计算机组成 CPU 设计文档-P5

计算机组成 CPU 设计文档-P5 1
整体结构与概览 2
1、 CPU 基本参数与指标 2
2、 CPU 模块结构 2
一、 模块规格（数据通路） 4
1、 数据通路 4
2、 IFU（取指令单元） 6
3、 GRF（通用寄存器组） 6
4、 ALU（逻辑运算单元） 7
5、 DM（数据存储器） 8
6、 Ext（位数扩展器） 9
7、 NPC（分支跳转指令地址计算器） 9
8、 CMP（分支条件判断） 10
9、 PipeReg（流水线寄存器） 10
二、 模块规格（控制电路） 11
三、 CPU 功能测试 17
1、 功能测试原则 17
2、 测试策略 17
3、 测试实例 18
四、本章思考题 24
五、有关 CPU 扩展的说明 27
六、版本信息 29

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：{addu, subu, jr, sll(nop), ori, lui, beq, lw, sw, jal, jr, j}

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

a) 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、DM（数据存储器）：存储数据部件，支持读写。

5、EXT（位扩展器）：将 16 位数扩展为 32 位数，支持有/无符号扩展。

6、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。

7、CMP（分支比较器）：根据指令条件设置，返回判断结果。

8、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。

9、FuncMux（功能多选器）：对同一端口多个数据源进行筛选，目前有 AluSrc、WaSel、WdSel 三个。

10、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。

b) 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
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
	C	DE_IR[s]
EM_IR		DE_IR
EM_ALU		ALU
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
	WD	MUX_WdSel
	WPC(display)	MW_PC

表格 3 数据通路端口合成-转发

部件	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					
D_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					
E_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]					
EM_IR		DE_IR					
EM_ALU		ALU					
EM_RD2		DE_RD2					
M_PC(display)		DE_PC					
DM	Addr	EM_ALU					
	WD	TMUX_EM_RD2					
MW_IR		EM_IR					
MW_ALU		EM_ALU					
MW_MD		DM					
W_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 1024*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
000/0	32 位 +，不带溢出检测	addu, lw, sw
001/1	32 位 - ，不带溢出检测	subu
010/2	32 位 左移运算 B<<C	sll(nop)
011/3	32 位 |	ori
100/4	32 位 B<<16	lui
101/5	32 位 B+4（PC+8）	Jal

5、DM（数据存储器）

a) 端口

表格 9 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，接受异步复位信号
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
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 1024*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。

6、Ext（位数扩展器）

表格 10 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。

7、NPC（分支跳转指令地址计算器）

表格 11 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。

8、CMP（分支条件判断）

表格 12 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）

9、PipeReg（流水线寄存器）

表格 13 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 14 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	控制信号，IF，IF 接受外部 Pc 更新信号（1-允许，0-不允许）
D_ExtOp	Out	控制信号，16-32 位扩展类型（1-符号扩展，0-无符号扩展）
D_nPc_Sel[2:0]	Out	控制信号，PC 分支跳转类型 0：Branch 1：Jump/Jal 2：Jr
D_CmpOp	Out	控制信号，Branch 指令比较策略（0-EQUAL）
E_AluSrc[1:0]	Out	控制信号，MUX，ALU-B 端口选择器信号（1-扩展器，0-RD2）
E_AluOp[3:0]	Out	控制信号，ALU，ALU 驱动信号（具体请参加 ALU 功能表）
M_MemRead	Out	控制信号，Mem，Mem 读使能信号（1-允许，0-不允许:高阻）
M_MemWrite	Out	控制信号，Mem，Mem 写使能信号（1-允许，0-不允许）
W_WaSel[1:0]	Out	控制信号，MUX，GRF 写地址选择 0：Rt 1：Rs

		2：31
W_WdSel[1:0]	Out	控制信号，MUX，GRF 写数据选择 0：Alu 1：Memory
W_RegWrite	Out	控制信号，GRF，GRF 写使能信号 （1–允许，0-不允许）

b) 信号真值表

表格 15 主控单元信号真值表

	addu	subu	sll	ori	lui	lw	sw	beq	j	jal	jr
Branch_Jump	0	0	0	0	0	0	0	1	1	1	1
RegWrite	1	1	1	1	1	1	0	0	0	1	0
ExtOp	x(0)	x(0)	x(0)	0	x(0)	1	1	1	x(0)	x(0)	x(0)
nPc_Sel	x(0)	x(0)	x(0)	x(0)	x(0)	x(0)	x(0)	0	1	1	2
AluOp	Add (0)	Sub (1)	Sll (2)	Or (3)	s16 (4)	Add (0)	Add (0)	x (0)	x (0)	B+4 (5)	x (0)
MemRead	0	0	0	0	0	1	0	0	0	0	0
MemWrite	0	0	0	0	0	0	1	0	0	0	0
CmpOp	x(0)	x(0)	x(0)	x(0)	x(0)	x(0)	x(0)	0	x(0)	x(0)	x(0)
AluSrc	0	0	0	1	1	1	1	x(0)	x(0)	2	x(0)
WaSel	1	1	1	0	0	0	x(0)	x(0)	x(0)	2	x(0)
WdSel	0	0	0	0	0	1	x(0)	x(0)	x(0)	0	x(0)

