


的步骤,使之也能处理6个条件传送指令。看看jXX指令的实现(图4-21)是如何处理条件行为的,可能会有所帮助。

阶段	cmovXX rA, rB
取指	$\text{icode, ifun} \leftarrow M_1[\text{PC}]$ $\text{rA, rB} \leftarrow M_1[\text{PC}+1]$ $\text{valP} \leftarrow \text{PC}+2$
译码	$\text{valA} \leftarrow R[\text{rA}]$
执行	$\text{valE} \leftarrow 0 + \text{valA}$
访存	
写回	$R[\text{rB}] \leftarrow \text{valE}$
更新 PC	$\text{PC} \leftarrow \text{valP}$

指令call和ret与指令pushq和popq类似,除了我们要将程序计数器的值入栈和出栈以外。对指令call,我们要将valP,也就是call指令后紧跟着的那条指令的地址,压入栈中。在更新PC阶段,将PC设为valC,也就是调用的目的地。对指令ret,在更新PC阶段,我们将valM,即从栈中取出的值,赋值给PC。

 **练习题 4.18** 填写下表的右边一栏,这个表描述的是图4-17中目标代码第9行call指令的处理情况:

阶段	通用	具体
	call Dest	call 0x041
取指	$\text{icode, ifun} \leftarrow M_1[\text{PC}]$ $\text{valC} \leftarrow M_8[\text{PC}+1]$ $\text{valP} \leftarrow \text{PC}+9$	
译码	$\text{valB} \leftarrow R[\%rsp]$	
执行	$\text{valE} \leftarrow \text{valB} + (-8)$	
访存	$M_8[\text{valE}] \leftarrow \text{valP}$	
写回	$R[\%rsp] \leftarrow \text{valE}$	
更新 PC	$\text{PC} \leftarrow \text{valC}$	

这条指令的执行会怎样改变寄存器、PC和内存呢?

我们创建了一个统一的框架,能处理所有不同类型的Y86-64指令。虽然指令的行为大不相同,但是我们可以将指令的处理组织成6个阶段。现在我们的任务是创建硬件设计来实现这些阶段,并把它们连接起来。

旁注 跟踪ret指令的执行

让我们来看看图4-17中目标代码的第13行ret指令的处理情况。指令的地址是0x041,只有一个字节的编码,0x90。前面的call指令将%rsp置为了120,并将返回地址0x040存放在内存地址120中。各个阶段的处理如下: