计算机组成 CPU 设计文档-P6

整体结构与概览

1、CPU 基本参数与指标

处理器类型：流水线 CPU

处理器字长：32 位

处理器支持指令集：

calr(24)	add	addu	sub	subu		
	sll	srl	sra	sllv	srlv	srav
	and	or	xor	nor		
	slt	sltu				
	mult	multu	div	divu		
	mthi	mtlo	mfhi	mflo		
cali(8)	addi	addiu				
	andi	ori	xori			
	lui					
	slti	stliu				
ld(5)	lw					
	lb	lbu				
	lh	lhu				
st(3)	sw	sh	sb			
branch(6)	beq	bne	blez	bgtz	bltz	bgez
j	j					
jal	jal					
jr	jr					
jalr	jalr					

顶层封装模块端口：

表格 1 顶层封装模块端口

端口名	类型	描述
reset	In	接受同步复位信号，对 PC、GRF、Mem 和 PipeReg 复位。
clk	In	接受时钟驱动信号，驱动 PC、GRF/W、IM/W、PipeReg

2、CPU 模块结构

数据通路

1. IFU（取指令单元）:包括 PC 和存放指令的 ROM，用于输出当前指令码。
2. GRF（通用寄存器组）：内含 32 个寄存器，支持对寄存器值的读写。
3. ALU（算术逻辑单元）：运算执行部件，对 32 位数执行多种运算。
4. XALU（乘除法单元）：乘除法特异性执行部件，内部含有 Hi 和 Lo 寄存器，对两个 32 位数执行乘除法，并且具有较长的时间延迟。
5. DM（数据存储器）：存储数据部件，支持读写。
6. BED（字节使能译码器）：根据主控信号，输出 DM 的字写入使能。
7. MDS（主存数据选择器）：根据主控信号，将 DM 输出的字进行处理。
8. EXT（位扩展器）：将 16 位数扩展为 32 位数，支持有/无符号扩展。
9. NPC（外部跳转分支计算器）：支持跳转计算和分支计算与判断，若为分支指令，会根据 CMP 结果选择正确的 pc 值。
10. CMP（分支比较器）：根据指令条件设置，返回判断结果。
11. PipeReg（流水线寄存器层）：实现 CPU 流水并行效果，上升沿时接受前一层运行完且需要传递至下一级的数据，其他时刻释放本层功能所需的数据。一共分为 FD，DE，EM，MW 四层。
12. FuncMux（功能多选器）:对同一端口多个数据源进行筛选，目前有 AluSrc、AluSel、WaSel、WdSel 四个。
13. TMux（转发多选器）：用于转发解决冲突时所使用的多选器，目前有 GRF_RD1, GRF_RD2, DE_RD1, DE_RD2, EM_RD2。

控制信号

1、主控器：识别指令并生成 CPU 各部分的控制信号，使用逻辑阵列实现。

2、冲突控制器：

a) GID :通用指令译码器 :根据所给指令和流水线段, 返回 Tuse、Tnew、写地址、数据管道等参数。

b) STALL：暂停控制器，输出是否暂停的指令。

c) TRANSMIT：转发控制器，输出 5 个转发多选器的控制信号。

一、 模块规格（数据通路）

1、数据通路

表格 2 数据通路端口合成-无转发

部件	端口名称（入）	汇总端口	多路选择器	控制信号	0	1	2
IFU	PC						
	ADD4(inside)	PC					
	IM(inside)	PC					
	PC(inside)	ADD4					
FD_IR		IM					
FD_PC4		ADD4					
FD_PC(display)		PC					
GRF	RA1	FD_IR[rs]					
	RA2	FD_IR[rt]					
EXT		FD_IR[Im16]					
CMP	A	RD1					
	B	RD2					
NPC	IM16	FD_IR[IM16]					
	IM26	FD_IR[Im26]					
	RegPc	RD1					
	Cmp	CMP					
	PC4	FD_PC4					
IFU	Pc_Update	NPC					
DE_IR		FD_IR					
DE_PC4		FD_PC4					
DE_RD1		RD1					
DE_RD2		RD2					
DE_EXT		EXT					
DE_PC(display)		FD_PC					
ALU	A	DE_RD1					
	B	MUX_AluSrc	MUX_AluSrc	AluSrc	DE_RD2	DE_EXT	DE_PC4
	C	DE_IR[s]					
XALU	A	DE_RD1					
	B	DE_RD2					
EM_IR		DE_IR					
EM_ALU		MUX_AluSel	MUX_AluSel	AluSel	Alu	XAlu-Hi	XAlu-Lo
EM_RD2		DE_RD2					
EM_PC(display)		DE_PC					
DM	Addr	EM_ALU					
	WD	EM_RD2					
MW_IR		EM_IR					
MW_ALU		EM_ALU					
MW_MD		DM					
MW_PC(display)		EM_PC					
GRF	WA	MUX_WaSel	MUX_WaSel	WaSel	MW_IR[rt]	MW_IR[rd]	31
	WD	MUX_WdSel	MUX_WdSel	WdSel	MW_ALU	MW_MD	
	WPC(display)	MW_PC					