*：lui 指令与 ori 指令：前者完全不顾及 GRF 读操作，且对扩展方式不在意。

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。

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 16 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

表格 17 指令分类与读写功能统计表格

指令分类：				读	写
calr	addu	subu	sll	√	√
cali	lui	ori		√	√
ld	lw			√	√
st	sw			√	
btype	beq			√	

[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》.

jal_type	jal				√
j_type	j				
jr_type	jr			√	

表格 18 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

表格 19 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	calr/rd 0	cali/rt 0	ld/rt 1	jal/$31 0	calr/rd 0	cali/rt 0	ld/rt 0	jal/$31 0

c) STALL 暂停控制模块

表格 20 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
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 段判断为例）：

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

$$Stall = Stall_{RS} \mid Stall_{Rt}$$

d) TRANSMIT 转发控制模块

表格 21 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：

$$\mathrm{TMux}_{\mathrm{GRFRD1}} = \big(RegWriteNonZero_X \,\&\&(IR_{DE}[rs] == A3_X)$$
$$\backslash\&\&(Tnew_X == 0)\&\&(Port_X == PORT)\big)?\, PORT.$$

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 22 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 寄存器的 ALU 供给者

MW_MD	In	数据通路，MW 寄存器的 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） -> 数据通路 -> 控制信号（暂停）->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：

a) Addu, subu: 正正（溢出/不溢)，负负（溢出/不溢)，正负，负正。

b) Lui，ori：负数的构造。

c) Sw，lw：非零数边界存储，正负偏移存储。

d) Beq：成立与不成立跳转，下跳与上跳。

e) Sll 检查：Nop 功能。

f) Jal 检查：正负绝对跳转，$31 赋值细节检查。(pc4)

g) Jr 检查：$31, $0, $8 检查

3、 控制信号（冲突）检查

a) GID 译码正确性：使用 testbench 模块测试 GID 对每条指令的返回结果。

b) STALL 正确性：使用 testbench 模块测试 STALL 模块所有 STOP 组

合。

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。

```
addu $t1, $t2, $t3
subu $t1, $t2, $t3
lw $t1, 12($t2)
sw $t1, 12($t2)
ori $t2, $t2, 0xffff
lui $t1, 0xf
sll $t1, $t2, 2
back:
beq $t1, $t2,back
jal next
j next
nop
next:
jr $ra
label:
```

代码 1 控制信号正确性检查代码

b) 单指令正确性检查

单指令正确性检查首先是 lui 和 ori，因为这两条指令才能构造非零数。而后，利用构造完成的各种数值，对指令 addu、subu、lw 和 sw 进行测试。

测试代码与期望结果如下：

```
# 单指令运算正确性检查:addu, subu, lw, sw, ori, lui   # 检查内容与期望结果
# Lui, Ori
lui $s0, 0x7ff1                # 功能检查：$16 = 0x7ff10000
# nop * 5
ori $s1, $zero, 0xf001          # 功能检查：$17 = 0x0000f001
# nop * 5
lui $s2, 0xffff
# nop * 5
ori $s3, $s2, 0xfffc            # 负数构造：$19 = -4
# nop * 5

# addu, subu
addu $t0, $s0, $s1              # 不溢出相加：$8 = 0x7ff1f001
# nop * 5
addu $t1, $s0, $s0              # 溢出相加：$9 = 0xffe20000
# nop * 5
addu $t2, $s3, $s3              # 负负相加 $10 = -8
# nop * 5
addu $t3, $s1, $t2              # 正负相加 $11 = 61433
# nop * 5
subu $t4, $s1, $t3              # 正正相减 $12 = 8
# nop * 5
subu $t5, $t3, $s1              # 正正相减 $13 = -8
# nop * 5
subu $t6, $s3, $t5              # 负负相减 $14 = 4
# nop * 5
lui $s4, 0x8000
# nop * 5
subu $t7, $s4, $t4              # 负正相减(溢出) $15 = 0x7ffffff8
# nop * 5

# sw, lw
ori $t8, 0
# nop * 5
sw $t0, 0($t8)                 # sw 下边界测试：mem[0] = 0x7ff1f001
# nop * 5
sw $t1, 16($t8)                # sw 偏移量测试：mem[4] = 0xffe20000
# nop * 5
ori $t9, 4092
# nop * 5
sw $t4, 0($t9)                 # sw 上边界测试：mem[1023] = 8
# nop * 5
sw $t5, -16($t9)                # sw 负偏移测试：mem[1019] = -8
# nop * 5

ori $t8, 13
# nop * 5
lw $s4, 3($t8)                 # 和为 4 倍偏测试：$20 = 0xffe20000
# nop * 5
lw $s5, 0($t9)                 # 上边界测试 $21 = 8
# nop * 5
lw $s6, 4($t9)                 # 无效界读入测试
# nop * 5
```

代码 2 **单指令真确性检查代码** 1

而后针对 beq、jal、jr 指令进行检测

```
# 单指令正确性检查：ori, lui, beq, jal, jr       # 检查期望结果和 pc 跳转情况
# beq                       # 检测目的与预期结果
ori $t0, 1
# nop*5
ori $t1, 2
# nop*5
ori $t2, 3
# nop*5
ori $t4, 1
# nop*5

addu $s0, $s1, $s2              # 非跳转语句选择无效检测：pc = pc + 4 (not pc +
4 + 0x8021)
# nop*5
beq $t0, $t1, label2              # 不跳转检测：pc = pc+4
# nop*5
nop
label1:
    lui $s0, 0xf000
    # nop*5
    beq $t0, $t4, label4          # 正跳转检测：pc = pc+4+8
    # nop*5
label2:
    lui $s1, 0x0f00
    # nop*5
label3:
    lui $s2, 0x00f0
    # nop*5
    jal function_begin
    # nop*5
    ori, $t5, 0x3048              # 非 ra 跳转，至 return 标签
    # nop*5
    jr $t5
    # nop*5
label4:
    lui $s3, 0x000f
    # nop*5
    addu $t0, $t0, $t4
    # nop*5
    beq $t0, $t1, label3          # 逆跳转检测：pc = pc+4-16
    # nop*5
    jal function_end
    # nop*5
return:
    ori $s6, $zero, 0x00f0
    # nop*5
    jal function_end
    # nop*5
function_begin:
    ori $s4, $zero, 0xf000
    # nop*5
    jr $ra
```

代码 3 **单指令正确性检查代码** 2

c) 冲突控制信号（暂停）检查

此时并未进行 CPU 检查，而是针对 STALL 模块进行 testbench 检查。

```
lw $1, 0($2)                    # calr - ld
subu $2, $1, $2

lw $1, 0($3)                    # cali - ld
ori $2, $0, 1

lw $1, 0($2)                    # ld - ld (rs)
lw $2, 0($1)

lw $1, 0($2)                    # st - ld (rs)
sw $2, 0($1)

lw $1, 0($2)                    # no stop! st-ld(rt)
sw $1, 0($0)

lw $1, 0($0)                    # btype - ld
beq $2,$1,label

lw $1, 0($0)                    # jr_type - ld
jr $1

addu $1,$2,$3                   # btype -calr
beq $1,$2,label

ori $1,$0,100                   # btype - cali
beq $2,$1,label

addu $1,$2,$3                   # jr_type - calr
jr $1

lui $1,0xf                      # jr_type - cali
jr $1

lw $1 0($1)                     # beq - ld
nop
beq $1,$2,label

lw $1 0($1)                     # jr - ld
nop
jr $1

    label:
```

代码 4 冲突控制信号（暂停）测试代码

d) 转发覆盖性检查

由于转发情况甚多，使用 testbench 检查过于复杂，因此采用自动化评测的方式测试，测试代码构建表格如下，测试代码过大，请参见附件。

	位置	前序指令	冲突寄存器	实例代码
1	D-M	nop, addu	Rs	addu $1, $2, $3 nop beq $1,$2, label
2	D-M	nop, sll	Rt	sll $1, $2, 5 nop beq $2, $1, label
3	D-M	nop, ori	Rs	ori $1, $1, 5 nop jr $1
4	D-M	nop, jal	Rt	jal label nop beq $ra $1 label
5	D-M	nop, jal	Rs	jal label nop jr $ra
6	D-W	nop, nop, lw	Rt	lw $1, 0($0) nop nop beq $1 $2 label
7	D-W	nop, nop, subu	Rs	subu $1, $2, $3 nop nop jr $1
8	D-W	nop, nop, lui	Rt	lui $1, 100 nop nop beq $2, $1, label
9	D-W	nop, nop, jal	Rs	jal label nop nop jr $ra
10	E-M	subu	Rs	subu $1, $2, $3 addu $2, $1, $2
11	E-M	lui	Rs	lui $1, 100 addu $2, $1, $1
12	E-M	sll	Rt	sll $1,$1,5 ori $1, $1, 1
13	E-M	jal	Rs	jal label lw $1, -4($31)
14	E-M	jal	Rs	jal label sw $1, 4($31)
15	E-M	ori	Rs	ori $1, 100

				lw $2, 100($1)

