

# 流水线 CPU 设计文档 2.0

贾博驿

2023 年 11 月 19 日

# 设计实现指令

共 50 条  
实际指令 47 条

R 型指令：

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

I 型指令：

6	5	5	16
opcode	rs (base)	rt	imm16 (immediate, offset)

J 型指令：

6	26
opcode	instr_index

以下除分支与跳转类指令，所有的 RTL 省略  $PC \leftarrow PC + 4$ 。

## 算数与逻辑类

1. 加减法类

a. R 型

名称	funct	RTL
add	100000	$GPR[rd] \leftarrow GPR[rs] + GPR[rt]$
addu	100001	$GPR[rd] \leftarrow GPR[rs] + GPR[rt]$
sub	100010	$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$
subu	100011	$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$

注：实现的 add 实际上为 addu、sub 实际上为 subu。二者均按照无符号指令处理。

b. I 型

名称	opcode	RTL
addi	001000	$GPR[rt] \leftarrow GPR[rs] + \text{sign\_extend}(\text{immediate})$
addiu	001001	$GPR[rt] \leftarrow GPR[rs] + \text{sign\_extend}(\text{immediate})$

注：实现的 addi 实际上为 addiu。按照无符号指令处理。

2. 逻辑类

a. R 型

名称	funct	RTL
and	100100	$GPR[rd] \leftarrow GPR[rs] \text{ AND}(\text{bitwise}) GPR[rt]$
or	100101	$GPR[rd] \leftarrow GPR[rs] \text{ OR}(\text{bitwise}) GPR[rt]$
xor	100110	$GPR[rd] \leftarrow GPR[rs] \text{ XOR}(\text{bitwise}) GPR[rt]$
nor	100111	$GPR[rd] \leftarrow GPR[rs] \text{ NOR}(\text{bitwise}) GPR[rt]$
sll	000000	$GPR[rd] \leftarrow GPR[rt]_{(31-\text{shamt})..0} \parallel 0_{\text{shamt}}$
srl	000010	$GPR[rd] \leftarrow 0_{\text{shamt}} \parallel GPR[rt]_{31..\text{shamt}}$
sra	000011	$GPR[rd] \leftarrow (GPR[rt]_{31})_{\text{shamt}} \parallel GPR[rt]_{31..\text{shamt}}$
sllv	000100	$s \leftarrow GPR[rs]_{4..0} \quad GPR[rd] \leftarrow GPR[rt]_{(31-s)..0} \parallel 0_s$
srlv	000110	$s \leftarrow GPR[rs]_{4..0} \quad GPR[rd] \leftarrow 0_s \parallel GPR[rt]_{31..s}$
srav	000111	$s \leftarrow GPR[rs]_{4..0} \quad GPR[rd] \leftarrow (GPR[rt]_{31})_s \parallel GPR[rt]_{31..s}$

注：根据寄存器移位指令只依照寄存器值的低五位移位，高位的值不影响。

### b. I 型

名称	opcode	RTL
andi	001100	$GPR[rt] \leftarrow GPR[rs] \text{ AND}(\text{bitwise}) \text{ zero\_extend}(\text{immediate})$
ori	001101	$GPR[rt] \leftarrow GPR[rs] \text{ OR}(\text{bitwise}) \text{ zero\_extend}(\text{immediate})$
xori	001110	$GPR[rt] \leftarrow GPR[rs] \text{ XOR}(\text{bitwise}) \text{ zero\_extend}(\text{immediate})$

### 3. 乘除法类（R 型）

名称	funct	RTL
mult	011000	$\text{prod} \leftarrow GPR[rs]_{31..0} \times GPR[rt]_{31..0} \quad \text{LO} \leftarrow \text{prod}_{31..0} \quad \text{HI} \leftarrow \text{prod}_{63..32}$
multu	011001	$\text{prod} \leftarrow (0 \parallel GPR[rs]_{31..0}) \times (0 \parallel GPR[rt]_{31..0}) \quad \text{LO} \leftarrow \text{prod}_{31..0} \quad \text{HI} \leftarrow \text{prod}_{63..32}$
div	011010	$q \leftarrow GPR[rs]_{31..0} \div GPR[rt]_{31..0}$ $r \leftarrow GPR[rs]_{31..0} \% GPR[rt]_{31..0}$ $\text{LO} \leftarrow q$ $\text{HI} \leftarrow r$
divu	011011	$q \leftarrow (0 \parallel GPR[rs]_{31..0}) \div (0 \parallel GPR[rt]_{31..0})$ $r \leftarrow (0 \parallel GPR[rs]_{31..0}) \% (0 \parallel GPR[rt]_{31..0})$ $\text{LO} \leftarrow q_{31..0}$ $\text{HI} \leftarrow r_{31..0}$
mfhi	010000	$GPR[rd] \leftarrow \text{HI}$
mthi	010001	$\text{HI} \leftarrow GPR[rs]$
mflo	010010	$GPR[rd] \leftarrow \text{LO}$
mtlo	010011	$\text{LO} \leftarrow GPR[rs]$

