
Reset	in	1	复位
Clk	in	[3:0]	时钟信号

2. M 级流水寄存器

```
REG #(32) REG_IR_M(E_IR,M_IR,clk,reset,en);
REG #(32) REG_PC_M(E_PC,M_PC,clk,reset,en);
```

REG #(32) REG_ALUOUT_M(E_ALUOUT,M_ALUOUT,clk,reset,en);

REG #(32) REG_RT_M(E_ALUB0,M_RT,clk,reset,en);

(五) W 级部件

1. W 级流水寄存器

REG #(32) REG_IR_W(M_IR,W_IR,clk,reset,en);

REG #(32) REG_PC_W(M_PC,W_PC,clk,reset,en);

REG #(32) REG_REGW_W(M_REGW,W_REGW,clk,reset,en);

四、 控制器设计

(一) Controller

1. 基本描述

Controller 主要功能是通过指令的 opcode 和 func 字段来判断指令的类型，从而决定各个部件的控制信号。

2. 端口定义

信号名	方向	位数	功能描述
IR	in	[31:0]	输入指令
ALU_OP	out	[3:0]	决定 ALU 的运算操作 4’b0000 加 4’b0001 减 4’b0010 按位与 4’b0011 按位或 4’b0100 左移 16 位 4’b0101 输出操作数 B
GRF_RW	out	[1:0]	2’b00：写入到 rd，即写入编号是指令中 11-15 位的寄存器(addu,subu) 2’b01：写入到 rt：写入编号为指令中 16-20 位的寄存器(ori,lui) 2’b10：写入到 31 号寄存器(用于 jal 指令)
GRF_WD	out	1	0：寄存器写使能端口无效 1：寄存器写使能端口有效
ALU_SRC	out	[1:0]	0：ALU B 口读入的是 GRF 的 busB 1：ALU B 口读入的是立即数 2：ALUB 口读入的是 pc8
GRF_WD	out	[1:0]	2’b00 ： 从 ALU 运 算 结 果 读 入 到 寄 存 器

			(addu,subu,ori,lui,jal) 2’b01：从 DM 中读入值到寄存器(lw)
MEM_WR	out	1	0：DM 写使能无效 1：DM 写使能有效
NPC_OP	out	[1:0]	选择下一个 PC 地址信号 2’b00:PC+4 2’b01:br 分支指令 2’b10:j 指令 2’b11:jr 指令
EXT_OP	out	1	0：进行 0 拓展 1：进行有符号拓展
IFU_OP	out	1	0：选择 add4 的结果 1：选择 npc 的结果
CMP_OP	out	[1:0]	1：beq 指令

3. 指令分析

见 excel 表格

4. 指令与控制信号真值表

见 excel 表格

(二) AT_Code

1. 基本描述

AT_Code 主要功能是通过指令来判断指令的类型，从而决定指令要写入的寄存器编号和计算每条指令的 Tuse 和 Tnew。

2. 端口定义

信号名	方向	位数	功能描述
IR	in	[31:0]	输入指令
Tuse_RS0,Tuse_RS1 Tuse_RT0,Tuse_RT1 Tuse_RT2,	out	1	解码产生各个指令的 Tuse 值
Tnew_D	out	[1:0]	解码产生各个指令的 Tnew 值
A3	out	[4:0]	解码产生各个指令要写寄存器的编号

3. 指令与控制信号真值表

见 excel 表格

(三) STALL_CTR

1. 基本描述

暂停控制器，生成 pc 使能信号，d 级使能和 e 级清空信号

2. 端口定义

信号名	方向	位数	功能描述
stall	in	1	暂停控制信号
PC_en	out	1	pc 使能信号=!stall
E_clr	out	1	E 级清空信号=stall
D_en	out	1	D 级使能信号=!stall

(四) STALL_SEL

1. 基本描述

主要功能是判断指令是否冲突，计算是否暂停。

2. 端口定义

信号名	方向	位数	功能描述
IR	in	[31:0]	输入指令
Tuse_RS0,Tuse_RS1 Tuse_RT0,Tuse_RT1 Tuse_RT2,	in	1	指令的 Tuse 值
Tnew_E,Tnew_M	in	[1:0]	E 级和 M 级指令的 Tnew 值
A3_E, A3_M	in	[4:0]	E 级和 M 级要写寄存器的编号
W_E,W_M	in	1	W 级和 M 级的写寄存器使能信号
stall	out	1	暂停信号

