数据通路

五级流水线

阶段	简称	功能概述
取指阶段（Fetch）	F	从指令存储器中读取指令
译码阶段（Decode）	D	从寄存器文件中读取源操作数并对指令译码以便得到控制信号
执行阶段（Execute）	E	使用 ALU 执行计算
存储阶段（Memory）	M	读或写数据存储器
写回阶段（Writeback）	W	将结果写回到寄存器文件

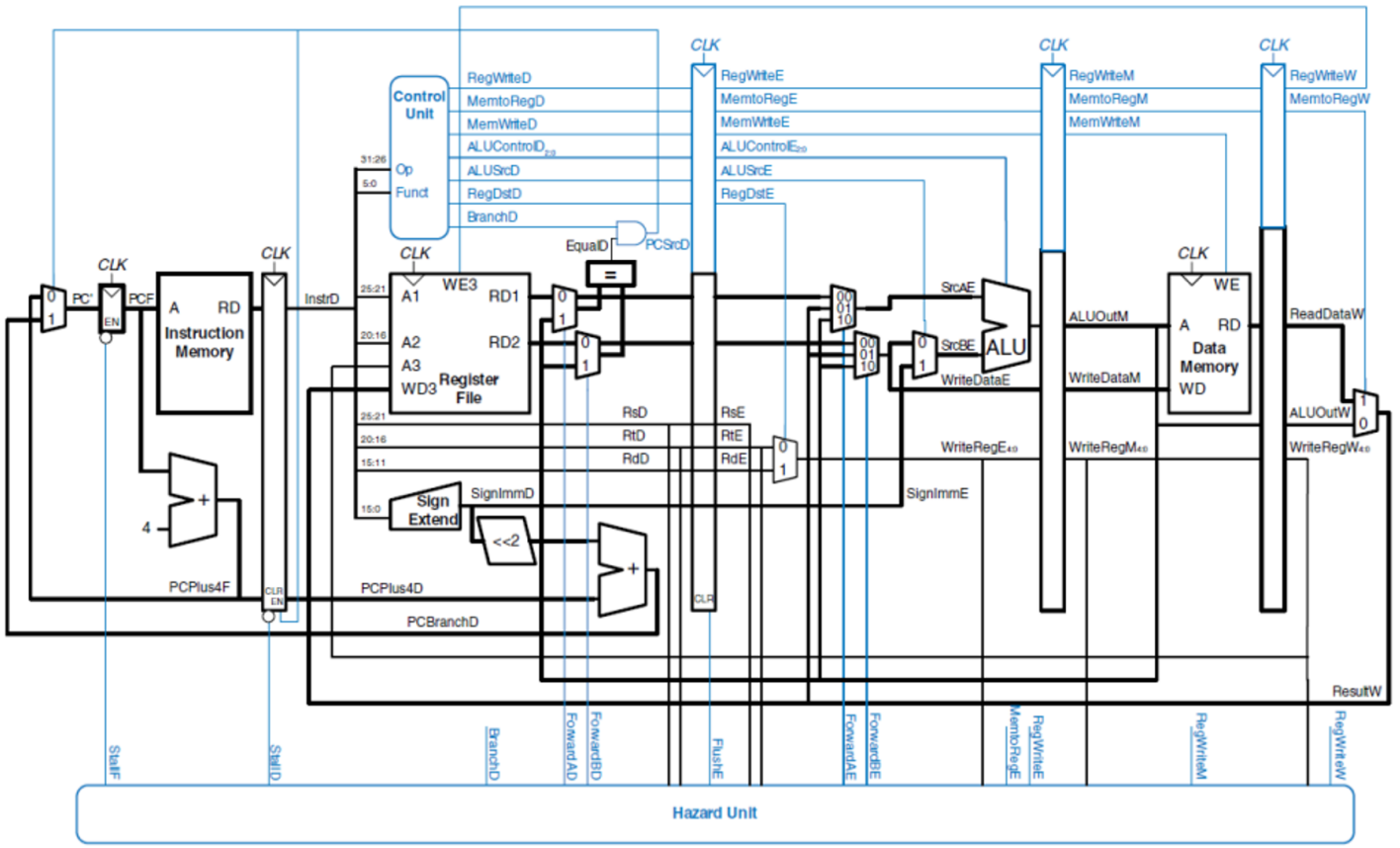

Figure 7.58 Pipelined processor with full hazard handling

F级

F_PC

信号名	方向	位宽	描述
clk	I	1	时钟信号

信号名	方向	位宽	描述
reset	I	1	同步复位信号
PC_en	I	1	PC使能信号
NPC	I	32	next PC
PC	O	32	PC

F_IM

信号名	方向	位宽	描述
PC	I	32	PC
Instr	O	32	Instruction

Fetch

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
PC_en	I	1	PC使能信号（冻结）
d_NPC	I	32	D级产生的next PC
f_PC	O	32	F级产生的PC
f_Instr	O	32	F级产生的Instruction