表格 3 数据通路端口合成-转发

部件	端口名称（入）	汇总端口	多路选择器	控制信号	0	1	2
IFU	PC						
	ADD4(inside)	PC					
	IM(inside)	PC					
	PC(inside)	ADD4					
FD_IR		IM					
FD_PC4		ADD4					
FD_PC(display)		PC					
GRF	RA1	FD_IR[rs]					
	RA2	FD_IR[rt]					
EXT		FD_IR[Im16]					
CMP	A	TMUX_GRF_RD1					
	B	TMUX_GRF_RD2					
NPC	IM16	FD_IR[IM16]					
	IM26	FD_IR[Im26]					
	RegPc	TMUX_GRF_RD1					
	Cmp	CMP					
	PC4	FD_PC4					
IFU	Pc_Update	NPC					
DE_IR		FD_IR					
DE_PC4		FD_PC4					
DE_RD1		TMUX_GRF_RD1					
DE_RD2		TMUX_GRF_RD2					
DE_EXT		EXT					
DE_PC(display)		FD_PC					
ALU	A	TMUX_DE_RD1					
	B	MUX_AluSrc	MUX_AluSrc	AluSrc	TMUX_DE_RD2	DE_EXT	DE_PC4
	C	DE_IR[s]					
XALU	A	TMUX_DE_RD1					
	B	TMUX_DE_RD2					
EM_IR		DE_IR					
EM_ALU		MUX_AluSel	MUX_AluSel	AluSel	Alu	XAlu-Hi	XAlu-Lo
EM_RD2		DE_RD2					
EM_PC(display)		DE_PC					
DM	Addr	EM_ALU					
	WD	TMUX_EM_RD2					
MW_IR		EM_IR					
MW_ALU		EM_ALU					
MW_MD		DM					
MW_PC(display)		EM_PC					
GRF	WA	MUX_WaSel	MUX_WaSel	WaSel	MW_IR[rt]	MW_IR[rd]	31
	WD	MUX_WdSel	MUX_WdSel	WdSel	MW_ALU	MW_MD	
	WPC(display)	MW_PC					

表格 4 转发多选器

Tmux	0	1	2	3
TMUX_GRF_RD1	RD1	EM_ALU	MW_ALU	MW_MD
TMUX_GRF_RD2	RD2	EM_ALU	MW_ALU	MW_MD
TMUX_DE_RD1	DE_RD1	EM_ALU	MW_ALU	MW_MD
TMUX_DE_RD2	DE_RD2	EM_ALU	MW_ALU	MW_MD
TMUX_EM_RD2	EM_RD2	X	MW_ALU	MW_MD

2、IFU（取指令单元）

a) 端口

表格 5 IFU 端口表

端口名称	类型	功能描述
Clock	In	控制信号，接受时钟信号
Reset	In	控制信号，接受 Pc 同步复位信号
Branch_Jump	In	控制信号，是否接受 NPC 输出作为 PC 新值
Pc_Update[31:0]	In	数据通路，分支/跳转指令中接受 PC 更新值
PC4[31:0]	Out	数据通路，输出 PC+4（32 位 **Byte 编址**）
Instr[31:0]	Out	数据通路，输出 32 位指令二进制码
PC[31:0]	Out	调试信号，输出 PC（32 位 **Byte 编址**）

b) 功能描述

- IFU 主要由 PC 和存放指令的 ROM 组成，用于取出指令和 PC 更新。
- ROM 规格为 4096*32bits，**字编址（访问时需地址转换）**。
- PC 为 32 位二进制，起始地址为 0x00003000，**字节编址**，支持向字编址转换（除 4）。

c) 注意事项

- ROM 和 RAM 部件的地址端口为字编址地址。为保证兼容性，该设计在顶层设计时用字节编址，而在次层具体部件设计时会进行字编址转换。

3、GRF（通用寄存器组）

a) 端口

表格 6 GRF **端口表**

端口名称	类型	功能描述
Clock	In	控制信号，接受时钟信号

Reset	In	控制信号，接受同步复位信号
RegWrite	In	控制信号，接受寄存器写使能信号
ReadAddr1/R[4:0]	In	数据通路，读，接受 Rs 寄存器地址
ReadAddr2/R[4:0]	In	数据通路，读，接受 Rt 寄存器地址
WriteAddr/W[4:0]	In	数据通路，写，接受被写入寄存器地址
WriteData/W[31:0]	In	数据通路，写，接受被写入数据
RegData1/R[31:0]	Out	数据通路，读，输出 Rs 寄存器值
RegData2/R[31:0]	Out	数据通路，读，输出 Rt 寄存器值
WPC[31:0]	None	调试信号，用于寄存器写时 display

b) 功能描述

- GRF 中共有 32 个寄存器，对应 MARS 中的 32 个通用寄存器。（注意：不包括 hi，lo，pc 寄存器。）
- 读：GRF 读功能时作为组合逻辑电路，根据输入地址信号，输出数据。
- 写：GRF 写功能作为时序逻辑电路，相应的 Addr, Data, RegWrite 应在时钟上升沿前做好准备。

c) 备注

- 此版本 GRF 读写功能的地址和数据端口独立，可实现同步读写操作。
- 0 号寄存器恒为 0 值，不可被改写。

4、ALU（逻辑运算单元）

a) 端口

表格 7 ALU 端口表

端口名称	类型	功能描述
AluOp[3:0]	In	控制信号，接受算术逻辑信号
A[31:0]	In	数据通路，接受算术逻辑操作数 A

B[31:0]	In	数据通路，接受算术逻辑操作数 B
C[4:0]	In	数据通路，接受算术逻辑操作数 C（常于移位）
Result[31:0]	Out	数据通路，输出结果

b) 功能描述

- ALU 受 AluController 的控制信号控制，输出不同的算术逻辑结果：

表格 8 ALU 功能表

AluOp	功能	适用指令
0	A + B	add addu addi addiu ld st
1	A - B	sub subu
2	B << s	sll
3	B >> s	srl
4	Signed B >> s	sra
5	B << A[4:0]	sllv
6	B >> A[4:0]	srlv
7	Signed B >> A[4:0]	srav
8	A and B	and andi
9	A or B	or ori
10	A xor B	xor xori
11	A nor B	nor
12	set 1 if A < B Signed	slt slti
13	set 1 if A < B Unsiged	sltu sltiu
14	B << 16	lui
15	B+4	jal jalr
other	0	other

* 黄色高亮符号代表此指令是一类指令。

5、XALU（乘除法运算器）

a) 端口 lock,Reset,Start,XALUOp,Busy,RD1,RD2,HI,LO

表格 9 XALU 端口表

端口名称	类型	功能描述
Clock	In	控制信号，接受时钟信号
Reset	In	控制信号，接受同步复位信号
Start[1:0]	In	控制信号，接受运算赋值启动信号 （1-乘除运算，2-mthi、mtlo 赋值）
XALUOp[2:0]	In	控制信号，接受 XALU 操作信号
RD1[31:0]	In	数据通路，接受操作数 A
RD2[31:0]	In	数据通路，接受操作数 B
Busy	Out	数据通路，输出 XALU 乘除运算繁忙信号
Hi[31:0]	Out	数据通路，输出 HI 寄存器值
Lo[31:0]	Out	输出通路，输出 LO 寄存器值

b) 功能描述

表格 10 XALU 功能表

XAluOp	功能	适用指令
0	signed(A * B)	mult
1	unsigned(A * B)	multu
2	signed(A / B)	div
3	unsigned(A / B)	divu
4	move RD1 to Hi	mthi
5	move RD1 to Lo	mtlo

c) 模拟规定

1. 自 Start 信号有效后的第 1 个 clock 上升沿开始，乘除部件开始执行运算，同时 Busy 置位为 1。
2. 在运算结果保存到 HI 和 LO 后，Busy 位清除为 0。

3. 当 Busy 为 1 时，mfhi、mflo、mthi、mtlo、mult、multu、div、divu 均被阻塞，即被阻塞在 IF/ID。

4. 数据写入 HI 或 LO，均只需 1 个 cycle。

6、DM（数据存储器）

a) 端口

表格 11 数据存储器端口表

端口名称	类型	功能描述
Clock	In	控制信号，接受时钟信号
Reset	In	控制信号，接受同步复位信号
MemWrite	In	控制信号，接受写使能信号
MemRead	In	控制信号，接受读使能信号
Load_Type[1:0]	In	控制信号，接受读位宽控制信号 （0-字，1-半字，2-字节）
Store_type	In	控制信号，接受写位宽控制信号 （0-字，1-半字，2-字节）
SigedRead	In	控制信号，读出内容符号拓展（1-拓展，0-不拓展）
MemAddr[31:0]	In	数据通路，接受**读/写**操作地址，**byte 编址**
WriteData[31:0]	In	数据通路，写，写入数据
ReadData[31:0]	Out	数据通路，读，输出数据
WPC[31:0]	None	调试信号，用于主存写时 display

b) 描述

- 数据存储器使用 RAM 实现，容量为 4096*32bits，RAM **字节编址**。
- 读/写共用一个地址端口，同一时钟周期只能进行读/写的其中之一。
- 起始地址：0x00000000。

7、BED（字节使能译码器）

表格 12 BED **端口表**

端口名称	类型	功能描述
StoreType[1:0]	In	控制信号，接受存储位（字-半字-字节）
Addr[1:0]	In	数据通路，地址后两位。
ByteEnable[3:0]	Out	数据通路, 输出 Addr[31:2]对应字允许写入的字节位置，以独热码形式输出。 ByteEnable[3] = 1 : [31:24] WriteEnable ByteEnable[2] = 1 : [23:16] WriteEnable ByteEnable[1] = 1 : [15:8] WriteEnable ByteEnable[0] = 1 : [7:0] WriteEnable

8、MDS（主存数据选择器）

表格 13 MDS **端口表**

端口名称	类型	功能描述
LoadType[1:0]	In	控制信号，接受输出位宽（字-半字-字节）
SignRead	In	控制信号，结果扩展（1-符号，0-无符号）
Addr[1:0]	In	数据通路，**字节编址**地址后两位
Word[31:0]	In	数据通路，按字读取的数据
RD[31:0]	Out	数据通路，指定位宽、拓展、地址的输出数据

9、Ext（位数扩展器）

表格 14 16-32 **位扩展器端口表**

端口名称	类型	功能描述
ExtOp	In	控制信号，控制扩展方式（0-zero，1-sign）
In[15:0]	In	数据通路，接收待扩展的 16 位数字。
Out[31:0]	Out	数据通路，输出扩展后的 32 位数字。

10、 NPC（分支跳转指令地址计算器）

表格 15 NPC **端口表**

端口名称	类型	功能描述
nPc_Sel[2:0]	In	控制信号，控制地址计算方式 （0-branch，1-j/jal，2-jr）
Cmp	In	数据通路，接受 CMP 的分支决策信号。(beq)
Im32[31:0]	In	数据通路，接受 EXT 的**符号扩展**立即数。(beq)
Im26[25:0]	In	数据通路，接受指令中的 26 位立即数。(j/jal)
Pc4[31:0]	In	数据通路，接受当前指令 Pc+4。(j/jal/beq)
RegPc[31:0]	In	数据通路，接受从寄存器中读取的跳转值。(jr)
Pc_Update[31:0]	Out	数据通路，输出 PC 分支跳转计算的地址。

11、 CMP（分支条件判断）

表格 16 CMP **端口表**

端口名称	类型	功能描述
CmpOp	In	控制信号，指定 branch 指令比较策略（0-equal）
A[31:0]	In	数据通路，比较数 A
B[31:0]	In	数据通路，比较数 B
Cmp	Out	比较结果（1-成立，0-不成立）

12、 PipeReg（流水线寄存器）

表格 17 PipeReg **端口表**

流水线级别	端口	功能描述
FD 段	F_IR → FD_IR	D 段指令
	F_Pc4 → FD_Pc4	D 段 PC+4
	F_Pc → FD_Pc	D 段 PC（display）

DE 段	D_IR → DE_IR	E 段指令
	D_Pc4 → DE_Pc4	E 段 Pc+4
	D_RD1 → DE_RD1	E 段 GRF[RS]
	D_RD2 → DE_RD2	E 段 GRF[RT]
	D_EXT → DE_EXT	E 段扩展立即数
	D_Pc → DE_Pc	E 段 PC（display）
EM 段	E_IR → EM_IR	M 段指令
	E_ALU → EM_ALU	M 段 ALU 计算结果
	E_RD2 → EM_RD2	M 段 GRF[RT]
	E_Pc → EM_Pc	M 段 PC（display）
MW 段	M_IR → MW_IR	W 段指令
	M_ALU → MW_ALU	W 段 ALU 计算结果
	M_MD → MW_MD	W 段 Memory 读取结果
	M_Pc → MW_Pc	W 段 Pc（display）

二、 模块规格（控制电路）

与单周期 CPU 不同的是，流水线 CPU 在运行中存在结构冒险、数据冒险和控制冒险三种冒险问题。结构冒险利用 GRF 和指令数据 DM 分离的方式已经解决，控制冒险使用分支跳转提前+延迟槽的方式实现，而数据冒险需要使用暂停和转发逻辑实现。

综上，控制电路分为主控单元 MainController 和冒险控制单元 HazardController。

1、Controller 主控单元

a) 端口

表格 18 主控单元端口功能表

端口名称	类型	功能（所在通路，作用部件，描述）
Op[5:0]	In	数据通路，指令的 Instr[31:26]
Func[5:0]	In	数据通路，指令的 Instr[5:0]
D_Branch_Jump	Out	通用控制，IF，IF 接受外部 Pc 更新信号（1-允许，0-不允许）
D_ExtOp	Out	个性控制，16-32 位扩展类型（1-符号扩展，0-无符号扩展）
D_nPc_Sel[2:0]	Out	通用控制，PC 分支跳转类型 0：Branch 1：Jump/Jal 2：Jr
D_CmpOp	Out	个性控制，Branch 指令比较策略
E_AluSrc[1:0]	Out	通用控制，MUX，ALU-B 端口选择器信号（1-扩展器，0-RD2）
E-AluSel[1:0]	Out	个性控制，MUX，ALU 和 XALU 输出结果选择 0：AluOut 1：XAlu-Hi 2：XAlu-Lo
E_AluOp[3:0]	Out	个性控制，ALU，ALU 驱动信号（具体请参加 ALU 功能表）
E_XAluOp[2:0]	Out	个性控制，XALU，XALU 驱动信号（具体请参见功能表）
M_MemRead	Out	通用控制，Mem，Mem 读使能信号（1-允许，0-不允许:高阻）
M_MemWrite	Out	通用控制，Mem，Mem 写使能信号（1-允许，0-不允许）
M_StoreType[1:0]	Out	个性控制，存储位宽选择（0-字，1-半字，2-字节）
M_LoadType[1:0]	Out	个性控制，加载位宽选择（0-字，1-半字，2-字节）
SignRead	Out	个性控制，加载内容扩展方式（0-无符号，1-符号）
W_WaSel[1:0]	Out	通用控制，MUX，GRF 写地址选择 0：Rt 1：Rs 2：31
W_WdSel[1:0]	Out	通用控制，MUX，GRF 写数据选择 0：Alu 1：Memory
W_RegWrite	Out	通用控制，GRF，GRF 写使能信号 （1–允许，0-不允许）

*通用控制：该控制信号能够直接在“指令类型”层面进行定义。

**个性控制：该控制信号需要在“具体某条指令”层面进行定义（常见于同类指令中不同功能）。

b) 信号真值表

i. 通用控制信号：

表格 19 主控单元通用控制信号真值表

	calr	cali	branch	ld	st	j	jal	jr	jalr
Branch_Jump	0	0	1	0	0	1	1	1	1
nPc_Sel	x(0)	x(0)	0	x(0)	x(0)	1	1	2	2
AluSrc	0	1	x(0)	1	1	x(0)	2	x(0)	2
MemRead	0	0	0	1	0	0	0	0	0
MemWrite	0	0	0	0	1	0	0	0	0
RegWrite	1	1	0	1	0	0	1	0	1
WaSel	1	0	x(0)	0	x(0)	x(0)	2	x(0)	1
WdSel	0	0	x(0)	1	x(0)	x(0)	0	x(0)	0

ii. 个性控制：ExtOp

表格 20 ExtOp

ExtOp	func							
0	unsigned	andi	ori	xori	(Lui)			
1	signed	addi	addiu	slti	slitu	ld	st	branch
x(0)		other						

iii. 个性控制：CmpOp

表格 21 CmpOp

CmpOp	func	
0	equal	beq
1	not equal	bne
2	less or equal	blez
3	greater than	bgtz
4	less than	bltz
5	greater or equal	bgez
x(0)		other

iv. 个性控制：AluOp

表格 22 AluOp

AluOp	function						
0	add	add	addu	addi	addiu	ld	st
1	sub	sub	subu				
2	<< s	sll					
3	>> s	srl					
4	Signed >> s	sra					
5	<< B	sllv					
6	>> B	srlv					
7	Signed >> B	srav					
8	and	and	andi				
9	or	or	ori				
10	xor	xor	xori				
11	nor	nor					
12	set 1 less Signed	slt	slti				
13	set 1 less Unsiged	sltu	sltiu				
14	<< 16	lui					
15	B+4	jal	jalr				
x(0)		other					

v. 个性控制：XAluOp

表格 23 XAluOp

XAluOp	func	
0	signed mul	mult
1	unsiged mul	multu
2	signed div	div
3	unsigned div	divu
4	move to hi	mthi
5	move to lo	mtlo
x(0)		other