3. 控制信号产生

```
    wire Stall_RS0_E = Tuse_RS0 & (Tnew_E > 0) & (`A1 == A3_E) & W_E & (A3_E != 0);
    wire Stall_RS0_M = Tuse_RS0 & (Tnew_M > 0) & (`A1 == A3_M) & W_M & (A3_M != 0);
    wire Stall_RS1_E = Tuse_RS1 & (Tnew_E > 1) & (`A1 == A3_E) & W_E & (A3_E != 0);
    wire Stall_RS = Stall_RS0_E | Stall_RS0_M |Stall_RS1_E;
```

```
    wire Stall_RT0_E = Tuse_RT0 & (Tnew_E > 0) & (`A2 == A3_E) & W_E & (A3_E !=
0);
    wire Stall_RT1_E = Tuse_RT1 & (Tnew_E > 1) & (`A2 == A3_E) & W_E & (A3_E !=
0);
    wire Stall_RT0_M = Tuse_RT0 & (Tnew_M > 0) & (`A2 == A3_M) & W_M & (A3_M !=
0);
    wire Stall_RT = Stall_RT0_E|Stall_RT1_E|Stall_RT0_M;

        assign stall = Stall_RS|Stall_RT;
```

(五) 控制寄存器

```
//Tnew
//从 d 级传入的 Tnew
    T_REG E_TNEW(Tnew_D,Tnew_E,clk,(reset|E_clr),en);
    T_REG M_TNEW(Tnew_E,Tnew_M,clk,reset,en);
    T_REG W_TNEW(Tnew_M,Tnew_W,clk,reset,en);

//A3
     REG #(5)A3_E(D_A3,E_A3,clk,(reset|E_clr),en);
     REG #(5)A3_M(E_A3,M_A3,clk,reset,en);
     REG #(5)A3_W(M_A3,W_A3,clk,reset,en);

//IR
     REG #(32)IR_E(D_IR,E_IR,clk,(reset|E_clr),en);
     REG #(32)IR_M(E_IR,M_IR,clk,reset,en);
    REG #(32)IR_W(M_IR,W_IR,clk,reset,en);
```

五、 测试程序

(一) MIPS 代码

.text

####初始化

lui $1 0x3344

ori $1 $1 0x8788

lui $2 0x6565

ori $2 $2 0x1298

lui $3 0x3789

ori $3 $3 0x4444

```
lui $4 0x0233
ori $4 $4 0xffac
lui $5 0xffff
ori $5 $5 0xabbb
```

####计算型指令 E 级使用数据的指令

```
subu $4 $2 $3
addu $3 $4 $1
subu $1 $2 $3
jal a1
nop
ori $31 $0 0
a1:
addu $4 $3 $31
addu $3 $31 $4
ori $4 $5 0x1122
addu $4 $3 $2
lui $3 0xffff
addu $4 $2 $3
```

######计算型指令 nop E 级使用数据的指令

```
subu $4 $2 $3
nop
addu $3 $4 $1
nop
subu $1 $2 $3
jal a2
nop
ori $31 $0 0
a2:
```

```
nop
addu $4 $3 $31
addu $3 $31 $4
ori $4 $5 0x1122
nop
addu $4 $3 $2
lui $3 0xffff
nop
addu $4 $2 $3

#计算型指令　D 级使用数据的指令
subu $4 $3 $0
beq $4 $3 a3
nop
lui $3 0x8888
a3:
lui $3 0x9999

addu $4 $3 $0
beq $3 $4 a4
nop
lui $4 0x7777
a4:
lui $3 0x6666

jal a5
nop
j a6
nop
a5:
```

```
jr $31
nop
a6:
ori $2 $31 0xaccc
ori $3 $2 0
beq $3 $2 a7
nop
lui $4 0xffff
a7:
lui $4 0x8888
beq $4 $3 a8
nop
lui $9 0x8989
a8:
lui $9 0x1234


#计算型指令  nop D 级使用数据的指令
subu $4 $3 $0
nop
beq $4 $3 b1
nop
lui $3 0x8888
b1:
lui $3 0x9999

addu $4 $3 $0
nop
beq $3 $4 b2
nop
```

```
lui $4 0x7777
b2:
lui $3 0x6666

jal b3
nop
j b4
nop
b3:
addu $3 $31 $31
nop
jr $31
nop
b4:
ori $2 $31 0xaccc
ori $3 $2 0
nop
beq $3 $2 b5
nop
lui $4 0xffff
b5:
lui $4 0x8888
nop
beq $4 $3 b6
nop
lui $9 0x8989
b6:
lui $9 0x1234
```