IF/ID寄存器

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
IFID_en	I	1	IF/ID寄存器使能信号（冻结）
f_PC	I	32	流水F级PC -in
f_Instr	I	32	流水F级Instruction -in
IFID_PC	O	32	流水F级PC -out
IFID_Instr	O	32	流水F级Instruction -out

D_W级

D_CMP

信号名	方向	位宽	描述
A	I	32	输入数据1
B	I	32	输入数据2
Equ	O	1	beq比较判断结果

D_EXT

信号名	方向	位宽	描述
EXTOp	I	2	EXT功能选择信号
EXTin	I	16	16imm
EXTout	O	32	扩展结果

D_GRF

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
RegWrite	I	1	写寄存器使能信号
ReadReg1	I	5	读地址输入1
ReadReg2	I	5	读地址输入2
WriteReg	I	5	写地址输入
WriteData	I	32	写数据输入
PC	I	32	当前PC
ReadData1	O	32	读数据输出1
ReadData1	O	32	读数据输出1

D_PCcounter

信号名	方向	位宽	描述
PCSrc	I	3	NPC输出选择信号
Equ	I	1	beq比较判断结果

信号名	方向	位宽	描述
f_PC	I	32	F级PC
IFID_PC	I	32	IF/ID流水线PC
imm26	I	26	26imm
GPR_jump	I	32	跳转到寄存器的值
NPC	O	32	D级NPC输出
pc8	O	32	PC + 8 输出

Decode_W

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
f_PC	I	32	F级PC输入
IFID_PC	I	32	IF/ID寄存器PC输入
IFID_Instr	I	32	IF/ID寄存器Instruction输入
MEMWB_PC	I	32	MEM/WB寄存器PC输入
Forward_rs_D	I	2	D级rs转发信号
Forward_rt_D	I	2	D级rt转发信号
IDEX_Dout	I	32	ID/EX寄存器Dout转发数据
EXMEM_Eout	I	32	EX/MEM寄存器Eout转发数据
MEMWB_Mout	I	32	MEM/WB寄存器Mout转发数据
w_RegWrite	I	W级写寄存器使能信号	
MEMWB_WriteReg	I	5	W级写寄存器地址
d_PC	O	32	D级的PC
d_Instr	O	32	D级的Instruction
d_WriteReg	O	5	D级产生的写寄存器地址
d_Dout	O	32	D级EXTout和pc8选择后产生的Dout
d_NPC	O	32	D级产生的next PC
d_MF_rs	O	32	D级转发后的rs数据

信号名	方向	位宽	描述
d_MF_rt	O	32	D级转发后的rt数据

ID/EX寄存器

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
d_PC	I	32	流水D级PC -in
d_Instr	I	32	流水D级Instrution -in
d_WriteReg	I	5	流水D级写寄存器地址 -in
d_Dout	I	32	流水D级Dout -in
d_MF_rs	I	32	流水D级转发后rs数据 -in
d_MF_rt	I	32	流水D级转发后rt数据 -in
IDEX_PC	O	32	流水D级PC -out
IDEX_Instr	O	32	流水D级Instrution -out
IDEX_WriteReg	O	5	流水D级写寄存器地址 -out
IDEX_Dout	O	32	流水D级Dout -out
IDEX_MF_rs	O	32	流水D级转发后rs数据 -out
IDEX_MF_rt	O	32	流水D级转发后rt数据 -out

E级

E_ALU

信号名	方向	位宽	描述
ALUOp	I	3	ALU功能选择信号
A	I	32	输入数据1
B	I	32	输入数据2
Result	O	32	ALU运算结果输出

Execute

信号名	方向	位宽	描述
IDEX_PC	I	32	ID/EX寄存器PC输入
IDEX_Instr	I	32	ID/EX寄存器Instruction输入
Forward_rs_E	I	2	E级rs转发信号
Forward_rt_E	I	2	E级rt转发信号
EXMEM_Eout	I	32	EX/MEM寄存器Eout转发数据
MEMWB_Mout	I	32	MEM/WB寄存器Mout转发数据
IDEX_WriteReg	I	5	ID/EX寄存器写寄存器地址输入
IDEX_Dout	I	32	ID/EX寄存器Dout输入
IDEX_RD1	I	32	ID/EX寄存器RD1输入
IDEX_RD2	I	32	ID/EX寄存器RD2输入
e_RegWrite	O	1	E级写寄存器使能信号
e_PC	O	32	E级的PC
e_Instr	O	32	E级的Instruction
e_WriteReg	O	5	E级写寄存器地址
e_Eout	O	32	E级Dout和ALUout选择后产生的Eout
e_RD2	O	32	E级RD2数据