## 寄存器操作类

### a. R 型

名称	funct	RTL
slt	101010	$GPR[rd] \leftarrow 0_{GPRLEN-1} \parallel GPR[rs] < GPR[rt]$
sltu	101011	$GPR[rd] \leftarrow 0_{GPRLEN-1} \parallel (0 \parallel GPR[rs]) < (0 \parallel GPR[rt])$

注：slt 类指令成立时置 1，不成立时置 0。

### b. I 型

名称	opcode	RTL
slti	001010	$GPR[rt] \leftarrow 0_{GPRLEN-1} \parallel GPR[rs] < \text{sign\_extend}(\text{immediate})$
sltiu	001011	$GPR[rt] \leftarrow 0_{GPRLEN-1} \parallel (0 \parallel GPR[rs]) < (0 \parallel \text{sign\_extend}(\text{immediate}))$
lui	001111	$GPR[rt] \leftarrow \text{immediate} \parallel 0_{16}$

注：sltiu 对立即数的处理是先有符号扩展再进行无符号比较。

## 分支与跳转类

### 1. 分支类（I 型）

名称	opcode	rt	RTL
(all)			<b>I+1:</b> if condition then $\text{PC} \leftarrow \text{PC} + \text{sign\_extend}(\text{offset} \parallel 0_2)$ endif
bltz	000001	00000	<b>I:</b> condition $\leftarrow GPR[rs] < 0_{32}$
bgez	000001	00001	<b>I:</b> condition $\leftarrow GPR[rs] \geq 0_{32}$

beq	000100	rt	I: condition $\leftarrow$ (GPR[rs] = GPR[rt])
bne	000101	rt	I: condition $\leftarrow$ (GPR[rs] $\neq$ GPR[rt])
blez	000110	00000	I: condition $\leftarrow$ GPR[rs] $\leq$ 0 <sub>32</sub>
bgtz	000111	00000	I: condition $\leftarrow$ GPR[rs] $>$ 0 <sub>32</sub>

注：流水线 CPU 分支指令不需要 PC + 4 后再加上 offset，直接在延迟槽指令的地址上加。

## 2. 跳转类

### a. R 型

名称	funct	RTL
jr	001000	I: I+1: PC $\leftarrow$ GPR[rs]
jalr	001001	I: GPR[rd] $\leftarrow$ PC + 8 I+1: PC $\leftarrow$ GPR[rs]

### b. J 型

名称	opcode	RTL
j	000010	I: I+1: PC $\leftarrow$ PC <sub>31..28</sub>    instr_index    0 <sub>2</sub>
jal	000011	I: GPR[31] $\leftarrow$ PC + 8 I+1: PC $\leftarrow$ PC <sub>31..28</sub>    instr_index    0 <sub>2</sub>

注：流水线 CPU 跳转指令由于分支指令存在应该加上 PC + 8，而非单周期的 PC + 4。

## 内存操作类（I 型）

名称	opcode	RTL
(all)		addr $\leftarrow$ sign_extend(offset) + GPR[base]
lb	100000	byte $\leftarrow$ addr <sub>1..0</sub> GPR[rt] $\leftarrow$ sign_extend(memory[addr] <sub>7+8*byte..8*byte</sub> )
lh	100001	half $\leftarrow$ addr <sub>1</sub> GPR[rt] $\leftarrow$ sign_extend(memory[addr] <sub>15+8*half..8*half</sub> )
lw	100011	GPR[rt] $\leftarrow$ memory[addr]
lbu	100100	byte $\leftarrow$ addr <sub>1..0</sub> GPR[rt] $\leftarrow$ zero_extend(memory[addr] <sub>7+8*byte..8*byte</sub> )
lhu	100101	half $\leftarrow$ addr <sub>1</sub> GPR[rt] $\leftarrow$ zero_extend(memory[addr] <sub>15+8*half..8*half</sub> )
sb	101000	memory[addr] <sub>7..0</sub> $\leftarrow$ GPR[rt] <sub>7..0</sub>
sh	101001	memory[addr] <sub>15..0</sub> $\leftarrow$ GPR[rt] <sub>15..0</sub>
sw	101011	memory[addr] <sub>31..0</sub> $\leftarrow$ GPR[rt]

注：针对 halfword 操作需要 addr 为 2 的非负整数倍，针对 word 操作需要 addr 为 4 的非负整数倍。针对 byte 和 halfword 操作时只修改对应 byte 和 halfword 的值，同一 word 的其他部分不变。

# 转发与暂停设计

## 指令分类

类型名	信号名	数量	指令
算数逻辑寄存器	ALREG	16	<div><div>_arith（算数）：<span>add</span>, addu, <span>sub</span>, subu</div><div>_logic（逻辑）：<span>and</span>, or, xor, nor</div><div>_shift（移位）：<span>sll</span>, srl, sra, sllv, srlv, srav</div><div>_comp（比较）：<span>slt</span>, sltu</div></div>
算数逻辑立即数	ALIMM	7	<div><div>_sign（有符号拓展）：<span>addi</span>, addiu, slti, sltiu</div><div>_zero（无符号拓展）：<span>andi</span>, ori, xori</div></div>
乘除法	MLUTDIV	4	<span>mult</span> , multu, div, divu
乘除寄存器写入	MLTO	2	<span>mthi</span> , mtlo
乘除寄存器读取	MLFROM	2	<span>mfhi</span> , mflo
高位加载	EXTLUI	1	<span>lui</span>
两数比较分支	BRANCHE	2	<span>beq</span> , bne
与零比较分支	BRANCHZ	4	<span>bltz</span> , bgez, blez, bgtz
仅读取跳转	JUMPR	1	<span>jr</span>
仅写入跳转	JUMPW	1	<span>jal</span>
读取写入跳转	JUMPRW	1	<span>jalr</span>
读取内存	LOAD	5	<span>lb</span> , lh, lw, lbu, lhu
写入内存	STORE	3	<span>sb</span> , sh, sw

注：不包括j指令，因为j指令不读取寄存器也不写入寄存器。

## Tuse 与 Tnew

信号名	Tuse_rs	Tuse_rt	GRF_WD_W_Sel	Tnew_E	Tnew_M	Tnew_W
ALREG	1	1	00	1	0	0
ALIMM	1	(3)	00	1	0	0
MLUTDIV	1	1	(00)	(0)	(0)	(0)
MLTO	1	(3)	(00)	(0)	(0)	(0)
MLFROM	(3)	(3)	00	1	0	0
EXTLUI	(3)	(3)	10	0	0	0
BRANCHE	0	0	(00)	(0)	(0)	(0)
BRANCHZ	0	(3)	(00)	(0)	(0)	(0)
JUMPR	0	(3)	(00)	(0)	(0)	(0)
JUMPW	(3)	(3)	11	0	0	0
JUMPRW	0	(3)	11	0	0	0
LOAD	1	(3)	01	2	1	0
STORE	1	2	(00)	(0)	(0)	(0)

注：

1. (3)表示该类指令不会读取 rs 或 rt 域翻译为寄存器编号后对应的寄存器，除 GRF 内部转发外不产生任何转发和暂停。
2. (00)和(0)表示表示该类指令 GRFWE = 0，虽然使用 ALU 数据但无意义。

## 转发与暂停

SPL_rs != 5'd0 && GRFWE_X && SPL_rs == GRF_A3_X									
流水段(X)	E				M				W
GRF_WD_W_Sel_X(Tnew_X)	00(1)	01(2)	10(0)	11(0)	00(0)	01(1)	10(0)	11(0)	All
Tuse_rs == 0	S1	S2	F43	F44	F21	S3	F23	F24	F1
Tuse_rs == 1	F51	S4	F43	F44	F21	F32	F23	F24	F1
SPL_rt != 5'd0 && GRFWE_X && SPL_rt == GRF_A3_X									
流水段(X)	E				M				W
GRF_WD_W_Sel_X(Tnew_X)	00(1)	01(2)	10(0)	11(0)	00(0)	01(1)	10(0)	11(0)	All
Tuse_rt == 0	S1	S2	F43	F44	F21	S3	F23	F24	F1
Tuse_rt == 1	F51	S4	F43	F44	F21	F32	F23	F24	F1
Tuse_rt == 2	F51	F62	F43	F44	F21	F32	F23	F24	F1
ISMULTDIV & (MULT_Start   MULT_Busy)									
SMULTDIV									

## 转发信号表

SPL_rs != 5'd0 && GRFWE_X && SPL_rs == GRF_A3_X			
类型	描述	FMUX_V1_D_Sel	FMUX_V1_E_Sel
S	暂停		
F51	V1_E 转发 OP_M		11
F43	V1_D 转发 ext32_E	110	
F44	V1_D 转发 pc8_E	111	
F32	V1_E 转发 DM_Q_W		10
F21	V1_D 转发 OP_M	011	
F23	V1_D 转发 ext32_M	100	
F24	V1_D 转发 pc8_M	101	
F1	GRF 内部转发		
F0	不转发	000	00

SPL_rt != 5'd0 && GRFWE_X && SPL_rt == GRF_A3_X				
类型	描述	FMUX_V2_D_Sel	FMUX_V2_E_Sel	FMUX_OP_M_Sel
S	暂停			
F62	DM_D_M 转发 DM_Q_W			1
F51	V2_E 转发 OP_M		11	
F43	V2_D 转发 ext32_E	110		
F44	V2_D 转发 pc8_E	111		
F32	V2_E 转发 DM_Q_W		10	
F21	V2_D 转发 OP_M	011		
F23	V2_D 转发 ext32_M	100		
F24	V2_D 转发 pc8_M	101		
F1	GRF 内部转发			
0	不转发	000	00	0

暂停变化表

I	I+1	I+2	I	I+1	I+2
S1	F21		S3	F1	
S2	S3	F1	S4	F32	

注：

1.优先级  $S > F4$ 、 $F5$ 、 $F6 > F2$ 、 $F3$ 。

2.F2 和 F3 之间、F4、F5 和 F6 之间不会发生冲突，因为转发的是同一流水级的不同数据。

3.当 MULT\_Busy 信号或 MULT\_Start\_E 信号为 1 时，MLUTDIV、MLTO、MLFROM 指令均被阻塞在 D 流水级，即 SMULTDIV 暂停。

4.未标注的信号取值与不转发相同。

# 模块设计

## IFU

1. 描述

取指令模块。内有 PC 子模块、NPC 子模块。接收分支和跳转指令的控制信号和数据并实现跳转。输出指令地址，通过与外部 ROM 通信获取指令。

2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
STALL_EN_N	I	暂停使能信号
NPCSel[1:0]	I	NPC 子模块控制信号
BranchComp	I	NPC 子模块分支指令控制信号
offset[15:0]	I	分支指令跳转偏移
instr_index[25:0]	I	j、jal 指令跳转地址
instr_register[31:0]	I	jr、jalr 指令跳转寄存器
InstrAddr[31:0]	O	即将进入流水线的指令地址，接入顶层 i_inst_addr

3. 控制信号

BranchComp: 接 COMP 的 BranchComp，表示分支指令的条件是否满足。

NPCSel:

指令（信号）名	取值	描述
其他	00	PC←PC+4
BRANCHE、BRANCHZ	01	使用 offset，结合 BranchComp 判断
j、jal	10	使用 instr_index
jr、jalr	11	使用 instr_register

## IFU\_PC

1. 描述

程序计数器子模块。在复位信号有效时复位为 32'h00003000。

2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
STALL_EN_N	I	暂停使能信号
D[31:0]	O	下一指令地址
Q[31:0]	O	即将进入流水线的指令地址

## IFU\_NPC

1. 描述

下一条指令地址计算子模块。输入 IFU 接收的分支和跳转指令的控制信号和数据，使用 MUX，计算出下一条指令的地址。

2. 接口

端口名	方向	描述
-----	----	----



PC[31:0]	I	即将进入流水线的指令地址
NPCSel[1:0]	I	同 IFU
BranchComp	I	同 IFU
offset[15:0]	I	同 IFU
instr_index[25:0]	I	同 IFU
instr_register[31:0]	I	同 IFU
NPC[31:0]	O	下一指令地址

## SPL

### 1. 描述

指令分线器模块。按照三种指令的格式将一条指令分为不同的部分，供其他模块使用。

### 2. 接口

端口名	方向	描述
Instr[31:0]	I	当前指令
opcode[5:0]	O	指令的操作数
rs[4:0]	O	R、I 型指令的 rs
rt[4:0]	O	R、I 型指令的 rt
rd[4:0]	O	R 型指令的 rd
shamt[4:0]	O	R 型指令的 shamt
funct[5:0]	O	R 型指令的 funct
imm16[15:0]	O	I 型指令的立即数
instr_index[25:0]	O	J 型指令的跳转地址

## GRF (GRF\_R, GRF\_W)

### 1. 描述

寄存器堆模块，共有 31 个寄存器（0 号寄存器直接接地）。通过 A1、A2 输入地址，可以取出指定寄存器的完成内部转发的数据。当 WE 使能时，通过 A3 输入地址，D 输入数据，可以修改指定寄存器存储的数据并进行内部转发。在流水线设计时分为两个部分体现。

### 2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
WE	I	写入使能信号，同时转发输出到顶层 w_grf_we
A1[4:0]	I	读取的第一个寄存器的编号
A2[4:0]	I	读取的第二个寄存器的编号
A3[4:0]	I	写入的寄存器的编号，同时转发输出到顶层 w_grf_addr
D[31:0]	I	写入寄存器的数据，同时转发输出到顶层 w_grf_wdata
V1[31:0]	O	读取的第一个寄存器的完成内部转发的值
V2[31:0]	O	读取的第二个寄存器的完成内部转发的值

### 3. 控制信号

GRFWE:

指令信号名	取值	描述
-------	----	----

其他	0	不涉及寄存器写入的指令
ALREG	1	
ALIMM、EXTLUI	1	
MLFROM	1	
LOAD	1	
JUMPW、JUMPRW	1	

#### 4. 关联 MUX 控制信号

GRF\_A3\_D\_Sel:

指令信号名	取值	描述
其他	00	接入 rd
ALIMM、EXTLUI、LOAD	10	接入 rt
JUMPW	11	固定 32'd31

GRF\_WD\_W\_Sel:

指令信号名	取值	描述
其他	00	接入 OP_W
LOAD	01	接入 DM_Q_W
EXTLUI	10	接入 ext32_W
JUMPW、JUMPRW	11	接入 pc8_W

## COMP

#### 1. 描述

比较模块，完两数相等、不等，一数有符号小于、小于等于、大于、大于等于零。比较结果用于判断分支。

#### 2. 接口

端口名	方向	描述
CompSel[2:0]	I	比较方式控制信号
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
BranchComp	O	比较结果

#### 3. 控制信号

CompSel:

指令名	取值	描述
bltz(00000)	000	有符号小于 0
bgez(00001)	001	有符号大于等于 0
beq(000100)	100	两数相等
bne(000101)	101	两数不等
blez(000110)	110	有符号小于等于 0
bgtz(000111)	111	有符号大于 0

## EXT

#### 1. 描述

位扩展模块，将 16 位立即数无符号扩展、有符号扩展、右补 16 位 0 为 32 位。

#### 2. 接口

端口名	方向	描述
EXTSel[1:0]	I	扩展方式控制信号
imm16[15:0]	I	16 位立即数
ext32[31:0]	O	32 位扩展结果

### 3. 控制信号

EXTSel:

指令名	取值	描述
其他	00	有符号拓展
andi、ori、xori	01	无符号拓展
lui	10	右补 16 位 0

## ALU

### 1. 描述

算数逻辑模块，完成算数运算（加、减）、逻辑运算（与、或、异或、或非）、移位运算（左移、逻辑右移、算数右移）、比较运算（两数有符号小于，无符号小于）。计算结果用于保存到寄存器。

### 2. 接口

端口名	方向	描述
OPSel[1:0]	I	OP 输出控制信号
FuncSel[1:0]	I	计算方式控制信号
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
shamt[4:0]	I	移位立即数
OP[31:0]	O	计算结果

### 3. 控制信号

OPSel:

指令（信号）名	取值	描述
add、addu、sub、subu、addi、addiu	00	算数运算
LOAD、STORE	00	算数运算
and、or、xor、nor、andi、ori、xori	01	逻辑运算
sll、srl、sra、sllv、srlv、sra	10	移位运算
slt、sltu、slti、sltiu	11	比较运算

FuncSel: 接 CTRL，根据 OPSel 选择。

OPSel == 00: (只用设置 01)

指令名	取值	描述
add、addu、addi、addiu	00	加法
LOAD、STORE	00	加法
sub、subu	01	减法

OPSel == 01: ALREG 接 funct[1:0]、ALIMM 时接 opcode[1:0]

指令名	取值	描述
and(100100)、andi(001100)	00	与
or(100101)、ori(001101)	01	或
xor(100110)、xori(001110)	10	异或

nor(100111)	11	或非
-------------	----	----

OPSel == 10: 接 funct[1:0]。

指令名	取值	描述
sll(000000)、sllv(000100)	00	逻辑左移
srl(000010)、srlv(000110)	10	逻辑右移
sra(000011)、srav(000111)	11	算数右移

OPSel == 11: (只用设置 01)

指令名	取值	描述
slt、slti	00	有符号两数小于
sltu、sltiu	01	无符号两数小于

#### 4. 关联 MUX 控制信号

ALU\_B\_E\_Sel:

指令信号名	取值	描述
ALREG	0	接入 GRF
ALIMM、LOAD、STORE	1	接入 EXT

## MULT

#### 1. 描述

乘除法模块，执行乘法的时间为 5 个时钟周期，执行除法的时间为 10 个时钟周期（包含写入内部的 HI、LO 寄存器）。自 1 个时钟周期 Start 信号有效后的第 1 个 clock 上升沿开始乘除法运算，同时 Busy 置位为 1。在运算结果保存到 HI 寄存器和 LO 寄存器后，Busy 清为 0。写入 HI、LO 均只需 1 个时钟周期，类似 GRF。读取 HI、LO 直接读取，类似 ALU。

#### 2. 接口

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号
ISMULTDIV	I	是否为乘除指令信号
MULTSel[1:0]	I	乘除模块控制信号
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
Start	O	乘除计算启动信号，输出 ISMULTDIV & MULTSel[2]
Busy	O	乘除计算进行信号
HILO[31:0]	O	读取的 HI、LO 的结果

#### 3. 控制信号

ISMULTDIV: 指令信号名为 MULTDIV、MLTO、MLFROM 时为 1。

MULTSel: 接 {funct[3], funct[1:0]}。

指令名	取值	描述
mult(011000)	100	准备执行有符号乘法操作
multu(011001)	101	准备执行无符号乘法操作
div(011010)	110	准备执行有符号除法操作
divu(011011)	111	准备执行无符号除法操作
mfhi(010000)	000	HILO 输出 HI
mflo(010010)	010	HILO 输出 LO

mthi(010001)	001	准备将 A 写入 HI
mtlo(010011)	011	准备将 A 写入 LO

#### 4. 关联 MUX 控制信号

OP\_E\_Sel:

指令信号名	取值	描述
其他	0	接入 ALU_OP
MLFROM	1	接入 MULT_HILO

### DM

#### 1. 描述

内存处理模块。将控制信号和数据处理为与外部 RAM 通信的合法信号。

#### 2. 接口

端口名	方向	描述
WE	I	写入使能信号
DMSel[2:0]	I	读取存储模式控制信号
A[31:0]	I	读取或写入的地址，同时转发输出到顶层 m_data_addr
D[31:0]	I	原始的写入数据
rdata[31:0]	I	原始的读取数据，接入顶层 m_data_rdata
wdata[31:0]	O	处理后的读取数据，接入顶层 m_data_wdata
byteen[3:0]	O	字节使能信号，接入顶层 m_data_byteen
Q[31:0]	O	处理后的读取数据

注：字节使能信号表示 D 的哪些字节要被写入，若要将低位的数据写入高位需要进行移位。

#### 3. 控制信号

DMWE:

指令信号名	取值	描述
其他	0	不涉及写入寄存器的指令
STORE	1	

DMSel: 接 opcode[2:0]。

指令名	取值	描述
lb(100000)、sb(101000)	000	针对 byte 操作，读取时有符号拓展
lh(100001)、sh(101001)	001	针对 halfword 操作，读取时有符号拓展
lw(100011)、sw(101011)	011	针对 word 操作
lbu(100100)	100	针对 byte 操作，读取时无符号拓展
lhu(100101)	101	针对 halfword 操作，读取时无符号拓展

#### 4. 输出控制信号

byteen:

指令名	DMWE	DMSel	A[1:0]	byteen						
其他	0		00	0000	01	0000	10	0000	11	0000
sb	1	000		0001		0010		0100		1000
sh	1	001		0011		/		1100		/
sw	1	011		1111		/		/		/

### FR\_D, FR\_E, FR\_M, FR\_W

# 1. 描述

D、E、M、W 段流水线寄存器。

# 2. 接口

通用：

端口名	方向	描述
RESET	I	同步复位信号
clk	I	时钟信号

以下端口均有 I 和 O 两个方向，I 方向信号名以 D\_开头，O 方向信号名以 Q\_开头。

FR\_D:

端口名	接入
STALL_EN_N	CTRL_S_FR_D_EN
Instr[31:0]	顶层 i_inst_rdata
InstrAddr[31:0]	IFU_InstrAddr

FR\_E:

端口名	接入
STALL_RESET	CTRL_S_FR_E_RESET
DMWE	CTRL_D_DMWE
GRFWE	CTRL_D_GRFWE
DMSel[2:0]	CTRL_D_DMSel
FuncSel[1:0]	CTRL_D_FuncSel
OPSel[1:0]	CTRL_D_OPSel
GRF_WD_W_Sel[1:0]	CTRL_D_GRF_WD_W_Sel
ALU_B_E_Sel	CTRL_D_ALU_B_E_Sel
OP_E_Sel	CTRL_D_OP_E_Sel
MULTSel[2:0]	CTRL_D_MULTSel
ISMULTDIV	CTRL_D_ISMUTLDIV
V1[31:0]	FMUX_V1_D
V2[31:0]	FMUX_V2_D
shamt[4:0]	SPL_shamt
GRF_A3[4:0]	MUX_GRF_A3_D
ext32[31:0]	EXT_ext32
pc8[31:0]	pc8_D
FMUX_V1_E_Sel[1:0]	CTRL_F_FMUX_V1_E_Sel
FMUX_V2_E_Sel[1:0]	CTRL_F_FMUX_V2_E_Sel
FMUX_DM_D_M_Sel	CTRL_F_FMUX_DM_D_M_Sel

FR\_M:

端口名	接入
DMWE	DMWE_E
GRFWE	GRFWE_E
DMSel[2:0]	DMSel_E
GRF_WD_W_Sel[1:0]	GRF_WD_W_Sel_E
V2[31:0]	FMUX_V2_E
OP[31:0]	MUX_OP_E

GRF_A3[4:0]	GRF_A3_E
ext32[31:0]	ext32_E
pc8[31:0]	pc8_E
FMUX_DM_D_M_Sel	FMUX_DM_D_M_Sel_E

FR\_W:

端口名	接入
GRFWE	GRFWE_M
GRF_WD_W_Sel[1:0]	GRF_WD_W_Sel_M
OP[31:0]	OP_M
DM_Q[31:0]	DM_Q
GRF_A3[4:0]	GRF_A3_M
ext32[31:0]	ext32_M
pc8[31:0]	pc8_M

