

“计算机组织结构” 作业 08 参考答案

1.假定某计算机中有一条转移指令，采用相对寻址方式，共占 2 个字节，第一字节是操作码，第二字节是相对位移量（用补码表示），CPU 每次从内存只能取一个字节。假设执行到某转移指令时 PC 的内容为 200，执行该转移指令后要求转移到 100 开始的一段程序执行，则该转移指令第二字节的内容应该是多少（二进制表示，需要在末尾加 B）？

10011010B

$100=200+2+Offset$, $Offset=100-202=-102=10011010B$

2.假设地址为 1200H 的内存单元中的内容为 120CH，地址为 120CH 的内存单元的内容为 38B8H，而 38B8H 单元的内容为 88F9H。说明以下各情况下操作数是多少（十六进制表示，需要在末尾加 H）？[陈姿丽，121250018]

(1)操作数采用变址寻址，变址寄存器的内容为 12，指令中给出的形式地址为 1200H。

(2)操作数采用一次间接寻址，指令中给出的地址码为 1200H。

(3)操作数采用寄存器间接寻址，指令中给出的寄存器编号为 8，8 号寄存器的内容为 1200H。

(1)有效地址 $EA=000CH+1200H=120CH$ 操作数为 **38B8H**

(2)有效地址 $EA=(1200H)=120CH$ 操作数为 **38B8H**

(3)有效地址 $EA=1200H$ 操作数为 **120CH**

3.考虑一个 16 位处理器，它的一条装入指令以如下情况出现在主存，起始地址为 200。

200	Load to AC	Mode
201	500	
202	下一条指令	

第一字的第一部分指出此指令是将一个值装入累加器。Mode 字段用于指定一种寻址方式。若寻址方式需要的话，Mode 字段拨出一部分指定源寄存器；这里假定使用的源寄存器是 R1，有值 400。还有一个基址寄存器，它有值 100。地址 201 处的值 500，可以是立即数也可以是地址计算的一部分。假定位置 399 处有值 999，位置 400 处有值 1000，如此等等。请对如下寻址方式确定将被装入的操作数（十进制表示）：

(3-1)直接

(3-2)立即

(3-3)间接

(3-4)PC 相对

(3-5)基址

(3-6)寄存器

(3-7)寄存器间接

(3-8)变址（用 R1 自动增量）

[潘琦，121250105]

- | | |
|--------------------------|----------------------|
| (1)有效地址 EA=500 | 被装入的操作数为 1100 |
| (2)有效地址 EA=201 | 被装入的操作数为 500 |
| (3)有效地址 EA=(500)=1100 | 被装入的操作数为 1700 |
| (4)有效地址 EA=200+2+500=702 | 被装入的操作数为 1302 |
| (5)有效地址 EA=100+500=600 | 被装入的操作数为 1200 |
| (6)有效地址 EA=R1 | 被装入的操作数为 400 |
| (7)有效地址 EA=(R1)=400 | 被装入的操作数为 1000 |
| (8)有效地址 EA=500+400=900 | 被装入的操作数为 1500 |

4.若 CPU 取并执行一条间接地址方式指令，指令是：

(1)一个要求单操作数的计算，CPU 需要访问存储几次？

(2)一个转移，CPU 需要访问存储几次？

(1)CPU 访问主存 **3** 次

CPU 取指令访问主存 1 次；2.CPU 间接寻址取得操作数需访问主存 2 次（因为是单操作数，所以是 $AC \leftarrow AC + Y$ ，所以无存回操作）

[王子安, 141250146]

(2)CPU 访问主存 **2** 次

1.CPU 取指令访问主存 1 次；2.CPU 取转移地址访问主存 1 次

5.考虑一个包括基址带变址寻址方式的处理器。假设遇到使用这种寻址方式的一条指令，指令给定的偏移量是 1970（十进制）。当前的基址和变址寄存器分别有十进制数 48022 和 8。操作数的地址是什么（十进制表示）？

操作数地址为 $48022 + 8 + 1970 = \mathbf{50000}$

6.一 PC 相对寻址方式的转移指令存于地址为 620（十进制）的存储器位置中。它要转移到 530（十进制）位置上。指令长度为 32 位，其中地址字段长度是 10 位，其二进制值是什么（二进制表示，需要在末尾加 B）？

执行到转移指令时，PC 已经完成自增，值为 $620 + 4 = 624$ ，则 $\text{offset} = 530 - 624 = -94$ 。

由于指令中地址段长度为 10 位，二进制表示为 **1110100010B**。

7. 某计算机指令系统采用定长指令字格式，指令字长 16 位，每个操作数的地址码长 6 位。指令分为 2 地址、1 地址和 0 地址三类。如果 2 地址的指令有 k_2 条，0 地址的指令有 k_0 条，那么 1 地址的指令最多有多少条？（提示：任何指令不能有二义性，即任何指令无法同时用 2-、1-、0-地址法中两种或两种以上方式解释。）[刘璟,121250083]

