流水线CPU设计文档

> FU：Forwarding Unit
>
> SU：Stalling Unit
>
> CU：Control Unit
>
> 采用分布式译码

> 比较默认用比较器有符号比较。无符号比较0||data1 > 0||data2，直接用比较器的Unsigned比较即可。

支持指令集分类：

Load	Save	Cal_r	Cal_i	B_type	J_type	Shamt(5bit)	Other
lw	sw	addu	ori	beq	j	sll	nop
lh	sh	subu	slti	bne	jal		lui
lhu	sb	and		bgez	jr		
lb		or			jalr		
lbu		sllv					
		slt					
		div???					

添加指令步骤：新指令分析 -> 数据通路构建（判断是否需要添加新的数据来源） -> 控制器设计

新指令分析

阶段	任务
取指令（对NPC的影响）	
指令译码	
指令执行（ALU）	
存储器访问	
回写	

数据通路构建

阶段	模块	输入	Load	Save	Cal_r	Cal_i	B_type	Shamt	lui	Jal	Jr	J			综合
取指令	NPC	NPCSrc	PC4	PC4	PC4	PC4	Imm16	PC4	PC4	Imm26	RD1	Imm26			PC4 Imm26 RD1
	PC	/	/	/	/	/	/	/	/	/	/	/			/
	IM	/	/	/	/	/	/	/	/	/	/	/			/
译码	SPLT	/	/	/	/	/	/	/	/	/	/	/			/
	EXT	Imm16	Imm16	Imm16	/	Imm16	Imm16	/	Imm16	/	/	/			Imm16
	GRF	A1	rs	rs	rs	rs	rs	/	/	/	rs	/			rs
		A2	/	rt	rt	/	rt	rt	/	/	/	/			rt
执行	ALU	A	RD1	RD1	RD1	RD1	RD1	/	/	/	/	/			RD1
		B	Ext32	Ext32	RD2	Ext32	RD2	RD2	Ext32	/	/	/			RD2 Ext32
		Shamt	/	/	/	/	/	shamt	/	/	/	/			shamt
读存储器	DM	Address	ALUResult	ALUResult	/	/	/	/	/	/	/	/			ALUResult
		WD	/	RD2	/	/	/	/	/	/	/	/			RD2
回写	RF	A3	rt	/	rd	rt	/	rd	rt	ra	/	/			rt rd ra
		WD	RD	/	ALUResult	ALUResult	/	ALUResult	ALUResult	PC8	/	/			RD ALUResult PC8

控制器设计

指令	ALUControl	NPCSel	WDSel	A3Sel	ALUSrcBSel	EXTOp	DMWr	!RFWr	DataType
addu	0000	000	00	00	00	/	0	0	000
subu	0001	000	00	00	00	/	0	0	000
ori	0010	000	00	01	01	0	/	0	000
lw	0000	000	01	01	01	1	/	0	000
sw	0000	000	/	/	01	1	1	1	000
beq	0001	000/001	/	/	/	/	0	1	000
jal	/	010	10	10	/	/	0	0	000
lui	0100	000	00	01	01	0	0	0	000
and	0011	000	00	00	00	/	0	0	000
or	0010	000	00	00	00	/	0	0	000
jr	/	011	/	/	/	/	/	1	000
sll	0101	000	00	00	00	/	/	0	000
sllv	0110	000	00	00	00	/	/	0	000
lh	0000	000	01	01	01	1	/	0	001
lhu	0000	000	01	01	01	1	/	0	010
lb	0000	000	01	01	01	1	/	0	011
lbu	0000	000	01	01	01	1	/	0	100
sh	0000	000	/	/	01	1	1	1	001
sb	0000	000	/	/	01	1	1	1	010
j	/	010	/	/	/	/	/	1	000
jalr	/	011	10	00	/	/	/	0	000
bne	0001	000/001	/	/	/	/	/	1	000
bgez	0000	000/001	/	/	/	/	/	1	000
slt	0111	000	00	00	00	/	/	0	000
slti	0111	000	00	01	01	1	/	0	000
div									
nop	/	000	/	/	/	/	/	1	000