## CTRL\_Decoder

### 1. 描述

译码器模块。通过 opcode、funct 和 rt 判断指令的类型，输出各个模块的控制信号。

### 2. 接口

端口名	方向	端口名	方向
opcode[5:0]	I	DMSel[2:0]	O
funct[5:0]	I	DMWE	O
rt[4:0]	I	GRFWE	O
CompSel[2:0]	O	GRF_A3_D_Sel[1:0]	O
EXTSel[1:0]	O	GRF_WD_W_Sel[1:0]	O
NPCSel[1:0]	O	ALU_B_E_Sel	O
OpSel[1:0]	O	OP_E_Sel	O
FuncSel[1:0]	O	Tuse_rs[1:0]	O
MULTSel[2:0]	O	Tuse_rt[1:0]	O
ISMULTDIV	O		

## CTRL\_Stall

### 1. 描述

暂停控制器模块。判断是否需要暂停，若需要暂停则产生暂停信号。

### 2. 接口

端口名	方向	端口名	方向
Tuse_rs[1:0]	I	GRF_A3_E[4:0]	I
Tuse_rt[1:0]	I	GRF_A3_M[4:0]	I
SPL_rs[4:0]	I	ISMULTDIV	I
SPL_rt[4:0]	I	MULT_Start	I
GRFWE_E	I	MULT_Busy	I
GRFWE_M	I	IFU_EN_N	O
GRF_WD_W_Sel_E[1:0]	I	FR_D_EN_N	O
GRF_WD_W_Sel_M[1:0]	I	FR_E_RESET	O

## CTRL\_Forward

### 1. 描述

转发控制器模块。判断是否需要转发，若需要转发转发什么数据。

### 2. 接口

端口名	方向	端口名	方向
Tuse_rs[1:0]	I	GRF_A3_E[4:0]	I
Tuse_rt[1:0]	I	GRF_A3_M[4:0]	I
SPL_rs[4:0]	I	FMUX_V1_D_Sel[2:0]	O
SPL_rt[4:0]	I	FMUX_V2_D_Sel[2:0]	O
GRFWE_E	I	FMUX_V1_E_Sel[1:0]	O
GRFWE_M	I	FMUX_V2_E_Sel[1:0]	O
GRF_WD_W_Sel_E[1:0]	I	FMUX_DM_D_M_Sel	O
GRF_WD_W_Sel_M[1:0]	I		



## P5 思考题

1. 在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

类型名	拓展部分
算数逻辑寄存器	ALU 拓展对应的运算。 CTRL_Decoder 加 ALU 的控制信号。
算数逻辑立即数	ALU 拓展对应的运算。 EXT 拓展对应的扩展方式。 CTRL_Decoder 加 ALU 和 EXT 的控制信号。
立即数加载	EXT 拓展对应的扩展方式。 CTRL_Decoder 加 ALU 和 EXT 的控制信号。
比较与跳转	COMP 拓展对应的比较方式。 CTRL_Decoder 加 COMP 和 NPC 的控制信号。
读取内存	DM 拓展对应的读取的方式。 CTRL_Decoder 加 DM 的控制信号。
写入内存	DM 拓展对应的读取的方式。 CTRL_Decoder 加 DM 的控制信号。

此外还有两种：

- 1.条件写入寄存器（内存）。实现方法如下：拓展 COMP。不执行该指令时，寄存器（内存）的使能信号为控制器输出的。执行该指令时，将控制器输出的信号与 BranchCOMP 进行与然后再进入 E 级。

- 2.不同条件写入寄存器（内存）不同种类的数据。实现方法如下：若正常情况下写入的数据为 ext32 或者 pc8，执行该指令时将 BranchCOMP 的信号作为与 ext32 或者 pc8 的信号相连的 MUX 的控制信号。选择 ext32/pc8 与给定的其他值。若正常情况下写入的数据不是 ext32 或者 pc8，还要将 BranchCOMP 的信号作为 MUX\_GRP\_A3\_D 后的一个 MUX 的控制信号，正常情况使用 MUX\_GRP\_A3\_D，异常情况转到 ext32 或者 pc8。

注意以上两种情况控制信号不能影响到正常使用 EXT 或者 pc8 信号的指令。

2. 确定你的译码方式，简要描述你的译码器架构，并思考该架构的优势以及不足。

采用集中式译码方式。所有的译码工作均在 D 级完成。译码器分为解码器\_Decoder，暂停器\_Stall 和转发器\_Forward。解码器负责通过指令输出各个模块的控制信号。暂停器负责根据当前流水线的情况确定是否要在 D 级暂停。转发器负责根据当前流水线的情况确定是否要进行转发以及如何进行转发。

优势：译码器数量少，移植和进行拓展快，控制信号跟随流水线往下走非常清晰。

不足：单个译码器代码庞大，需要在流水线寄存器中增加用于传递控制信号的寄存器。暂停器和转发器的输入信号存在重复。

3. 我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这

种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

第三题若分支指令的判断在 ALU 处完成，则其  $T_{use}=1$ 。以下情况将不会发生阻塞。但是提前到 D 级执行，将导致阻塞。

```
lw $1, 0($0)
ori $2, $0, 0
beq $1, $2, label
```