vi. 个性控制：AluSel

表格 24 AluSel

AluSel	func	
0	aluout	other
1	HI	mfhi
2	LO	mflo
x(0)		other

vii. 个性控制：Start

表格 25 Start

Start	func				
1	start and busy = 1	mult	multu	div	divu
2	start and busy = 0	mthi	mtlo		
x(0)		other			

viii. 个性控制：Store_Type

表格 26 Store Type

Store_Type	func	
0	word	sw
1	halfword	sh
2	byte	sb
x(0)		other

ix. 个性控制：Load_Type

表格 27 Load Type

Load_Type	func		
0	word	lw	
1	halfword	lh	lhu
2	byte	lb	lbu
x(0)		other	

x. 个性控制：SigneRead

表格 28 SignRead

Sign_Read	func		
1	Sign_Ext	lb	lh
0	Unsigned_Ext	lbu	lhu
x(0)		other	

2、Hazard 冲突控制单元

a) 概览

冲突控制是目前流水线 CPU 与单周期 CPU 差异最大的地方，在实现上也具有一定难度。本 CPU 的冲突控制单元主要解决的是数据冒险问题，（结构冒险和控制冒险已通过功能部件和数据通路构造），通过 “流水线工程化”

[1]方法，主要比较 Tnew、Tuse 等值，即可实现对应的转发和暂停策略。

强抽象：在弱抽象基础上，将指令抽象为 Tnew、Tuse、rwnz 等运行指标。CPU 的冲突控制单元正是建立在这种强抽象上的。

弱抽象：根据指令代码结构和功能划分为 calr、cali、branch 等信号。

b) General Instruction Decoder （通用指令译码器）

流水线工程化方法实质是对各级指令所对应的 Tnew、Tuse、操作寄存器等参数进行比对，形成解决策略。由于每条指令的参数不相同，为模块化功能，设计了 “通用指令译码器” 进行一系列参数的计算。

表格 29 GID **端口表**

端口名称	类型	功能描述
IR[31:0]	In	数据通路，对应流水段指令
Pipe[2:0]	In	数据通路，流水段编号（F-1，D-2，E-3，M-4）
Tuse_Rs[2:0]	Out	数据通路，当前流水段 Rs 的 Tuse no_more_use = 7
Tuse_Rt[2:0]	Out	数据通路，当前流水段 Rt 的 Tuse no_more_use = 7
RegWriteNonZero	Out	数据通路，当前指令是否**向非零寄存器写值**
A3	Out	数据通路，当前指令写入寄存器地址 Rt：lw、cali Rd：calr Jal：$31
Tnew[2:0]	Out	数据通路，当前指令产生写入结果所需时间 no_more_new = 0
Dport[2:0]	Out	数据通路，当前指令写入结果所在寄存器 “管口” 1：EM_ALU 2：MW_ALU 3：MW_MD
HiLo	Out	数据通路，当前指令是否涉及 Hi、Lo 寄存器的操作

[1] L15-流水线工程化方法-2018-V1，高小鹏，北京航空航天大学《计算机组成课程设计》.

表格 30 指令分类与读写功能统计表格

指令分类：	读	写
calr	√	√
cali	√	√
ld	√	√
st	√	
btype	√	
jal		√
j		
jr	√	
jalr	√	√

表格 31 F 段 Tuse

Tuse										
IF/ID										
calr/rs/1	calr/rt/1	cali/rs/1	cali/rt/1	ld/rs/1	st/rs/1	st/rt/2	btype/rs/0	btype/rt/0	jr/rs/0	jalr/rs/0

表格 32 各级 Tnew（仅针对产生写的指令）

Tnew														
ID/EX					EX/MEM					MEM/WB				
calr/rd/1	cali/rt/1	ld/rt 2	jal/$31 1	jalr/rd/1	calr/rd/0	cali/rt/0	ld/rt/1	jal/$31/0	jalr/rd/0	calr/rd/0	cali/rt/0	ld/rt/0	jal/$31/0	jalr/rd/0

c) STALL 暂停控制模块

表格 33 STALL 暂停控制模块端口

端口名称	类型	功能描述
FD_IR[31:0]	In	数据通路，D 段指令
DE_IR[31:0]	In	数据通路，E 段指令
EM_IR[31:0]	In	数据通路，M 段指令
MW_IR[31:0]	In	数据通路，W 段指令
E_Start[1:0]	In	控制信号，E 段 XALU 运算赋值启动信号 （0-不启动，1-乘除运算，2-HiLo 赋值）

E_Busy	In	数据通路，E 段 XALU 乘除占用信号
Stall	Out	数据通路，暂停信号

暂停判断逻辑基于 GID 返回的参数结果，判断式中不含有具体的指令类型，因此具有可延伸性。具体判断逻辑式如下（以 Rs 与第 X 段判断和 XALU 乘除法判断为例）：

$$Stall_{Rs} = (IR_{FD}[Rs] == A3_X)\&\&(RegWriteNonZero_X)\&\&(Tuse_{FD} < Tnew_X)$$

$$Stall_{HiLo} = (HiLo_{FD}\ \&\&\ (Start == 1\ ||\ Busy))$$

$$Stall = Stall_{RS}\ |\ Stall_{Rt}|Stall_{HiLo}$$

d) TRANSMIT 转发控制模块

表格 34 TRANSMIT **端口**

端口名称	类型	功能描述
FD_IR[31:0]	In	数据通路，D 段指令
DE_IR[31:0]	In	数据通路，E 段指令
EM_IR[31:0]	In	数据通路，M 段指令
MW_IR[31:0]	In	数据通路，W 段指令
TMux_GRF_RD1_Sel[2:0]	Out	数据通路，TMux_GRF_RD2 转发器选择信号
TMux_GRF_RD2_Sel[2:0]	Out	数据通路，TMux_GRF_RD2 转发器选择信号
TMux_EM_RD1_Sel[2:0]	Out	数据通路，TMux_GRF_RD2 转发器选择信号
TMux_EM_RD2_Sel[2:0]	Out	数据通路，TMux_GRF_RD2 转发器选择信号
TMux_MW_RD2_Sel[2:0]	Out	数据通路，TMux_GRF_RD2 转发器选择信号