为了避免指令的二义性，要求同一条指令不能同时可能被解释为 2 地址、1 地址和 0 地址中的两种或三种。对于 1 地址指令，操作数长度为 6 位，因此操作码长度为 10 位。由于 2 地址指令共有 k_2 种，所以前 10 位的取值中有 $k_2 \times 2^6$ 种可以被解释为 2 地址指令；由于 0 地址指令共有 k_0 种，所以前 10 位的取值中至少有 $\lceil k_0 / 2^6 \rceil$ 种可以被解释为 0 地址指令。所以 1 地址指令最多有 $k_1 = 2^{10} - k_2 \times 2^6 - \lceil k_0 / 2^6 \rceil$ 。

补充:

$$k_0 = 2^{16} - k_2 \times 2^{12} - k_1 \times 2^6。$$

$$k_2 = 2^4 - \left\lceil \left(\left\lfloor \frac{k_0}{2^6} \right\rfloor + k_1 \right) / 2^6 \right\rceil。$$

[熊禧华, 191250161]

8. 以 0-、1-、2-、3-地址法分别编写程序来计算:

$$X = (A + B \times C) / (D - E \times F)$$

0 地址	1 地址	2 地址	3 地址
PUSH M	LOAD M	MOV(X<-Y)	MOVE(X<-Y)
POP M	STORE M	ADD(X<-X+Y)	ADD(X<-Y+Z)
ADD	ADD M	SUB(X<-X-Y)	SUB(X<-Y-Z)
SUB	SUB M	MUL(X<-X×Y)	MUL(X<-Y×Z)
MUL	MUL M	DIV(X<-X/Y)	DIV(X<-Y/Z)
DIV	DIV M		

其中, 0 地址法是采用了堆栈, 每次对堆栈顶端的两个数进行操作, 例如 ADD 实际上是用堆栈次顶端的数加上堆栈顶端的数。

0 地址	1 地址	2 地址	3 地址
PUSH A	LOAD E	MOV R0,E	MUL R0,B,C
PUSH B	MUL F	MUL R0,F	ADD R0,A,R0
PUSH C	STORE P	MOV R1,D	MUL R1,E,F
MUL	LOAD D	SUB R1,R0	SUB R1,D,R1
ADD	SUB P	MOV R0,B	DIV X,R0,R1
PUSH D	STORE P	MUL R0,C	
PUSH E	LOAD B	ADD R0,A	
PUSH F	MUL C	DIV R0,R1	
MUL	ADD A	MOV X,R0	
SUB	DIV P		
DIV	STORE X		
POP X			

9. 假设某个计算机只有一条指令:

SUBS X 累加器减去位置 X 处的内容, 结果存入累加器和位置 X 处。

通过编程实现以下功能:

- 将位置 X 处的数据传输到累加器
- 将累加器的数据传输到位置 X 处
- 将位置 X 处的内容加到累加器

(提示: 可以使用 1 个或多个内容为 0 的位置 Y、Z……)

假设 AC 处的初始值为 a, X 处的值为 x, Y、Z 处的初始值为 0

- SUBS Y; // AC = a, X = x, Y = a

SUBS Y; // AC = 0, X = x, Y = 0

SUBS X; // AC = -x, X = -x, Y = 0

SUBS Y; // AC = -x, X = -x, Y = -x

SUBS Y; // AC = 0, X = -x, Y = 0

```

SUBS X;      // AC = x, X = x, Y = 0

b) SUBS Y;   // AC = a, X = x, Y = a
    SUBS X;   // AC = a-x, X = a-x, Y = a
    SUBS X;   // AC = 0, X = 0, Y = a
    SUBS Y;   // AC = -a, X = 0, Y = -a
    SUBS X;   // AC = -a, X = -a, Y = -a
    SUBS Y;   // AC = 0, X = -a, Y = 0
    SUBS X;   // AC = a, X = a, Y = 0

c) SUBS Y;   // AC = a, X = x, Y = a, Z = 0
    SUBS Z;   // AC = a, X = x, Y = a, Z = a
    SUBS Y;   // AC = 0, X = x, Y = 0, Z = a
    SUBS X;   // AC = -x, X = -x, Y = 0, Z = a
    SUBS Z;   // AC = -x-a, X = -x, Y = 0, Z = -x-a
    SUBS Y;   // AC = -x-a, X = -x, Y = -x-a, Z = -x-a
    SUBS Y;   // AC = 0, X = -x, Y = 0, Z = -x-a
    SUBS Z;   // AC = x+a, X = -x, Y = 0, Z = x+a

```