冲突处理设计

tuse：还要几个周期要使用寄存器中的数据。

tnew：还要几个周期**产生**（要求是后一级的流水寄存器产生）要写入的数据。

阶段	指令类型	S1	Tuse1	S2	Tuse2	Out	Tnew
D	Save	base(Rs)	1	Rt	2	/	/
	Load	Rs	1	/	/	Rt	3
	Cal_r	Rs	1	Rt	1	Rd	2
	Cal_i	Rs	1	/	/	Rt	2
	Lui	/	/	/	/	Rt	2（ALU计算）
	B_type	Rs	0（Cmp在D级）	Rt	0	/	/
	J	/	/	/	/	/	/
	Jal	/	/	/	/	￥31	1
	Jr	Rs	0	/	/	/	/
	Nop	/	/	/	/	/	/
E	Save	base(Rs)	0	Rt	1	/	/
	Load	Rs	0	/	/	Rt	2
	Cal_r	Rs	0	Rt	0	Rd	1
	Cal_i	Rs	0	/	/	Rt	1
	Lui	/	/	/	/	Rt	1（ALU计算）
	B_type	/	/	/	/	/	/
	J	/	/	/	/	/	/
	Jal	/	/	/	/	￥31	0
	Jr	/	/	/	/	/	/
	Nop	/	/	/	/	/	/
M	Save	/	/	Rt	0	/	/

阶段	指令类型	S1	Tuse1	S2	Tuse2	Out	Tnew
	Load	/	/	/	/	Rt	1
	Cal_r	/	/	/	/	Rd	0
	Cal_i	/	/	/	/	Rt	0
	Lui	/	/	/	/	Rt	0（ALU计算）
	B_type	/	/	/	/	/	/
	J	/	/	/	/	/	/
	Jal	/	/	/	/	￥31	0
	Jr	/	/	/	/	/	/
	Nop	/	/	/	/	/	/
W	Save	/	/	Rt	0	/	/
	Load	/	/	/	/	Rt	0
	Cal_r	/	/	/	/	Rd	0
	Cal_i	/	/	/	/	Rt	0
	Lui	/	/	/	/	Rt	0（ALU计算）
	B_type	/	/	/	/	/	/
	J	/	/	/	/	/	/
	Jal	/	/	/	/	￥31	0
	Jr	/	/	/	/	/	/
	Nop	/	/	/	/	/	/

功能模块

模块	功能
NPC: Next Program Counter	计算下一条指令地址
PC: Program Counter	存储当前指令地址
IM: Instruction Memory	存储所有指令
SPLT: Split	指令分线器
EXT: Extend	立即数扩展
GRF(RF): General Register File	32个寄存器
ALU: Arithmetic logic unit	算数逻辑单元
DM: Data Memory	存储所有数据
CU: Control Unit	计算控制信号

二、功能模块定义

NPC

端口定义

端口	输入/输出	位宽	描述
PC	I	32	当前指令地址
Imm26	I	26	当前指令的26位立即数
RA	I	32	当前指令的RS
NPCSel	I	3	NPC计算方式选择信号 00：PC+4 01：Imm26<<2 + PC + 4 10：PC[31:28]||Imm26||00 11：RA
PC4	O	32	当前指令地址+4
NPC	O	32	下一条指令地址

功能定义

编号	名称	描述
1	地址计算	根据选择信号NPCSel，计算对应下一条指令的地址NPC
2	地址计算	计算PC+4，为后续组件提供输入数据

PC

端口定义

端口	输入/输出	位宽	描述
DI	I	32	下一条指令地址
clk	I	1	时钟信号
reset	I	1	异步复位信号
DO	O	32	当前指令地址

功能定义

编号	名称	描述
1	复位	reset有效时，PC异步置为0x00000000
2	地址存储	保存并输出当前指令的地址DO

IM

端口定义

端口	输入/输出	位宽	描述
Address	I	32	当前指令地址
Data	O	32	输出当前地址的指令

功能定义

编号	名称	描述
1	指令存储	保存并输入Address对应的指令

SPLT

端口定义

端口	输入/输出	位宽	描述
Instruction	I	32	当前待拆分的指令
Op:31_26	O	6	操作码
Rs:25_21	O	5	寄存器1地址
Rt:20_16	O	5	寄存器2地址
Rd:15_11	O	5	寄存器3地址
Shamt:10_6	O	5	偏移量
Func:5_0	O	6	函数码
Imm:15_0	O	16	16位立即数
Imm:25_0	O	26	26位立即数

功能定义

编号	名称	描述
1	分线	将指令拆分

EXT

端口定义

端口	输入/输出	位宽	描述
Imm16	I	16	待扩展的16位立即数
Signed?	I	1	符号扩展类型选择 0：无符号扩展 1：符号扩展
Ext32	O	32	扩展后的16位立即数