EX/MEM寄存器

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
e_PC	I	32	流水E级PC -in
e_Instr	I	32	流水E级Instruction -in
e_WriteReg	I	5	流水E级写寄存器地址 -in
e_Eout	I	32	流水E级Eout -in
e_RD2	I	32	流水E级RD2数据 -in
EXMEM_PC	O	32	流水E级PC -out

信号名	方向	位宽	描述
EXMEM_Instr	O	32	流水E级Instruction -out
EXMEM_WriteReg	O	5	流水E级写寄存器地址 -out
EXMEM_Eout	O	32	流水E级Eout -out
EXMEM_RD2	O	32	流水E级RD2数据 -out

M级

M_DM

信号名	方向	位宽	描述
PC	I	32	当前PC
clk	I	1	时钟信号
reset	I	1	同步复位信号
MemWrite	I	1	内存写使能信号
DMaddr	I	32	内存地址输入
DMin	I	32	内存写数据输入
DMout	O	32	内存读数据输出

Memory

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
EXMEM_PC	I	32	EX/MEM寄存器PC输入
EXMEM_Instr	I	32	EX/MEM寄存器Instruction输入
Forward_rt_M	I	2	M级rt转发信号
MEMWB_Mout	I	2	MEM/WB寄存器Mout转发数据
EXMEM_WriteReg	I	5	EX/MEM寄存器写寄存器地址输入
EXMEM_Eout	I	32	EX/MEM寄存器Eout输入
EXMEM_RD2	I	32	EX/MEM寄存器RD2输入
m_RegWrite	O	1	M级写寄存器使能信号

信号名	方向	位宽	描述
m_PC	O	32	M级的PC
m_Instr	O	32	M级的Instruction
m_Mout	O	32	M级Eout和DMout选择后产生的Mout
m_WriteReg	O	5	M级写寄存器地址

MEM/WB寄存器

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
m_PC	I	32	流水M级PC -in
m_Instr	I	32	流水M级Instruction -in
m_Mout	I	32	流水M级Mout -in
m_WriteReg	I	5	流水M级写寄存器地址 -in
MEMWB_PC	O	32	流水M级PC -out
MEMWB_Instr	O	32	流水M级Instruction -out
MEMWB_Mout	O	32	流水M级Mout -out
MEMWB_WriteReg	O	5	流水M级写寄存器地址 -out

控制模块

采用分布式译码

控制信号	失效时作用（0）	有效时作用（1）
RegDst[1:0]	GRF写入端地址选择Rt字段	GRF写入端地址选择Rd字段
RegDst[1:0]	GRF写入端地址选择GPR[jump]	无
ALUSrc[1:0]	00:ALU输入端B选择RD2	01:ALU输入端B选择Dout
ALUSrc[1:0]	无	无
WBreg[1:0]	00:GRF写入端数据来自ALU输出	01:GRF写入端数据来自DM输出
WBreg[1:0]	10:GRF写入端数据来自pc8输出	无
RegWrite	无	把数据写入GRF中对应寄存器

控制信号	失效时作用（0）	有效时作用（1）
MemWrite	无	数据存储器DM写数据（输入）
PCSrc[2:0]	000:PC输入端选择PC+4	001:if(Zero),PC输入端选择b_type指令的目的地址
PCSrc[2:0]	010:PC输入端选择j_jump指令的目的地址	011:PC输入端选择j_reg指令的目的地址
PCSrc[2:0]	无	无
ExtOp[1:0]	00:sign_ext	01:zero_ext
ExtOp[1:0]	10:lui_ext	无
ALUOp[2:0]	000:与	001:或
ALUOp[2:0]	010:加法	011:减法
ALUOp[2:0]	100:保持不变	无

