| 阶段 | 通用 | 具体 |
|---|---|---|
| | popq rA | popq %rax |
| 取指 | icode:ifun ← $M_1$[PC]<br>rA:rB ← $M_1$[PC+1]<br><br>valP ← PC+2 | icode:ifun ← $M_1$[0x02c]=b:0<br>rA:rB ← $M_1$[0x02d]=0:f<br><br>valP ← 0x02c+2=0x02e |
| 译码 | valA ← R[%rsp]<br>valB ← R[%rsp] | valA ← R[%rsp]=120<br>valB ← R[%rsp]=120 |
| 执行 | valE ← valB+8 | valE ← 120+8=128 |
| 访存 | valM ← $M_8$[valA] | valM ← $M_8$[120]=9 |
| 写回 | R[%rsp]← valE<br>R[rA]← valM | R[%rsp]← 128<br>R[%rsp]← 9 |
| 更新 PC | PC ← valP | PC ← 0x02e |

该指令将%rax 设为 9，将%rsp 设为 128，并将 PC 加 2。

4.15　沿着图 4-20 中列出的步骤，这里 rA 等于%rsp，我们可以看到，在访存阶段，指令会将 valA(即栈指针的原始值)存放到内存中，与我们在 x86-64 中发现的一样。

4.16　沿着图 4-20 中列出的步骤，这里 rA 等于%rsp，我们可以看到，两个写回操作都会更新%rsp。因为写 valM 的操作后发生，指令的最终效果会是将从内存中读出的值写入%rsp，就像在 x86-64 中看到的一样。

4.17　实现条件传送只需要对寄存器到寄存器的传送做很小的修改。我们简单地以条件测试的结果作为写回步骤的条件：

| 阶段 | cmovXX rA,rB |
|---|---|
| 取指 | icode:ifun ← $M_1$[PC]<br>rA:rB ← $M_1$[PC+1]<br>valP ← PC+2 |
| 译码 | valA ← R[rA] |
| 执行 | valE ← 0+valA<br>Cnd ← Cond(CC, ifun) |
| 访存 | |
| 写回 | if(Cnd)<br>R[rB]← valE |
| 更新 PC | PC ← valP |

4.18　我们可以看到这条指令位于地址 0x037，长度为 9 个字节。第一个字节值为 0x80，而后面 8 个字节是 0x0000000000000041 按字节反过来的形式，即调用的目标地址。popq 指令(第 7 行)将栈指针设为 128。

| 阶段 | 通用 | 具体 |
|---|---|---|
| | call Dest | call 0x041 |
| 取指 | icode:ifun ← $M_1$[PC]<br><br>valC ← $M_8$[PC+1]<br>valP ← PC+9 | icode:ifun ← $M_1$[0x037]=8:0<br><br>valC ← $M_8$[0x038]=0x041<br>valP ← 0x037+9=0x040 |
| 译码 | valB ← R[%rsp] | valB ← R[%rsp]=128 |
| 执行 | valE ← valB+ −8 | valE ← 128+ −8=120 |
| 访存 | $M_8$[valE]← valP | $M_8$[120]← 0x040 |
| 写回 | R[%rsp]← valE | R[%rsp]← 120 |
| 更新 PC | PC ← valC | PC ← 0x041 |