指令只是简单地经过各个阶段，除了要将 PC 加 1，不进行任何处理。halt 指令使得处理器状态被设置为 HLT，导致处理器停止运行。

1. 取指阶段

如图 4-27 所示，取指阶段包括指令内存硬件单元。以 PC 作为第一个字节(字节 0)的地址，这个单元一次从内存读出 10 个字节。第一个字节被解释成指令字节，(标号为“Split”的单元)分为两个 4 位的数。然后，标号为“icode”和“ifun”的控制逻辑块计算指令和功能码，或者使之等于从内存读出的值，或者当指令地址不合法时(由信号 imem_error 指明)，使这些值对应于 nop 指令。根据 icode 的值，我们可以计算三个一位的信号(用虚线表示)：

instr_valid：这个字节对应于一个合法的 Y86-64 指令吗？这个信号用来发现不合法的指令。

need_regids：这个指令包括一个寄存器指示符字节吗？

need_valC：这个指令包括一个常数字吗？

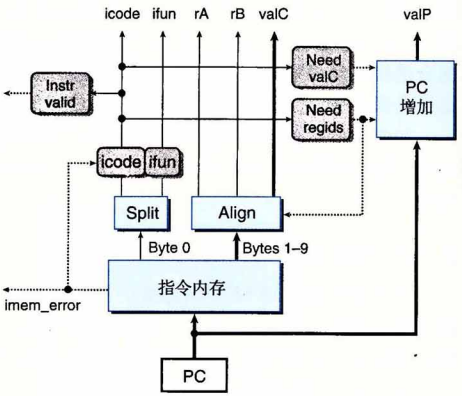

图 4-27　SEQ 的取指阶段。以 PC 作为起始地址，从指令内存中读出 10 个字节。根据这些字节，我们产生出各个指令字段。PC 增加模块计算信号 valP

(当指令地址越界时会产生的)信号 instr_valid 和 imem_error 在访存阶段被用来产生状态码。

让我们再来看一个例子，need_regids 的 HCL 描述只是确定了 icode 的值是否为一条带有寄存器指示值字节的指令。

```
bool need_regids =
        icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ,
                   IIRMOVQ, IRMMOVQ, IMRMOVQ };
```

练习题 4.19　写出 SEQ 实现中信号 need_valC 的 HCL 代码。

如图 4-27 所示，从指令内存中读出的剩下 9 个字节是寄存器指示符字节和常数字的组合编码。标号为“Align”的硬件单元会处理这些字节，将它们放入寄存器字段和常数字中。当被计算出的信号 need_regids 为 1 时，字节 1 被分开装入寄存器指示符 rA 和 rB 中。否则，这两个字段会被设为 0xF(RNONE)，表明这条指令没有指明寄存器。回想一下(图 4-2)，任何只有一个寄存器操作数的指令，寄存器指示值字节的另一个字段都设为 0xF(RNONE)。因此，可以将信号 rA 和 rB 看成，要么放着我们想要访问的寄存器，要么表明不需要访问任何寄存器。这个标号为“Align”的单元还产生常数字 valC。根据信号 need_regids 的值，要么根据字节 1～8 来产生 valC，要么根据字节 2～9 来产生。

PC 增加器硬件单元根据当前的 PC 以及两个信号 need_regids 和 need_valC 的值，产生信号 valP。对于 PC 值 p、need_regids 值 r 以及 need_valC 值 i，增加器产生值 $p+1+r+8i$。