这条指令的效果就是将%rsp设为120，将0x040(返回地址)存放到该内存地址，并将PC设为0x041(调用的目标地址)。

4.19 练习题中所有的HCL代码都很简单明了，但是试着自己写会帮助你思考各个指令，以及如何处理它们。对于这个问题，我们只要看看Y86-64的指令集(图4-2)，确定哪些有常数字段。

```
bool need_valC =
        icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IJXX, ICALL };
```

4.20 这段代码类似于srcA的代码：

```
word srcB = [
        icode in { IOPQ, IRMMOVQ, IMRMOVQ  } : rB;
        icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
        1 : RNONE;  # Don't need register
];
```

4.21 这段代码类似于dstE的代码：

```
word dstM = [
        icode in { IMRMOVQ, IPOPQ } : rA;
        1 : RNONE;  # Don't write any register
];
```

4.22 像在练习题4.16中发现的那样，为了将从内存中读出的值存放到%rsp，我们想让通过M端口写的优先级高于通过E端口写。

4.23 这段代码类似于aluA的代码：

```
word aluB = [
        icode in { IRMMOVQ, IMRMOVQ, IOPQ, ICALL,
                   IPUSHQ, IRET, IPOPQ } : valB;
        icode in { IRRMOVQ, IIRMOVQ } : 0;
        # Other instructions don't need ALU
];
```

4.24 实现条件传送令人吃惊的简单：当条件不满足时，通过将目的寄存器设置为RNONE禁止写寄存器文件。

```
word dstE = [
        icode in { IRRMOVQ } && Cnd : rB;
        icode in { IIRMOVQ, IOPQ} : rB;
        icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
        1 : RNONE;  # Don't write any register
];
```

4.25 这段代码类似于mem_addr的代码：

```
word mem_data = [
        # Value from register
        icode in { IRMMOVQ, IPUSHQ } : valA;
        # Return PC
        icode == ICALL : valP;
        # Default: Don't write anything
];
```

4.26 这段代码类似于mem_read的代码：

```
bool mem_write = icode in { IRMMOVQ, IPUSHQ, ICALL };
```

4.27 计算Stat字段需要从几个阶段收集状态信息：

```
## Determine instruction status
word Stat = [
        imem_error || dmem_error : SADR;
        !instr_valid: SINS;
        icode == IHALT : SHLT;
        1 : SAOK;
];
```