2. 译码和写回阶段

图 4-28 给出了 SEQ 中实现译码和写回阶段的逻辑的详细情况。把这两个阶段联系在一起是因为它们都要访问寄存器文件。

寄存器文件有四个端口。它支持同时进行两个读(在端口 A 和 B 上)和两个写(在端口 E 和 M 上)。每个端口都有一个地址连接和一个数据连接，地址连接是一个寄存器 ID，而数据连接是一组 64 根线路，既可以作为寄存器文件的输出字(对读端口来说)，也可以作为它的输入字(对写端口来说)。两个读端口的地址输入为 srcA 和 srcB，而两个写端口的地址输入为 dstE 和 dstM。如果某个地址端口上的值为特殊标识符 0xF(RNONE)，则表明不需要访问寄存器。

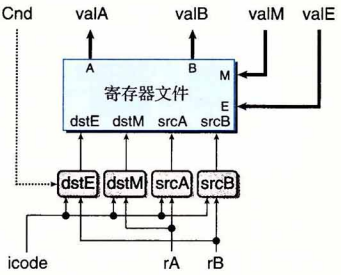

图 4-28　SEQ 的译码和写回阶段。指令字段译码，产生寄存器文件使用的四个地址(两个读和两个写)的寄存器标识符。从寄存器文件中读出的值成为信号 valA 和 valB。两个写回值 valE 和 valM 作为写操作的数据

根据指令代码 icode 以及寄存器指示值 rA 和 rB，可能还会根据执行阶段计算出的 Cnd 条件信号，图 4-28 底部的四个块产生出四个不同的寄存器文件的寄存器 ID。寄存器 ID srcA 表明应该读哪个寄存器以产生 valA。所需要的值依赖于指令类型，如图 4-18～图 4-21 中译码阶段第一行中所示。将所有这些条目都整合到一个计算中就得到下面的 srcA 的 HCL 描述(回想 RRSP 是%rsp 的寄存器 ID)：

```
word srcA = [
        icode in { IRRMOVQ, IRMMOVQ, IOPQ, IPUSHQ  } : rA;
        icode in { IPOPQ, IRET } : RRSP;
        1 : RNONE; # Don't need register
];
```

练习题 4.20　寄存器信号 srcB 表明应该读哪个寄存器以产生信号 valB。所需要的值如图 4-18～图 4-21 中译码阶段第二步所示。写出 srcB 的 HCL 代码。

寄存器 ID dstE 表明写端口 E 的目的寄存器，计算出来的值 valE 将放在那里。图 4-18～图 4-21写回阶段第一步表明了这一点。如果我们暂时忽略条件移动指令，综合所有不同指令的目的寄存器，就得到下面的 dstE 的 HCL 描述：

```
# WARNING: Conditional move not implemented correctly here
word dstE = [
        icode in { IRRMOVQ } : rB;
        icode in { IIRMOVQ, IOPQ} : rB;
        icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
        1 : RNONE;  # Don't write any register
];
```

我们查看执行阶段时，会重新审视这个信号，看看如何实现条件传送。

练习题 4.21　寄存器 ID dstM 表明写端口 M 的目的寄存器，从内存中读出来的值 valM 将放在那里，如图 4-18～图 4-21 中写回阶段第二步所示。写出 dstM 的 HCL 代码。

练习题 4.22　只有 popq 指令会同时用到寄存器文件的两个写端口。对于指令 popq