#计算型指令 nop nop D 级使用数据的指令

```
subu $4 $3 $0
nop
nop
beq $4 $3 c1
nop
lui $3 0x8888
c1:
lui $3 0x9999

addu $4 $3 $0
nop
nop
beq $3 $4 c2
nop
lui $4 0x7777
c2:
lui $3 0x6666

jal c3
nop
nop
j c4
nop
c3:
addu $3 $31 $31
nop
nop
jr $31
nop
```

```
c4:
ori $2 $31 0xaccc
ori $3 $2 0
nop
nop
beq $3 $2 c5
nop
lui $4 0xffff
c5:
lui $4 0x8888
nop
nop
beq $4 $3 c6
nop
lui $9 0x8989
c6:
lui $9 0x1234

#"计算型指令  M 级使用数据"
ori $2 $0 24
ori $3 $0 12
subu $4 $2 $3
lw $3    -8($4)

ori $2 $0 24
ori $3 $0 12
subu $4 $2 $3
sw $4    -8($3)

jal d1
```

```
nop
ori $3 $0 1234
j d2
d1:
sw $31 0($0)
jr $31
nop
d2:

ori $3 $0 32
sw $4 8($3)

#"计算型指令  nop M 级使用数据"
ori $2 $0 24
ori $3 $0 12
subu $4 $2 $3
nop
lw $3    -8($4)

ori $2 $0 24
ori $3 $0 12
subu $4 $2 $3
nop
sw $4    -8($3)

jal e1
nop
ori $3 $0 1234
j e2
e1:
```

```
nop
sw $31 0($0)
jr $31
nop
e2:
ori $3 $0 32
nop
sw $4 8($3)

#load 类指令  D 级使用数据
sw $4 0($0)
lw $3 0($0)
beq $3 $4 f1
nop
lui $3 1000
f1:
lui $3 989

#load 类指令  nop D 级使用数据
sw $4 0($0)
lw $3 0($0)
nop
beq $3 $4 f2
nop
lui $3 1000
f2:
lui $3 989
```

```
#load 类指令  E 级使用数据
sw $4 0($0)
lw $3 0($0)
addu $2 $3 $3
subu $3 $2 $2

ori $4 $0 32
sw $4 0($0)
lw $3 0($0)
lw $4 -32($3)

#load 类指令  nop E 级使用数据
sw $4 0($0)
lw $3 0($0)
nop
addu $2 $3 $3
subu $3 $2 $2

ori $4 $0 32
sw $4 0($0)
lw $3 0($0)
nop
lw $4 -32($3)

#load 类指令  M 级使用数据的指令
ori $4 $0 12
sw $4 0($0)
lw $3 0($0)
lw $4 8($3)
```

```
ori $4 $0 24
sw $4 0($0)
lw $3 0($0)
sw $3 8($3)


#load 类指令  nop M 级使用数据的指令
ori $4 $0 16
sw $4 0($0)
lw $3 0($0)
nop
lw $4 8($3)
ori $4 $0 8
sw $4 0($0)
lw $3 0($0)
nop
sw $3 8($3)
```

六、 冲突测试设计

冲突是由现在要用的寄存器的值还没有写入寄存器堆产生的，所以，只有写寄存器的指令在前，用寄存器的指令在后才会出现冲突。在用寄存器的值的时候，判断当前上一条指令有没有产生寄存器的值，如果产生就转发，如果没产生，就暂停。