4. 因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回  $PC+8$ ，请

思考为什么这样设计？

因为  $PC+4$  是延迟槽指令的地址，跳转返回时不是从延迟槽指令开始而是从延迟槽后的一条指令开始。所以需要写回  $PC+8$ 。

5. 我们要求大家所有转发数据都来源于流水寄存器而不能是功能部件（如 DM、ALU），

请思考为什么？

因为在一个周期中的很长一段时间，功能部件因为存在延迟，输出的都不是期望值，如果转发这些值进行后续指令的计算，将会导致后续指令的计算出现错误，等到功能部件输出期望值时，后续指令的使用的功能部件可能来不及得到新的正确值一个周期就结束了。而流水线寄存器的值是稳定输出的期望值。

6. 我们为什么要使用 GPR 内部转发？该如何实现？

通过 GPR 内部转发可以统一解决 W 级向 D 级的转发问题。因为 W 级所有指令的  $T_{use}$  都为 0，数据都可以转发。且无论在之后的哪一级使用 W 级要写入的寄存器的数据，都可以在通过 GPR 内部转发在 D 级完成转发。实现方式如下：

```
assign RD1 = (A1 == 5'd0) ? 32'h00000000 : RF[A1];
assign RD2 = (A2 == 5'd0) ? 32'h00000000 : RF[A2];
assign V1 = (A1 != 5'd0 && WE && A3 == A1) ? D : RD1;
assign V2 = (A2 != 5'd0 && WE && A3 == A2) ? D : RD2;
```

7. 我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路？

见上文转发与暂停设计中“转发与暂停”表。

## P6 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 CPU 执行乘除法的时间较长, 而且执行乘除法的次数不多。如果将乘除法整合进 ALU, 执行次数更多的其他算数逻辑指令将会被阻塞, 设计单独的乘除法部件可以让较慢的乘除法在执行过程中, 其他的仍然可以执行。因为乘除法的结果不能用一个寄存器表示, 需要使用两个寄存器, 这两个寄存器专用于乘除法。由高内聚低耦合的原则, 需要设置独立的 HI、LO 寄存器而不是使用通用寄存器。

2. 真实的流水线 CPU 是如何使用实现乘除法的? 请查阅相关资料进行简单说明。

使用独立于主流流水线的特殊硬件功能单元来处理整数乘法和除法, 可能使用器件包括乘累加乘法器、阵列乘法器、Baugh-Wooley 乘法器、Radix-2 Booth 乘法器、Radix-4 Booth 乘法器、恢复余数法除法器、不恢复余数法除法器、基 2 SRT 算法除法器。

3. 请结合自己的实现分析, 你是如何处理 Busy 信号带来的周期阻塞的?

当 Busy 信号或 Start 信号为 1 时且进入 D 级的指令为 MULTDIV、MLTO、MLFROM 时暂停, 其余指令因为不使用的乘除法模块按照正常的转发暂停逻辑执行。

4. 请问采用字节使能信号的方式处理写指令有什么好处? (提示: 从清晰性、统一性等角度考虑)

清晰性上, 通过字节使能信号能够直接读出需要修改的字节。

统一性上, 统一了 DM 写使能信号和具体写入的字节的地址信号, 无需传递两个信号。

5. 请思考, 我们在按字节读和按字节写时, 实际从 DM 获得的数据和向 DM 写入的数据是否是一字节? 在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢?

按字节读和按字节写的时候获得的数据并不是一字节, 而是一字。只是将字的其他部分进行了处理, 当对单个字节进行读写时, 按字节读和按字节写的效率更高, 例如在 C 语言中处理 char 类型数组时, 如果按字读和按字写还要在一个字进行定位。

6. 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?

主要采取的手段包括将指令进行分类 (见转发与暂停设计一节), 利用指令的分类进行译码 (见各模块控制信号), 在处理数据冲突的时候也是将分类之后的指令进行 Tuse 和 Tnew 的归类来进行暂停与转发的设计 (见转发与暂停设计一节)

7. 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？

发现并处理的指令类型组合产生的冲突见转发与暂停设计一节的转发与暂停表，根据数据的情况全力进行转发，无法通过转发处理的进行暂停处理。使用 Python 辅以序列设计进行测试样例的构建。

8. 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

使用 Python 辅以序列设计进行测试样例的构建，先确定要进行测试的指令的组合，然后通过 `lui` 和 `ori` 进行组合对初始数据进行赋值，然后将要测试的指令放置在相应的流水级，前面的流水级随机从可生成数据的指令中选取，后面的流水级均用与冲突寄存器无关的指令填充。生成时模拟指令的行为，如 `beq` 模拟指令的行为计算跳转时的值，在初始数据中赋值，`bne` 模拟指令的行为计算不跳转时的值，并且在初始数据中赋值。

面前遇到的问题是 `jal` 和 `jr` 的测试比较困难。