功能定义

编号	名称	描述
1	无符号扩展	当Signed?为0时，将Imm16无符号扩展输出
2	符号扩展	当Signed?为1时，将Imm16无符号扩展输出

GRF

端口定义

端口	输入/输出	位宽	描述
A1	I	5	指定32个寄存器中的一个，将其中存储的数据读出到RD1
A2	I	5	指定32个寄存器中的一个，将其中存储的数据读出到RD2
A3	I	5	指定32个寄存器中的一个，作为WD的写入地址
WD	I	32	32位写入数据
we	I	1	写使能信号 0：禁止向GRF中写入数据 1：允许向GRF中写入数据
rst	I	1	异步复位信号，将32个寄存器中全部清零 0：无效 1：复位
RD1	O	32	输出A1指定的寄存器的32位数据
RD2	O	32	输出A2指定的寄存器的32位数据

功能定义

编号	名称	描述
1	异步复位	reset为1时，将所有寄存器清零
2	读数据	将A1和A2地址对应的寄存器的值分别通过RD1和RD2读出
3	写数据	当we为1且时钟上升沿来临时，将WD写入到A3对应的寄存器内部

ALU

端口定义

端口	输入/输出	位宽	描述
SrcA	I	32	参与运算的第一个数
SrcB	I	32	参与运算的第二个数
ALUControl	I	4	决定ALU做何种操作 0000：无符号加 0001：无符号减 0010：按位或 0011：按位与 0100：将SrcB左移16位 0101：将SrcB左移Shamt位
Shamt	I	5	偏移量

端口	输入/输出	位宽	描述
ALUOp	I	3	运算时，两操作数是否相等
ALUResult	O	32	运算后结果

功能定义

编号	名称	描述
1	无符号加	ALUResult = SrcA + SrcB
2	无符号减	ALUResult = SrcA - SrcB
3	按位或	ALUResult = SrcA | SrcB
4	按位与	ALUResult = SrcA & SrcB
5	将SrcB左移16位	ALUResult = SrcB << 16
6	将SrcB左移Shamt位	ALUResult = SrcB << Shamt

DM

端口定义

端口	输入/输出	位宽	描述
clk	I	1	时钟信号
Wr	I	1	写使能信号 0：禁止向DM中写入数据 1：允许向DM中写入数据
reset	I	1	异步复位信号 0：无效 1：复位
Address	I	32	读取或写入信号地址
WD	I	32	32位写入数据
RD	O	32	32位读出数据

功能定义

编号	名称	描述
1	异步复位	当reset为1时，DM中所有数据清零
2	写入数据	当Wr有效时，时钟上升沿来临时，WD中数据写入A对应的DM地址Address中
3	读出数据	RD始终读出Address对应的DM地址中的值

CU

端口定义

端口	输入/输出	位宽	描述
Opcode	I	6	6位控制信号
Func	I	6	6位控制信号
Equal	I	1	两操作数是否相等
RD1	I	32	寄存器读出结果1
NPCSel	O	3	NPC的选择信号
EXTOp	O	1	EXT的选择信号
ALUControl	O	4	ALU的控制信号
DMWr	O	1	DM写使能信号

端口	输入/输出	位宽	描述
ALUSrcBSel	O	1	SrcB的选择信号
RFWr	O	1	GRF写使能信号
WDSel	O	2	GRF的WD的选择信号
A3Sel	O	2	GRF的A3的选择信号
Data_type	O	3	数据类型

功能定义

编号	名称	描述
1	控制信号生成	判断指令类型并产生对应的控制信号

MW

端口定义

端口	输入/输出	位宽	描述
Data_type	I	3	写入/读出数据类型
Data	I	32	原始写入数据
Old_Data	I	32	待写入位置原数据
Address	I	32	写入地址
Out_Data	O	32	写入数据
Out_Addr	O	32	写入地址

功能定义

编号	名称	描述
1	写入数据处理	根据写入指令对写入数据预处理

MR

端口定义

端口	输入/输出	位宽	描述
Data_type	I	3	写入/读出数据类型
Data	I	32	原始读出数据
Address	I	32	读出地址
RD	O	32	读出数据

