

3月21日

3. RISC-V 汇编中存在许多伪指令，它们一般是具有特殊操作数的基本指令或指令组合。

请写出与以下伪指令等价的基本指令或指令组合。

- 1) nop
- 2) ret
- 3) call offset
- 4) mv rd,rs
- 5) rdcycle rd
- 6) sext.w rd,rs

- 1) add x0,x0,0 无操作
- 2) jalr x0,x1,0 从子过程返回
- 3) auipc x1, offset[31:12]
jalr x1,x1,offset[11:0] 远程调用子过程
- 4) addi rd,rs,0 复制寄存器
- 5) csrrs rd,cycle,x0 读取周期计时器
- 6) addiw rd,rs,0 有符号扩展字

7. RISC-V 标准指令集并未为加法指令的溢出引入专用的标志位，因此通常需要额外的指令以检查加法溢出。

1) 考虑如下的指令序列：

add t0,t1,t2

bne t3,t4,overflow

若 t0 和 t1 都是有符号数，请在横线处填入正确的指令，使得当 t0 和 t1 的加法发生溢出时，控制流可以正确跳转到 overflow 位置。（请勿使用除 t0~t4 以外的任何寄存器）

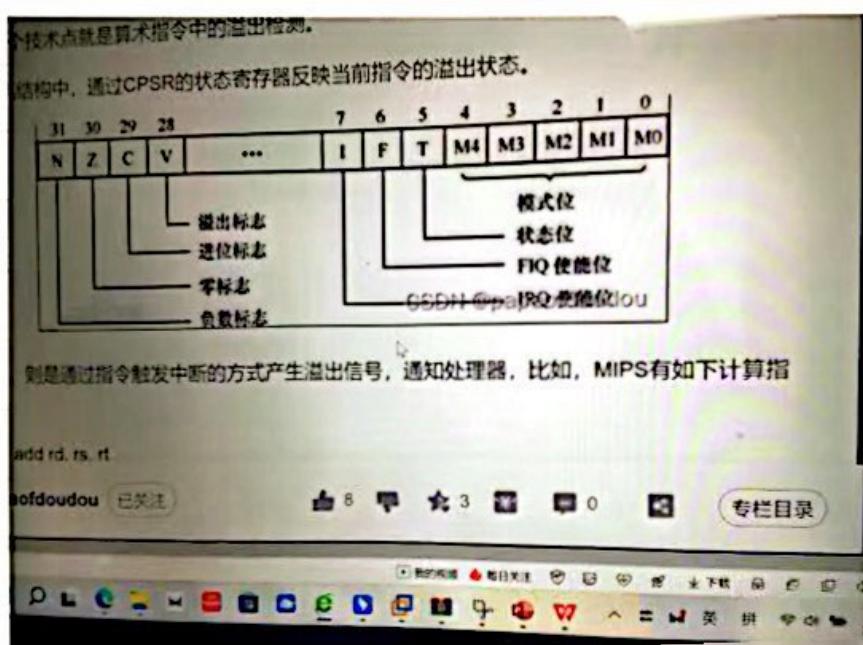
2) 当 t0 和 t1 都是无符号数时，请给出尽量简单的检测 add t0,t1,t2 指令加法是否溢出的指令序列。

3) 调研其他指令集架构（如 x86、ARM 等）是如何检测加法溢出的。

> slti t3,t2,0 (if $t_2 < 0, t_3 = 1$)
slt, t4,t0,t1 (if $t_0 < t_1, t_4 = 1$)

2) addu t0,t1,t2
bltu t0,t1,overflow

3) ARM：通过 CPSR 的状态寄存器反映
当前指令的溢出状态



8. 阅读 RISC-V 规范以了解 RISC-V 对除数为 0 的除法指令的处理方法，回答以下问题。

- 1) 对整型除法，填写下表。整型除法中除数为 0 是否会引起 RISC-V 抛出异常？试分析为什么采取这样的设计。

指令	rs1	rs2	Op=DIVU 时 rd 值	Op=REMU 时 rd 值	Op=DIV 时 rd 值	Op=REM 时 rd 值
Op rd,rs1,rs2	x	0				

- 2) 对浮点除法，除数为 0 将会引起 fcsr 控制寄存器中的相关标志位被置位。下图给出了 fcsr 的构成，请说明 flags 的各位分别代表什么含义。flags 被置位是否会使处理器陷入系统调用？

31	Reserved	8 7	Rounding Mode (frm)	5 4	3	2	1	0
24		3		NV	DZ	OF	UF	NX

- 3) 调研其他指令集架构（如 x86、ARM 等）是如何处理除数为 0 的。

1)

x_{LEN-1} , x_{-1}, x 不会产生异常
设计原因：因此微体系结构可以将这些融合为一个单一的除法操作，而不是执行两个分开的除法。若触发异常，这个异常在绝大多数执行环境里会屏蔽自身。当语言要求一个除以 0 异常必须导致一个立即的控制流改变时，只需在一个除法操作时增加一条分支指令即可。