写入寄存器的指令有：addu, subu, ori, lui, lw, jal

要读寄存器的指令有：addu, subu, ori, lui, lw, sw, beq, jr

根据各个指令 Tnew 和 Tuse 的情况来设计冲突测试样例。

		R 型		**J 型**		**I 型**	
冲突	处理方式	RS 冲突样例	RT 冲突样例	RS 冲突样例	RT 冲突样例	RS 冲突样例	RT 冲突样例

计算型指令 **E 级使用数据的指令**	M 级转发至 E 级 ALU	subu $4 $2 $3 addu $3 $4 $1	addu $4 $2 $3 subu $1 $1 $4	jalr $4 $3 addu $4 $4 $3	jal label addu $4 $4 $31	ori $3 $5 0xff12 addu $4 $3 $2	lui $3 0xff12 addu $4 $2 $3
计算型指令 **nop** **E 级使用数据的指令**	W 转发至 D 级	addu $4 $2 $3 nop subu $3 $4 $1	subu $4 $2 $3 nop addu $3 $1 $4	jal label nop addu $4 $31 $3	jalr $4 $3 nop addu $4 $3 $4	ori $3 $5 0xff12 nop addu $4 $3 $3	lui $3 0xff12 nop addu $4 $2 $4
计算型指令 **D 级使用数据的指令**	暂停一周期 M 级转发到 D 级	addu $4 $2 $3 beq $4 $3 label	addu $4 $2 $3 beq $4 $4 label	jal label nop jr $31	jalr $4 $3 nop jr $4	ori $3 $0 0x1111 ori $4 $0 0x1111 beq $4 $3 label	ori $3 $5 0xff12 ori $4 $3 0x9977 beq $3 $4 label
计算型指令 **nop** **D 级使用数据的指令**	M 级转发到 D 级	subu $4 $2 $3 nop beq $4 $3 label	addu $4 $2 $3 nop beq $4 $4 label	jal $5 label nop jr $31	jalr $4 $3 nop jr $4	ori $3 $0 0x1111 ori $4 $0 0x1111 nop beq $4 $3 label	ori $3 $5 0xff12 ori $4 $3 0x9977 nop beq $3 $4 label
计算型指令 **nop** **nop** **D 级使用数据的指令**	W 级转发到 D 级	subu $4 $2 $3 nop nop beq $4 $3 label	addu $4 $2 $3 nop nop beq $4 $4 label	jal $5 label nop nop jr $31	jalr $4 $3 nop nop jr $4	ori $3 $0 0x1111 ori $4 $0 0x1111 nop nop beq $4 $3 label	ori $3 $5 0xff12 ori $4 $3 0x9977 nop nop beq $3 $4 label
计算型指令 **M 级使用数据**	M 级转发至 E 级	subu $4 $2 $3 lw $3	subu $4 $2 $3 sw $4	jalr $4 $3 lw $3	jalr $4 $3 sw $4	ori $3 $5 0xff12 sw $4	lui $3 0xff12 lw $3

		-8($4)	-8($3)	8($4)	8($3)	8($3)	8($3)
计算型指令 nop M 级使用数据	M 级转发至 D 级	subu $4 $2 $3 nop lw $3 -8($4)	subu $4 $2 $3 nop sw $4 -8($4)	jalr $4 $3 nop lw $3 8($4)	jalr $4 $3 nop sw $4 8($4)	ori $3 $5 0xff12 nop sw $4 8($3)	lui $3 0xff12 nop lw $3 8($3)
load 类指令 D 级使用数据	暂停两周期 W 级转发至 D 级	sw $4 0($0) lw $3 0($0) beq $3 $4 label	sw $4 0($0) lw $3 0($0) beq $4 $3 label	sw $4 0($0) lw $31 0($0) jr $31			
load 类指令 nop D 级使用数据	暂停一周期 W 级转发至 E 级	sw $4 0($0) lw $3 0($0) nop beq $3 $4 label	sw $4 0($0) lw $3 0($0) nop beq $4 $4 label	sw $4 0($0) lw $31 0($0) nop jr $32			
load 类指令 E 级使用数据	暂停一周期 W 级转发到 E 级	sw $4 0($0) lw $3 0($0) addu $2 $3 $3	sw $4 0($0) lw $3 0($0) subu $2 $4 $3			sw $4 0($0) lw $3 0($0) lw $4 8($3)	sw $4 0($0) lw $3 0($0) sw $3 -8($3)
load 类指令 nop E 级使用数据	W 级转发至 D 级	sw $4 0($0) lw $3 0($0) nop subu $2 $3 $3	sw $4 0($0) lw $3 0($0) nop addu $2 $4 $3			sw $4 0($0) lw $3 0($0) nop sw $4 8($3)	sw $4 0($0) lw $3 0($0) nop sw $3 8($3)
load 类指令 M 级使用数据的指令	W 级转发至 M 级					sw $4 0($0) lw $3 0($0) lw $4 8($3)	sw $4 0($0) lw $3 0($0) sw $3 8($3)
load 类指令 nop M 级使用数据的指令	W 级转发至 E 级					sw $4 0($0) lw $3 0($0) nop	sw $4 0($0) lw $3 0($0) nop

						lw $4 8($4)	sw $3 8($4)