功能定义

编号	名称	描述
1	读出数据处理	根据读出指令对读出数据预处理

流水寄存器X_X_Reg

端口定义

端口	输入/输出	位宽	描述
xx_I	I	xx	前一级输出
RESET	I	1	重置信号
STALL	I	1	阻塞信号
xx_O	O	xx	后一级输入

功能定义

编号	名称	描述
1	信号存储传递	传递信号及结果。

三、测试方案

评测方案：python生成MIPS代码 + 自动对拍测试

- python生成MIPS代码：

  对每条指令进行覆盖率测试，覆盖所有寄存器、边缘极值等。

```python
import random


for i in range(27):
    imm =  random.randint(0,1000)
    imm = imm * 2
    print("ori ${}, $0, {}" .format(i, imm))
    for _ in range(4):
        print("nop")

for i in range(27):
    j = random.randint(0,27)
    k = random.randint(0, 27)
    print("addu ${}, ${}, ${}" .format(i, j, k))
    for _ in range(4):
        print("nop")

for i in range(27):
    j = random.randint(0,27)
    k = random.randint(0, 27)
    print("subu ${}, ${}, ${}" .format(i, j, k))
    for _ in range(4):
        print("nop")

for _ in range(32):
    i = random.randint(0,27)
    j = random.randint(0, 27)
    k = random.randint(0, 27)
    if _%3==2:
        print("or ${}, ${}, ${}".format(i, j, k))
    elif _%3==1:
        print("and ${}, ${}, ${}".format(i, j, k))
    else:
        print("sllv ${}, ${}, ${}".format(i, j, k))
    for _ in range(4):
        print("nop")

for _ in range(32):
    i = random.randint(0,27)
    j = random.randint(0, 27)
    imm = random.randint(0, 1000)
    print("ori ${}, ${}, {}".format(i, j, imm))
    for _ in range(4):
        print("nop")

for _ in range(32):
    i = random.randint(0,27)
    j = random.randint(0, 27)
    k = random.randint(0, 27)
    print("sll ${}, ${}, {}".format(i, j, k))
    for _ in range(4):
        print("nop")

for _ in range(32):
    i = random.randint(0,27)
```

```
    imm = random.randint(0,1000)
    print("lui ${} {}".format(i, imm))
    for _ in range(4):
        print("nop")
```

- 自动对拍测试：[uanu2002/BUAA-CO-CPU-Judge]

  调用Mars编译MIPS代码，并将结果载入值Logisim电路中的ROM，分别单步运行MIPS和Logisim，并对单步输出（GRF和DM操作、输出引脚）进行记录，运行结束后对输出进行对拍。

四、思考题

1. 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。

   例如：

   ```
   loop:
   add $1, $0, $1
   beq $1, $2, loop
   ```

   在提前分支判断的情况下，由于 `beq` 中 $1 需要在 D 级就被使用，因此需要进行一此阻塞。

   但如果分支判断更晚，则无需进行此次阻塞。

2. 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？

   因为存在延迟槽，如果使用`PC+4`，则会重复执行jal后的指令。

3. 我们要求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？

   因为在考虑数据冲突时，AT法中Tnew的数据产生是以新数据进入（流水）寄存器保存为“算出来”标志。

   如果直接使用功能部件的计算结果会导致电路更加复杂。

4. 我们为什么要使用 GRF 内部转发？该如何实现？

   原因：避免额外的转发逻辑。

   实现：在GRF内部增加判断和数据通路，如果有写使能、输入A1不为0且A1等于A3，则将写入数据直接读出，否则读出A1寄存器。

5. 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？

   可能来源于 D，E，M 级，从 E，M，W 级中提供数据

   转发通路:

   W -> E,M,D(或内部转发)

   M -> D,E

   E -> D

6. 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。

   - 可能需要添加ALU的操作。
   - 可能需要增加数据输入来源，例如RegData、RegAddr、ALUSrcA/B、DMData、DMAddr等。
   - 可能需要增加新的预处理组件，例如在某阶段进行比较，根据结果进行操作等。

7. 简要描述你的译码器架构，并思考该架构的优势以及不足。

本CPU采用分布式译码。

- 优势：每级单独实例化控制器，复用度高，便于添加新指令。高内聚，低耦合。
- 不足：会出现多余接口，造成浪费。

8. [P5 选做] 请详细描述你的测试方案及测试数据构造策略。
   - 不考虑冲突冒险，使用三中的测试方案。
   - 考虑冲突冒险，首先使用三中的测试方案作基础测试，保证随机情况下的正确性。

     先对指令集进行分类：`Load， Save， Cal_r， Cal_i， B_type， J_type， Shamt(5bit)， Other`。

     然后根据遍历每一种可能出现冲突冒险的情况，并考虑不同位置进行覆盖测试。