「计算机系统概论」TA Session 4

2022-2023 短学期计算系统概论课程

By @HobbitQia

Homework 2

Float Point Number

2.40 将浮点数转化为十进制表示：
0 11111111 00000000000000000000000
错误答案：2^{128}，应为 $+inf$

需要注意的是 $N = (-1)^S \times M \times 2^E$ 是规格化数的表示，但是阶码（exp）如果为以下两种情况，那么是属于非规格化数：

- $exp = 0$ 时规定 $M = 0.frac, E = 1 - bias = -126(-1023)$
  其中 $frac = 0$ 时，表示的数字为 0.0 （有 +0.0 和 -0.0）
- $exp = 1111\ 1111$ 即全 1 时
  - 若 $frac = 0$ 则这个数表示 $+inf / -inf$
  - 若 $frac \neq 0$ 则这个数表示 NaN （Not a Number）

Float Point Number (Cont.)

Question：将浮点数转为十进制表示：

- 1 11111111 00000000000000000000001
- 0 00000000 11000000000000000000000

Float Point Number (Cont.)

Question：将浮点数转为十进制表示:

- 1 11111111 00000000000000000000001
- 0 00000000 11000000000000000000000

Answer:

- 阶码（exp）全 1，尾数（frac）不为 0，因此 NaN。
- 阶码全 0，因此 $M = 0.frac$。答案为 0.75×2^{-126}

Multiplexer

3.20 16 输入的 Mux 需要几根 output lines，几根 select lines?

Mux 每次从 2^n 个输入中选出一个输入，因此需要 n 根 select lines，1 根 output line。

题目如果不特别说明，默认一根线能传输 1 bit 信息。实际上在计逻中 Mux 的输入可以是多位信号，比如 16 个 16 bit 输入的 Mux，这时依然只有一个输出，但是输出的线就需要 16 条了。

Chapter 5: The LC-3

The ISA of LC-3

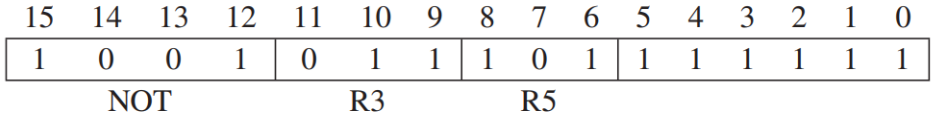
The ISA specifies the *memory organization, register set and instruction set*, including the opcodes, data types, addressing modes of the instruction in the instruction set.

- memory organization 内存组织
  - address space 2^{16} (**i.e.** 65536) locations.
    但并不是所有的地址都会用来存储值。
  - addressability
    在 LC-3 中可寻址能力是 16 bits，我们也称 16 bits 为 1 word（不同计算机字长可能不一样）
- register
  LC-3 中有 8 个 GPR 通用寄存器 $R0, R1, \ldots R7$

The ISA of LC-3 (Cont.)

- instruction set 指令集
  - opcode 操作码
    LC-3 有 15 种指令，因此需要四位来标明 opcode，存在 bit[15:12] 中。(见 P656 图 A.2)
  - data types 数据类型
    Chap02 中提到，bits are just bits. 同样的 bit pattern 可以被不同的指令解释为不同的数值。
  - addressing modes 寻址方式
    LC-3 支持 5 种寻址方式：immediate(literal 立即数)，register，indirect 间接，PC-relative PC 相对地址，base+offset 基地址加偏移量。

Operate Instructions - NOT



reg ← NOT reg

- bit[15:12]: 1001 (opcode of NOT)
- bit[11:9]: DR 目的寄存器
- bit[8:6]: SR 源寄存器
- bit[5:0]: 均被设为 1。

有了 NOT 指令，我们就可以实现 A-B。(对 B 取非加一后再和 A 相加)

NOT (Cont.)

Question: P154 Example 5.3
假设 A，B 分别被存在 R0，R1 中，我们要把 A-B 的结果存在 R2 中，请问下面的代码有什么问题，如果要修改应该如何修改?

```
NOT R1, R1
ADD R2, R1, #1
ADD R2, R0, R2
```

Operate Instructions - LEA

- bit[15:12]: 1110（opcode of LEA）
- bit[11:9]: DR 目的寄存器
- bit[8:0]: offset。我们会把 9 位的偏移量符号扩展为 16 位，并加到 PC（has incremented）上得到我们要访问的地址。

LEA 的格式与 LD 相同，流程也基本相同，区别在于 LEA 并不会真的访问内存！只是把地址（PC+offset）放到寄存器中；相比之下 LD 会把 PC+offset 作为地址去访问内存。