转发采用了改进的“暴力转发，随时转发”策略，其意思为：被转发位置的指令无论是否需要用到某个寄存器的值，一旦此寄存器的新值在后续流水段中已经产生，则会进行转发。当多个后级流水线转发时，级数较低的优先级更高。

以 TMux_GRF_RD1 接受 X 段 PORT “寄存器管口” 更新值为例：

$$\mathrm{TMux_{GRFRD1}} = \big(RegWriteNonZero_X\ \&\&(IR_{DE}[rs] == A3_X)$$

$$\backslash\&\&(Tnew_X == 0)\&\&(Port_X == PORT)\big)?\ PORT.$$

e) TMux（转发多选器）

转发多选器 TMux 规格是同样的，其每个端口所对应供给者是固定的，因此存在空缺端口的情况。

表格 35 TMUX **端口表**

端口名称	类型	功能描述
TMux_Sel[2:0]	In	控制信号，转发器选择信号
Ori[31:0]	In	数据通路，原始通路
EM_ALU[31:0]	In	数据通路，EM 的 ALU 供给者
MW_ALU[31:0]	In	数据通路，MW 的 ALU 供给者
MW_MD	In	数据通路，MW 的 MD 供给者
Forward[31:0]	Out	数据通路，通路正确结果

三、 CPU 功能测试

1、 功能测试原则

- 所测试的指令不能够超出 CPU 支持的范围。(谨防同指令标识的拓展指令。)
- 测试的首要目标：全面覆盖性（例如：冲突覆盖、单指令界限等)。
- 测试步骤：控制信号（主控） -> 功能性检查 -> 抽象模块 GID 正确性检查->冒险覆盖性测试（暂停+转发）

2、 测试策略

1、 控制信号检查（主控)：独立检查每条指令的控制信号是否符合设计。

2、 数据通路检查：

策略：按照 50 条指令功能进行测试+转发测试。

请参见“MIPS 测试策略(功能测试)”。

3、 抽象模块 GID 正确性检查

原因：除去乘除模块 busy 信号导致的暂停，其余暂停和转发选择信号都是依据 GID 模块抽象出的运行参数进行判断，因此首先对 GID 模块的“抽象功能”进行覆盖性检测。(50 条指令*4 流水线段)

4、 冒险覆盖性测试：

a) 暂停：枚举所有 STOP 情况和部分 NOSTOP 情况，观察 Stall 信号。

b) 转发：覆盖性测试，自动化评测。

3、 测试实例

a) 控制信号正确性检查

使用下列代码检查各指令运行时，控制信号值是否符合期望，控制信号期望输出于第三节以表格给出，在此不再赘述。

```
# calr
add $1 $2 $3
addu $1 $2 $3
sub $1 $2 $3
subu $1 $2 $3
sll $1 $2 0
srl $1 $2 0
sra $1 $2 0
sllv $1 $2 $3
srlv $1 $2 $3
srav $1 $2 $3
and $1 $2 $3
or $1 $2 $3
xor $1 $2 $3
nor $1 $2 $3
slt $1 $2 $3
sltu $1 $2 $3
mult $1 $2
multu $1 $2
div $1 $2
divu $1 $2
mthi $1
mtlo $1
mfhi $1
mflo $1

# cali
addi $1 $2 0
addiu $1 $2 0
andi $1 $2 0
ori $1 $2 0
xori $1 $2 0
lui $1 0
lhu $1 0($2)

# st
sw $1 0($2)
sh $1 0($2)
sb $1 0($2)

# branch
beq $1 $2 label
bne $1 $2 label
blez $1 label
bgtz $1 label
bltz $1 label
```

```
bgez $1 label

# j
j label
jal label
jr $ra
jalr $1 $2
label:
```

代码 1 控制信号正确性检查代码

b) 单指令正确性检查

请参见“MIPS 测试策略(功能测试)”。

c) 冲突控制信号（暂停）检查

暂停控制采用弱抽象，在 GID 抽象模型已正确的情况下，通过表格枚举 calr, cali, ld, st, branch, j, jal, jr, jalr 九类指令的暂停情况，对于一类指令，其具体指令随机选择。

测试表格如下：

编号	位置	前序指令	冲突寄存器	实例代码
1	F-D	ld	Rt	lw $1, 0($2) sub $2, $2, $1
2	F-D	ld	Rs	lh $2, $0($3) ori $3, $2, 1
3	F-D	ld	Rs	lb $1, 0($2) lw $2, 0($1)
4	F-D	ld	Rs	lhu $1, 0($2) sh $2, 0($1)
5	F-D	ld	None	lbu $1, 0($2) sw $1, 0($0)
6	F-D	ld	Rt	lw $1, 0($0) beq $2,$1,label
7	F-D	ld	Rs	lw $1, 0($0) jr $1
8	F-D	ld	Rs	lw $1,0($0) jalr $ra $1
9	F-D	calr	Rs	addu $1,$2,$3 beq $1,$2,label
10	F-D	cali	Rs	ori $1,$0,100 bgtz $1,label
11	F-D	calr	Rs	addu $1,$2,$3 jr $1
12	F-D	cali	Rs	lui $1,0xf jr $1
13	F-D	calr	Rs	srav $1 $2 $3 jalr $ra $1
14	F-D	cali	Rs	xori $1 $0 0 jalr $ra $1
15	F-E	nop, ld	Rs	lbu $1 0($1) nop bne $1,$2,label
16	F-E	nop, ld	Rs	lhu $1 0($1) nop jr $1
17	F-E	nop, ld	Rs	lb $1 0($1) nop jalr $1
18	XALU	mult		mult $1 $2 mfhi
19	XALU	div		div $1 $2 mtlo