Op	000000	000000	001101	100011	101011	000100	001111	000011	000000
Func	100000	100010	无	无	无	无	无	无	001000
	add	sub	ori	lw	sw	beq	lui	jal	jr
RegDst[1:0]	01	01	00	00	x	x	00	10	x
ALUSrc[1:0]	00	00	01	01	01	x	01	01	x
WBreg[1:0]	00	00	00	01	x	x	00	10	x
RegWrite	1	1	1	1	0	0	1	1	0
MemWrite	0	0	0	0	1	0	0	0	0
PCSrc[2:0]	000	000	000	000	000	001	000	010	011
ExtOp[1:0]	x	x	01	00	00	x	10	x	x
ALUOp[2:0]	010	011	001	010	010	x	100	100	x

指令集分类

类型	指令	说明
cal_r	add, sub	R型计算类指令
ori	ori	I型计算类指令-E级产生结果
lui	lui	I型计算类指令-D级产生结果
ld	lw	load取数

类型	指令	说明
st	sw	store存数
b_type	beq	B型分支跳转
jal	jal	跳转并链接
jr	jr	跳转至寄存器

T_{use}：表示数据到了 D 级之后还需要多少个周期要使用，每个指令的T_{use}是固定不变的（指令进入IF/ID寄存器后，其后的某个功能部件再经过多少cycle就必须要使用相应的寄存器值）
T_{new}：表示数据还有多长时间产生，会随着数据的流水动态减少（位于ID/EX及其后各流水线的指令，再经过多少个时钟周期，能够产生要写入寄存器的结果）
会产生结果的指令：cal_r类，cal_i类，ld类———可充当供给方，考察其T_{new}

T_{use}表

指令类型	源寄存器	T_{use}
cal_r	rs/rt	1
ori	rs	1
ld	rs	1
st	rs	1
st	rt	2
b_type	rs/rt	0
jal	无	无
jr	rs	0

T_{new}表

指令类型	E_T_{new}(ID/EX)	M_T_{new}(EX/MEM)	W_T_{new}(MEM/WB)
cal_r	1/rd	0/rd	0/rd
ori	1/rt	0/rt	0/rt
lui	0/rt	0/rt	0/rt
ld	2/rt	1/rt	0/rt
jal	0/$31	0/$31	0/$31

测试方案

文本文档

思考题

在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

如图所示，直接利用 NPCOp 控制信号维护下一周期的 PC 值，其中有四种可能性即 branch 型， jr/jalr 型， j/jal 型和 PC+4 型

对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8 ？
需要考虑延迟槽，在跳转指令后面后面有 nop 或者一条数据无关的指令。

数据冒险的分析
为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由 ALU 或者 DM 等部件来提供数据?

如果从非流水线寄存器部件转发，那么某一级的总延迟就会增加，从而根据木桶效应，时钟周期就会增加，总效率反而降低，得不偿失。

AT 法处理流水线数据冒险
“转发（旁路）机制的构造” 中的 Thinking 1-4；

如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。

计算过程或存储过程中会用到还未更改过的寄存器值，从而出错。例如：

ori 1,0, 1
nop
nop
nop
nop
lw $1, 0(0)$
nop
sw $1, 4(0)$
这时，当指令 sw 达到 M 级时， lw 已经执行完毕，不会转发，但是我们有没有保留 E 级已经转发过的数据，这样 sw 指令就会把 1 存到 DM 中。

我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢?

GPR 采用内部转发机制相当于 MW 流水线寄存器的值直接实时反馈到 GPR 的输出端，从而当前处于 D 级的指令可以直接用到对应寄存器的值，即 W 级到 D 级的转发。

如果不采用内部转发机制，需要额外建立从 MW 流水线寄存器转发到 D 级的数据通路。

为什么 0 号寄存器需要特殊处理?

因为指令可以对 0 号寄存器赋值，只是不会造成实际作用，但是转发过程中如果不特判就默认 0 号寄存器的值被更改了，从而造成错误。

什么是 “最新产生的数据”?

根据指令的执行顺序，越后执行的指令更改的寄存器的值越新，按照 DE、EM、MW 的顺序，越靠前所转发出的信息越新，因此优先级更高。

在 AT 方法讨论转发条件的时候，只提到了 “供给者需求者的 A 相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 WE 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了用且仅用 A 和 T 完成转发，在翻译出 A 的时候，要结合 WE 做什么操作呢?

AT 法要求：只要当前位点的读取寄存器地址和某转发输入来源的写入寄存器地址相等且不为 0

那么既然是要写入的，WE 必然为 1，因此不用特判，如果不需要写入，我们零待写入地址 GRFA3 为 0，向 0 号寄存器里写入数据相当于不写

如果 WE 是 0，我们把 GRFA3 设为 5'b00000 即可