LEA 并不和内存交互，按照第三版书的定义应该是属于 Operate Instructions。

Operate Instructions - LEA

- bit[15:12]: 1110 (opcode of LEA)
- bit[11:9]: DR 目的寄存器
- bit[8:0]: offset。我们会把 9 位的偏移量符号扩展为 16 位，并加到 PC（has incremented）上得到我们要访问的地址。

LEA 的格式与 LD 相同，流程也基本相同，区别在于 LEA 并不会真的访问内存！只是把地址（PC+offset）放到寄存器中；相比之下 LD 会把 PC+offset 作为地址去访问内存。

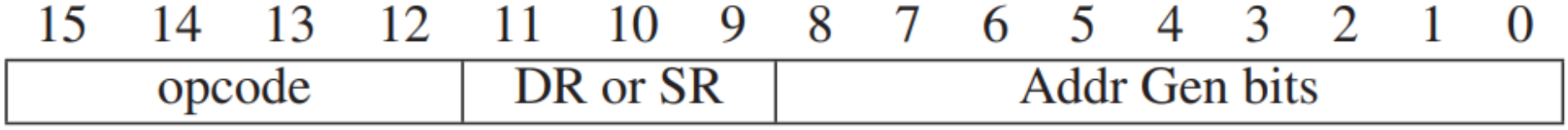
LEA 并不和内存交互，按照第三版书的定义应该是属于 Operate Instructions。

e.g. 假设 MEM[x2000]=xABCD 则 LD R1, x2000 的结果为 xABCD，但 LEA R1, x2000 的结果为 x2000。

Data Movement Instructions

从内存中读出数据称为 load，把数据从寄存器中存到内存中称为 store。

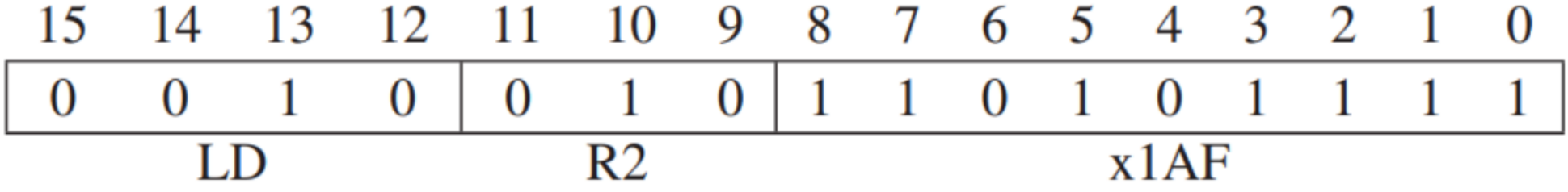
LC-3 只有 6 条指令可以和内存交互：LD，LDR，LDI，ST，STR，STI。



需要注意的是只有 LD，LDR，LDI 会设置 CC 条件码，store 类指令不会。如果是 load 类指令，我们会根据最后从内存中加载出来，并且要被放入寄存器的值更新 CC。

此外，bit[11:9] 对于 load 指令放的是 DR，对于 store 指令放的是 SR。

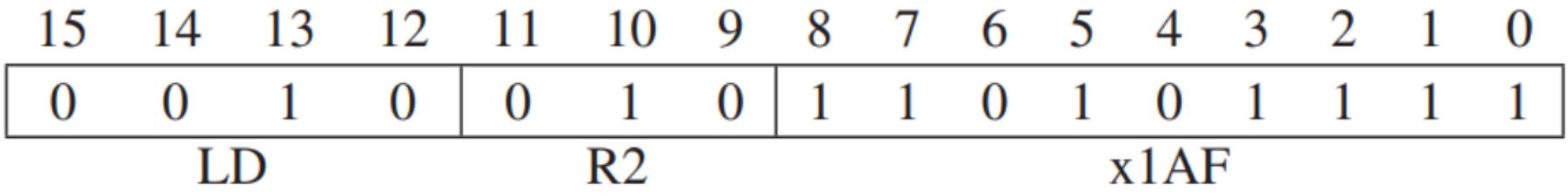
PC-Relative Mode



LD (opcode=0010) and **ST** (opcode=0011) specify the **PC-relative** addressing mode.

要访问的地址通过 PC(has incremented)+ offset(bit[8:0] 符号扩充为 16 位) 计算。

PC-Relative Mode

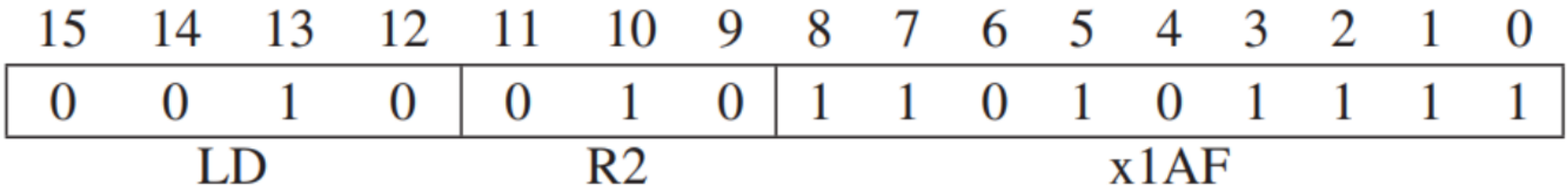


LD (opcode=0010) and **ST** (opcode=0011) specify the **PC-relative** addressing mode.