2)

NV：非法操作，DZ：除以 0，OF：上溢，UF：下溢，NX：不精确，会陷入系统调用

3)

当CCR 寄存器的 DIV-0-TPR 位被配置为 1，即使除以 0 操作也会发生异常，只有当该位置 1 时，当发生除以 0 操作时才能发生异常事件，并产生相应中断；结合 Cortex-14 内核手册，当发生除以 0 操作是否触发异常事件是可以配置的，到除以 0 之后的简称为 DIV-0-TPR

12. 写出以下程序在 RISC-V 中应当处于的特权等级。

- 1) Linux Kernel
- 2) BootROM
- 3) BootLoader
- 4) USB Driver
- 5) vim

(1) Supervisor

(2) Machine

(3) Machine

(4) Supervisor

(5) User

13. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设 A 和 B 的起始地址存放于寄存器 t0 和 t1, C 的地址存放于寄存器 t2。

```
int vecMul(int *A, int *B, C){  
    for(int i = 0; i < 100; ++i){  
        A[i] = B[i] * C;
```

```
    }  
    return A[0];
```

```
di sp, sp, -32
```

```
v ra, 24(sp) # return address; add t7, t1, t3 # t7 = t1 + t3
```

```
d t3, x0, x # t3 = i
```

```
li t4, x0, 100 # t4 = 100
```

```
d t5, x0, x0. # t5 = temp
```

```
bge t3, t4, exit
```

```
sw t3, -20($0)
```

```
shl t3, t3, 2 # i * 4
```

```
mul t5, t2, (t1 + t3)
```

```
mv ($t0+t3), t5
```

```
lw t3, -20($0)
```

```
add t3, t3, 1
```

```
j loop
```

```
exit: mv a0, ($t0+0)  
lw ra, 24(sp)  
addi sp, sp, 32
```

add t3, x0, x # t3 = i

addi t4, x0, 100 # t4 = 100

add t5, x0, x # t5 = temp

loop: bge t3, t4, end

sw t3, -20(\$0)

shl t3, t3, 2 # t3 = i * 4

add t6, t0, t3 # t6 = t0 + t3

mul t5, t6, t2

add t7, t1, t3 # t7 = t1 + t3

mv t7, t5

lw t3, -20(\$0)

addi t3, t3, 1

j loop

end:

mv a0, (\$t0+0)

lw ra, 24(sp)

addi sp, sp, 32

14. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设 a、b 和 c 分别对应寄存器 a0、a1 和 a2。

```
int a, b, c;
if(a > b){
    c = a + b;
}
else{
    c = a - b;
}
```

add a0, x0, x6
add a1, x0, x6
add a2, x0, x6

bge a0, a1, Part :

else :

sub a2, a0, a1 # $a_0 < a_1, a_2 = a_0 - a_1$

j end

Part :

add a2, a0, a1 (这条紧跟在 end 故不需 j)
$a_0 > a_1, a_2 = a_0 + a_1$

end :

15. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设指针 p 已经通过程序 `int *p = (int *) malloc(4 * sizeof(int))` 得到，且 p 存放于 t0 中，a 存放于 t1 中。

```
p[0] = p;  
int a = 3;  
p[1] = a;  
p[a] = a;
```

|w t0, o(t0) # p[0]=p, 存放着指针|

addi t1, x0, 3 # a=t1=3

addi t2, t0, 4 }
|w t2, o(t2) # } p[1]=a
mv t2, t1 }

addi t3, t0, 12 }
|w t3, o(t3) # } p[a]=a
mv t3, t1 }

16. 写出实现以下 C 程序的 32 位 RISC-V 汇编代码。假设指针 a 和 b 分别存放于 t0 和 t1 中。

```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
    return;  
}
```

addi SP, SP, -32

sw ra, 24(SP)

sd so, 16(SP)

addi so, SP, 32

addi tp, X0, X0 - # $t_3 = \text{temp} = 0$

lw t2, 0(t0) # $t_2 = a$

lw t3, 0(t1) # $t_3 = b$

mv t4, t2 # $t_4 = a$

mv t2, t3 # 存放 a 的寄存器被 b 覆盖, 此时 $t_2 = b$

mv t3, tp # 存放 b 的寄存器被 a 覆盖, 此时 $t_3 = a$

sw t2, 0(t0)

sw t3, 0(t1)

lw ra, 24(SP)

SP, SP, 32

j ra

17. 解释以下 RISC-V 汇编代码实现的功能。

addi a0,x0,0	$a_0 = 0$
addi a1,x0,1	$a_1 = 1$
addi a2,x0,30	$a_2 = 30$
loop: beq a0,a2,done	
slli a1,a1,1	① $a_1 = 2^1, a_0 = 1$
addi a0,a0,1	② $a_1 = 2^2, a_0 = 2$
j loop	③ $a_1 = 2^3, a_0 = 3$
done: # exit code	④ $a_1 = 2^4, a_0 = 4$

计算 2^{29} 次方的值