%rsp，E 和 M 两个写端口会用到同一个地址，但是写入的数据不同。为了解决这个冲突，必须对两个写端口设立一个优先级，这样一来，当同一个周期内两个写端口都试图对一个寄存器进行写时，只有较高优先级端口上的写才会发生。那么要实现练习题 4.8 中确定的行为，哪个端口该具有较高的优先级呢？

3. 执行阶段

执行阶段包括算术/逻辑单元(ALU)。这个单元根据 alufun 信号的设置，对输入 aluA 和 aluB 执行 ADD、SUBTRACT、AND 或 EXCLUSIVE-OR 运算。如图 4-29 所示，这些数据和控制信号是由三个控制块产生的。ALU 的输出就是 valE 信号。

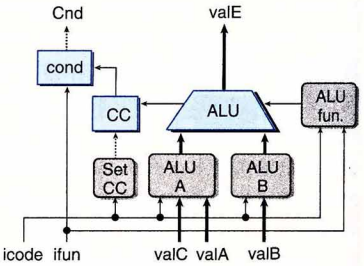

图 4-29　SEQ 执行阶段。ALU 要么为整数运算指令执行操作，要么作为加法器。根据 ALU 的值，设置条件码寄存器。检测条件码的值，判断是否该选择分支

在图 4-18～图 4-21 中，执行阶段的第一步就是每条指令的 ALU 计算。列出的操作数 aluB 在前面，后面是 aluA，这样是为了保证 subq 指令是 valB 减去 valA。可以看到，根据指令的类型，aluA 的值可以是 valA、valC，或者是－8 或＋8。因此我们可以用下面的方式来表达产生 aluA 的控制块的行为：

```
word aluA = [
        icode in { IRRMOVQ, IOPQ } : valA;
        icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ } : valC;
        icode in { ICALL, IPUSHQ } : -8;
        icode in { IRET, IPOPQ } : 8;
        # Other instructions don't need ALU
];
```

练习题 4.23　根据图 4-18～图 4-21 中执行阶段第一步的第一个操作数，写出 SEQ 中信号 aluB 的 HCL 描述。

观察 ALU 在执行阶段执行的操作，可以看到它通常作为加法器来使用。不过，对于 OPq 指令，我们希望它使用指令 ifun 字段中编码的操作。因此，可以将 ALU 控制的 HCL 描述写成：

```
word alufun = [
        icode == IOPQ : ifun;
        1 : ALUADD;
];
```

执行阶段还包括条件码寄存器。每次运行时，ALU 都会产生三个与条件码相关的信号——零、符号和溢出。不过，我们只希望在执行 OPq 指令时才设置条件码。因此产生了一个信号 set_cc 来控制是否该更新条件码寄存器：

```
bool set_cc = icode in { IOPQ };
```

标号为“cond”的硬件单元会根据条件码和功能码来确定是否进行条件分支或者条件数据传送(图 4-3)。它产生信号 Cnd，用于设置条件传送的 dstE，也用在条件分支的下一个 PC 逻辑中。对于其他指令，取决于指令的功能码和条件码的设置，Cnd 信号可以被设置为 1 或者 0。但是控制逻辑会忽略它。我们省略这个单元的详细设计。