20	XALU	mtlo		mtlo $1 mflo $2

表格 36 暂停检查表格

d) 转发覆盖性检查

在 p5 中，由 lite 指令集合的转发组合已经达到上百种，在 50 条指令的 p6 中若采用此方法，已经不再现实。因此，转发策略要建立在更加抽象的模型上。

1. CPU 的转发策略检测基于供给-需求者模型，一共建立起了（5+5+4）=14 条转发路径。

2. 分析每条转发路径上可行的供给者和需求者，得到如下的表格：

表格 37 供给-需求者弱抽象表格

	需求者	供给者
D-M-RS-1	branch,jr,jalr	calr,cali,jal,jalr
D-M-RT-1	branch	calr,cali,jal,jalr
D-W-RS-2	branch,jr,jalr	calr,cali,jal,jalr,ld
D-W-RT-2	branch	calr,cali,jal,jalr,ld
D-W-RS-3	branch,jr,jalr	ld
D-W-RT-3	branch	ld
E-M-RS-1	calr,cali,st,ld	calr,cali,jal,jalr
E-M-RT-1	calr	calr,cali,jal,jalr
E-W-RS-2	calr,cali,st,ld	calr,cali,ld,jal,jalr
E-W-RT-2	calr	calr,cali,ld,jal,jalr
E-W-RS-3	calr,cali,st,ld	ld
E-W-RT-3	calr	ld
M-W-RT-2	st	calr,cali,ld,jal,jalr
M-W-RT-3	st	ld

3. 借助于 GID 完全覆盖性测试的结果，将需求者弱抽象，将供给者强抽象模型，得到 “强抽象供给者” - “弱抽象需求者” 的对应图：

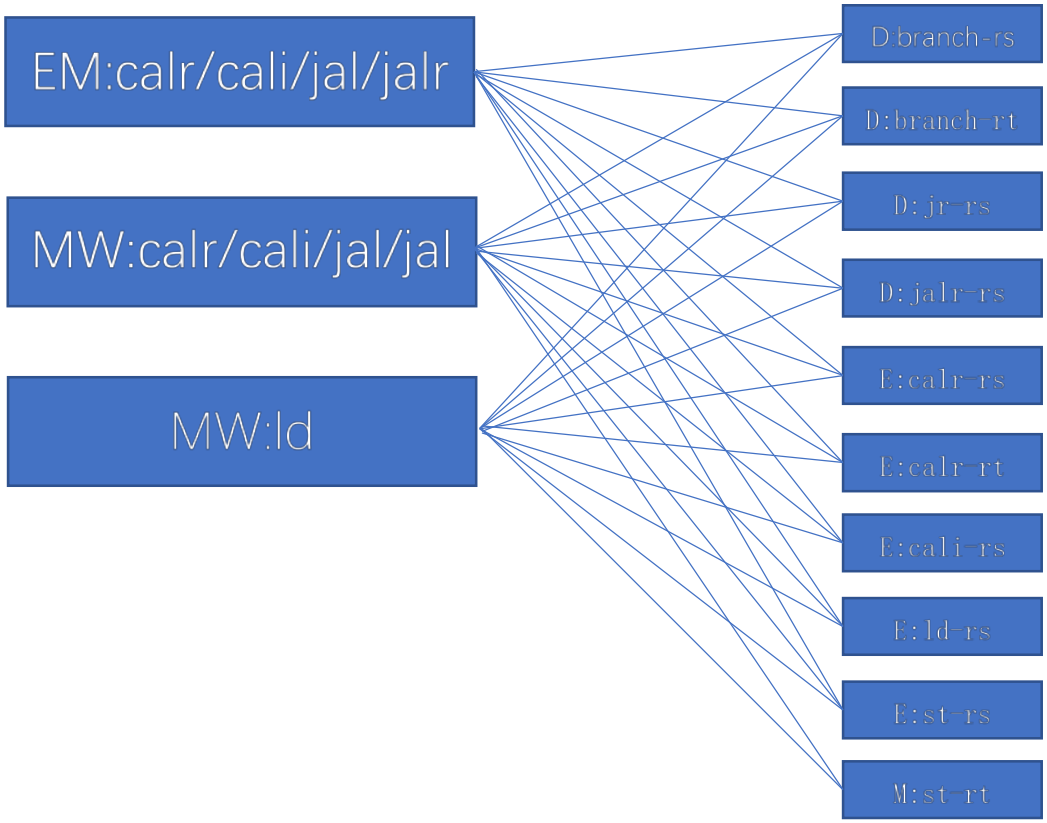

图 1 强抽象供给者” - “弱抽象需求者

4. 穷尽图中的 “每一条边” -26 条 (同一个抽象模型中具体指令随机选择)，即可完成基于抽象模型的覆盖性测试。

* ：强抽象：在弱抽象基础上，将指令抽象为 Tnew、Tuse、rwnz 等运行指标。CPU 的冲突控制单元正是建立在这种强抽象上的。

弱抽象：根据指令代码结构和功能划分为 calr、cali、branch 等信号。

**：抽象覆盖性测试的前提是 “GID 覆盖性测试完全充分”。

***：测试原理：加法变成乘法（类比于用二维地址表示一维数据）。

**** :强抽象同类划分参考依据 :在不同的流水段中，Tnew 和 D-Port 完全相同，且目标寄存器相同时 A3 和 RWNZ 相同。

四、本章思考题

- 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？

  答：若使用 ALU 模块则后续指令大概率会等待，会大幅度延长时钟周期，降低效率。而 Hi 和 Lo 寄存器不同于 GRF 中的寄存器在一个指令中具备写和读的同时操作，其行为方式比较像 DM，因此可以独立出来。

- 参照你对延迟槽的理解，试解释“乘除槽”。

  答：延迟槽可以理解为“跳的过程需要时间，中途还可以再执行一条”，因此乘除槽是指在乘除法运算的过程中“运行的指令”，具体而言，mult 指令后面 5 个周期中，像诸如 ld，st，calr，cali 类指令即可理解为乘除槽中的指令。

- 为何上文文末提到的 lb 等指令使用的数据扩展模块应在 MEM/WB 之后，而不能在 DM 之后?

  答：根据理论课老师的讨论，我认为 DM 的读操作是主要决定流水线时钟周期的步骤（读操作存在多级缓存，可能有不命中的情况出现，这时候读的时间长度会更长，写操作由于实际电路中存在写 buffer，因此会很快），若继续在 DM 层增加扩展模块，则时钟会继续延长。

- 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）

  答：字符由 0~255，恰好可以由一个字节来表示，这时便应该采取字节的方式进行存取访问。例如 mips 中申请字符串.asczii string “oasjdoiajsd asi ”。

- 如何概括你所设计的 CPU 的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？

答：CPU 风格：顶层布局简洁（IF、DM 字节处理、转发路径）、行为统一化（乘除加入 calr 型指令、lui 指令一般化、冲突判断逻辑）、模块功能专一化（GID、MCtrl、Hazard Units）、多级指令抽象（GID、wire 分类）。

我对抗复杂性的手段：

弱抽象（calr、cali、ld 等）和强抽象（tnew、tuse、rwnz 等）

通过现象看本质：指令层级的转发很难写，那就更深一层是哪些参数影响着转发逻辑的。

控制指令分通用控制和个性化控制分别对应不同的抽象层级

使用 head.v 文件宏定义参数

覆盖性测试与静态查错

- 你对流水线 CPU 设计风格有何见解？

  判断逻辑比较本质：一时间比较难以理解。

  CPU 简洁干练，自身比较完整，但“稳定性”不佳，对额外指令添加会较大破坏其规整性。

- 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。(**非常重要**)

答：实验中遇到了两类数据冒险，一类可以通过暂停来解决，而另一类则需通过转发来解决。下面针对两种情况分类作答。

a.通过“流水线工程化方法”，我们容易得到所有需要暂停的情况，且暂停情况相较于转发情况是相当有限的，本 CPU 的暂停策略如下：

表格 38 STALL **情况分析表格**

IF/ID当前指令			ID/EX(Tnew)					EX/MEM					MEM/WB
指令类型	源寄存器	Tuse	calr/rd/1	cali/rt/1	ld/rt/2	jal/$31/1	jalr/rd/1	calr/rd/0	cali/rt/0	ld/rt/1	jal/$31/0	jalr/rd/0	
calr	rs/rt	1			STOP								
cali	rs	1			STOP								
ld	rs	1			STOP								
st	rs	1			STOP								Stall_Impossible
st	rt	2											
btype	rs/rt	0	STOP	STOP	STOP	Impossible	Impossible			STOP			
jr	rs	0	STOP	STOP	STOP	Impossible	Impossible			STOP			
jalr	rs	0	STOP	STOP	STOP	Impossible	Impossible			STOP			

由以上表格，我罗列了伪代码表格，并在本章第四节附有实测代码：

b.转发相较于暂停，若只停留在指令覆盖性层面，则复杂性大幅度增长。因此对于转发正确性的测量，应该去到更抽象的维度，用更抽象的模型去代表一类指令，在实际测试时用少量的指令代表一类指令即可。因此这个测试问题量级就从一维变成了二维乘法了，测试中，首先是确保“抽象”过程的完全正确，其次随机选择少量指令代表性地检查。

1.具体的处理方法已经在验证章节——转发测试阶段详细说明，请参见。

2.关键问题 1——覆盖角度？CPU 电路中供给点和需求点的指令枚举。

3.关键问题 2——如何覆盖？乘法原理+抽象模型覆盖。

五、有关 CPU 扩展的说明

本 CPU 支持一定功能的扩展，需要在 Verilog 上进行改进。由于涉及到流水线的分层结构和冒险管理问题，因此较单周期 CPU 需要考虑的内容更多，因此在新增指令和调试时请务必参考以下的步骤。

1. 分析新增指令的需求，必要难以理解时及时借助 MARS 测试，将指令拆分为数据通路+控制逻辑。
2. 若时间充裕，以下所有的内容都应该**手稿分析后再实践**。
3. 数据通路：
   a) 根据指令需求绘制出数据通路图（注意分层和转发器位置）
   b) **先微观，再宏观：**若仅需细调部分部件功能，则实时调整；若需要新增通路，则应该以“尽量少增转发器和转发点”为原则构建，同时结合图与先前的数据通路表格，确定新增数据通路结构。
   c) 在完成数据通路搭建后，时间充裕情况下应进行检测。
4. 主控制指令：与单周期类似，可以**先按照单周期分析**，而后考虑流水线寄存器。若新增或修改指令，除此还需要考虑指令所在流水段，进行正确的添加。
5. ！冲突控制：分为暂停和转发
   a) 冲突控制一定要**借助通路图和工程化方法的指标**分析。
   b) 暂停：完善指令类型，明确指令的 Tnew，加入到 GID 中。
   c) ！转发：
      i. 若无新通路产生，需求解 Tuse，更改 GID，并细致分析。
      ii. 若有新的承载数据的通路，则 TMux、GID、TRANSMIT 可能需要都需要修改。

iii. 若有新的读取 GRF 的通路（不建议 !），则需新增 TMux，修改 GID 端口、修改 TRANSMIT。

d) 遇到问题：**看暂停表和 CPU 供给关键点图**。

6. 核查：沿着手绘的数据通路+两类控制单元复查。

7. 练手指令：movz, branch+link, jalr .