241880334 闵振昌 第五章

3

CALL 指令是调用子程序指令，它可以保存返回地址(PC+2，因为CALL占2个字)，并且转移到子程序，把指令中给出的子程序起始地址送入PC。

执行CALL指令的过程： 首先PC->MAR，然后根据PC地址从MDR读出指令存入IR，先读了CALL指令的第一个字也就是操作码。 然后PC加1，这个时候ALU是a+1控制信号，输出的F存入PC

然后同样方式取出CALL指令的第二个字，存入Y，然后ALU是a+1控制信号，输出的F存入SP 然后把Y给A，ALU是MOVa控制信号，输出F=A，再把F存入PC就可以开始执行子程序

5

- RegWr = 0，所有需要写回寄存器的指令都不能正确执行，R-type，lw，addi等需写寄存器的I型指令。
- RegDst = 0，写寄存器号总选 rt（而不是 rd），因此 R-type 指令会错误
- ALUSrc = 0，ALU 第二操作数总从寄存器读，不能选立即数，I 型立即数类指令出错
- Branch = 0，分支比较结果不会使 PC 条件转移，所以 beq 分支指令出错。
- MemWr = 0，存内存写失效，sw 无法把数据写入内存
- ExtOp = 0，使用符号扩展的指令受影响，地址计算/带符号立即数的指令会错

- R-type = 0，导致 R-type 指令没有被正确识别/执行，所有 R-type 算术逻辑指令不能正确执行。
- MemtoReg = 0，写回总从 ALU（而不是内存），lw 不能把内存读出的数据写回寄存器

6

- RegWr = 1，所有周期都会写寄存器，那些本不该写寄存器的指令会错误，比如sw，beq等
- RegDst = 1，写寄存器号总选 rd（而不是 rt），因此 I-type 指令需要写回的会错误，比如lw和addi
- ALUSrc = 1，总使用立即数作为 ALU 第二操作数，R-type 指令会错。
- Branch = 1，总把分支信号置位，会在不该跳时跳转，几乎所有指令都会出错
- MemWr = 1，总写内存，会在许多不该写内存的周期写内存
- ExtOp = 1，对本该零扩展的立即数指令（如 andi/ori）会错误
- R-type = 1，总把操作当作 R-type，I 型指令的译码/执行会错
- MemtoReg = 1，总从内存写回寄存器，add/sub/addi等会把内存数据写入而不是 ALU 结果。

7

（1）

```
xor rs, rs, rt
xor rt, rs, rt
xor rs, rs, rt
```

（2） 设原来每条指令时间为 1（单位），仿指令实现 swap 需要 3 条指令，所以平均每个 swap 相当于多用 2 条指令。 设程序原有 N 条指令，swap 占比 p 1+2p=1.1 p=0.05=5%

因此swap指令占程序指令大于等于5%的时候，用硬件实现才划算

8

- PCWr = 0，PC 不能更新，CPU 停在第一条指令，没有指令能够执行
- MemtoReg = 0，写回阶段只能从 ALU，而不能从内存，lw 指令错误
- IRWr = 0，指令寄存器不能更新，所有指令出错
- RegWr = 0，无法写回寄存器，R-type和lw出错
- BrWr = 0，分支目标地址无法写入 PC，beq和j型指令出错
- MemWr = 0，内存写禁止，sw出错
- PCWrCond = 0，分支条件成立时 PC 也不会更新，beq出错
- R-type = 0，R 型指令无法正确识别，R-type出错

9

- PCWr = 1，每个周期 PC 都更新，可能会跳过某些指令
- MemtoReg = 1，写回寄存器总是从内存，所有除了lw的指令出错
- IRWr = 1，IR 每周期都更新，指令可能被覆盖
- RegWr = 1，每周期都写寄存器，寄存器值被破坏
- BrWr = 1，每次都分支，指令错乱
- MemWr = 1，每周期写内存，内存写错误
- PCWrCond = 1，条件分支总成立，意外改变PC
- R-type = 1，所有指令被当成 R 型，非R-type指令出错

10

```
    beq   $t3, $zero, all_equal   # 如果长度为 0 -> 直接认为所有数据“相等”
loop:
    lw    $t4, 0($t1)             # 读 A 当前元素
    lw    $t5, 0($t2)             # 读 B 当前元素
    bne   $t4, $t5, not_equal     # 若两个元素不等 -> 退出，$t1/$t2 已是不等元素的地址
    addi  $t1, $t1, 4             # 指向下一个元素地址
    addi  $t2, $t2, 4
    addi  $t3, $t3, -1            # 剩余元素数减 1
    bne   $t3, $zero, loop        # 还没比完，继续
    # 如果走到这里，说明所有元素都相等
all_equal:
    addi  $t1, $zero, 0           # 把 0 写入 $t1
    j     done
not_equal:
    # $t1 和 $t2 已分别含有第一次出现不相等元素的地址
done:
```

每比较一个元素并且继续（该元素相等）需要28个周期

13

（1）

- 除数为 0：执行阶段
- 算术溢出：执行阶段
- 无效指令操作码：译码/取指阶段
- 无效指令地址：取指阶段
- 无效数据地址：访存阶段
- 缺页：取指或者访存阶段
- 访问越权：取指或者访存阶段
- 外部中断：通常在指令边界

（2） 保存异常原因，阻止未提交的写，将 PC 设为异常向量，由操作系统的异常处理程序完成上下文保存，处理完后由返回指令恢复 PC 与状态