16	E-W	nop, lw	Rs	lw $1, 0($0)
				nop
				sw $2, 0($1)

17	E-W	nop. lw	Rt	lw $1, 0($0)
				nop
				subu $2,$1,$1

18	E-W	nop, lw	RS	lw $1, 80($0) nop ori $1,$1,0xf000

19	E-W	nop, lw	Rs	lw $1, 200($0)
				nop
				lw $2, 0($1)

20	E-W	nop, ori	Rt	ori $1 $2 4 nop addu $1 $0 $1
21	E-W	nop, jal	Rs	jal label nop sw $1 0($31)

22	E-W	nop, subu	Rs	subu $2,$1,$1
				nop
				lw $3,0($2)

23	M-W	lw	Rt	lw $1 0($0) sw $1 4($0)
24	M-W	lui	Rt	lui $1 100 sw $1 0($0)
25	M-W	jal	Rt	jal label sw $31 100($0)
26	M-W	subu	Rt	subu $1,$0,$1 sw $1, 100($0)

27	无转发			lw $0, 0($0)
				nop
				addu $1,$0,$1

28	无转发			lui $0 100 sw $0 0($1)

四、本章思考题

- 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 23 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	calr/rd/0	cali/rt/0	ld/rt/1	jal/$31/0	
calr	rs/rt	1			STOP						Stall_Impossible
cali	rs	1			STOP						
ld	rs	1			STOP						
st	rs	1			STOP						
st	rt	2									
btype	rs/rt	0	STOP	STOP	STOP	Impossible			STOP		
jr	rs	0	STOP	STOP	STOP	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：

表格 24 Stall **功能测试**

编号	位置	前序指令	冲突寄存器	实例代码
1	F-D	lw	Rs	lw $1, 0($2) subu $2, $1, $2
2	F-D	lw	Rt	lw $1, 0($3) ori $2, $0, 1
3	F-D	lw	Rs	lw $1, 0($2) lw $2, 0($1)
4	F-D	lw	Rt	lw $1, 0($2) sw $2, 0($1)
5	F-D	lw	None	lw $1, 0($2) sw $1, 0($0)
6	F-D	lw	Rs	lw $1, 0($0) beq $1,$2,label
7	F-D	lw	Rt	lw $1, 0($0) beq $2,$1,label
8	F-D	lw	Rs	lw $1, 0($0) jr $1
9	F-D	calr	Rs	addu $1,$2,$3 beq $1,$2,label
10	F-D	calr	Rt	ori $1,$0,100 beq $1,$2,label
11	F-D	cali	Rs	ori $1,$0,100 beq $1,$2,label
12	F-D	cali	Rt	ori $1,$0,100 beq $2,$1,label
13	F-D	calr	Rs	addu $1,$2,$3 jr $1
14	F-D	cali	Rs	lui $1,0xf jr $1
15	F-E	nop, lw	Rs	lw $1 0($1) nop beq $1,$2,label
16	F-E	nop, lw	Rs	lw $1 0($1) nop beq $2,$1,label
17	F-E	nop, lw	Rs	lw $1 0($1) nop jr $1

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.在本 CPU 中抽象方法是 GID（通用指令译码器），其测试方法是用 testbench 测试每一条指令，观察其输出值（建立的模型）是否正确。期望输出请参见第三节中的指令分类、Tnew、Tuse 表格。

2.基于抽象模型建立正确的基础上，分析转发者和需求者：

1. 需求者：

a) D:beq, jr

b) E:calr, cali, lw, sw

c) M:sw

2. 供给者

a) EM:calr, cali, jal

b) MW: lw+(calr, cali, jal)

*：括号（）中的内容代表其不实新晋的供给者，但是依然具有供给功能。

同时这样就构建出了 D-EM, D-MW, E-EM, E-MW, M-MW 五类转发情况，对于每类转发情况，从中需求者和供给者中选出有代表性的指令测试即可。（实际操作时是对新晋供给者指令进行全覆盖测试，同时包含 Rs 和 Rt，不是新晋的供给者则抽样测试。）具体测试伪代码和代码请参见第三节代码和附件。

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 .

六、版本信息

1. Version0：构建了不支持转发和暂停的流水线数据通路。
2. Version1：增加了 Hazard 文件，GID、STALL、TRANSMIT 和 TMUX 新增，支持暂停和转发。（无法通过中强测试）。
3. Version2：修复了 bugs。Beq 指令未成功执行时，外部返回的 pc 更新值应该是 pc+8；GID 模块中 lw 指令没有抽象成 ld 指令，增加了拓展性。