要访问的地址通过 PC(has incremented)+ offset(bit[8:0] 符号扩充为 16 位) 计算。

Question： PC 相对的寻址方式能访问哪个范围内的数据？(假设 load/store 指令的地址是 x)

PC-Relative Mode



LD (opcode=0010) and **ST** (opcode=0011) specify the **PC-relative** addressing mode.

要访问的地址通过 PC(has incremented)+ offset(bit[8:0] 符号扩充为 16 位) 计算。

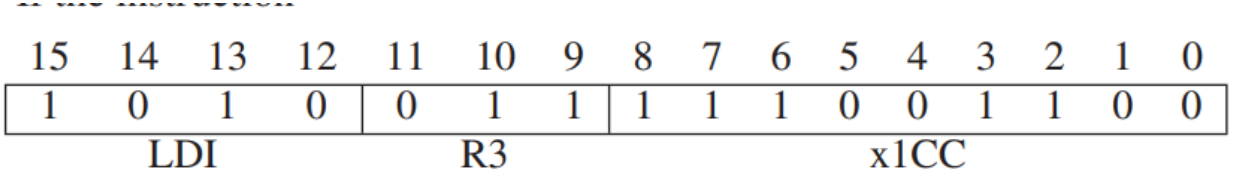
Question： PC 相对的寻址方式能访问哪个范围内的数据？(假设 load/store 指令的地址是 x)

$$[x - 2^8 + 1, x + 2^8]$$

Indirect Mode

LDI (opcode=1010) and **STI**(opcode=1011) specify the **indirect** addressing mode.

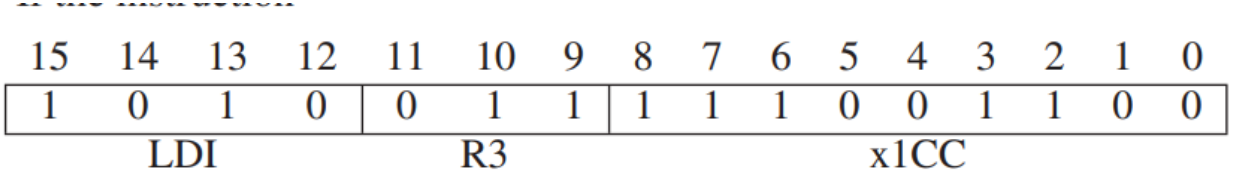
间接寻址，指令格式和 LD 相同，只是需要访问两次内存：首先计算出 PC+offset，随后访问内存取出值，然后将刚刚取出的值作为地址再次访问内存，最后取出的值加载到寄存器中。



Indirect Mode

LDI (opcode=1010) and **STI**(opcode=1011) specify the **indirect** addressing mode.

间接寻址，指令格式和 LD 相同，只是需要访问两次内存：首先计算出 PC+offset，随后访问内存取出值，然后将刚刚取出的值作为地址再次访问内存，最后取出的值加载到寄存器中。

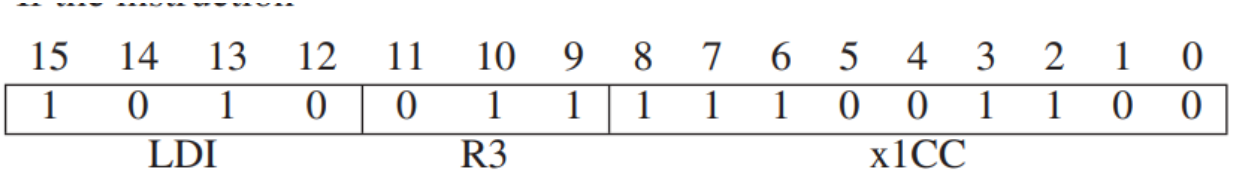


Question：Indirect 寻址能访问哪个范围内的数据？（假设 load/store 指令的地址是 x）

Indirect Mode

LDI (opcode=1010) and **STI**(opcode=1011) specify the **indirect** addressing mode.

间接寻址，指令格式和 LD 相同，只是需要访问两次内存：首先计算出 PC+offset，随后访问内存取出值，然后将刚刚取出的值作为地址再次访问内存，最后取出的值加载到寄存器中。

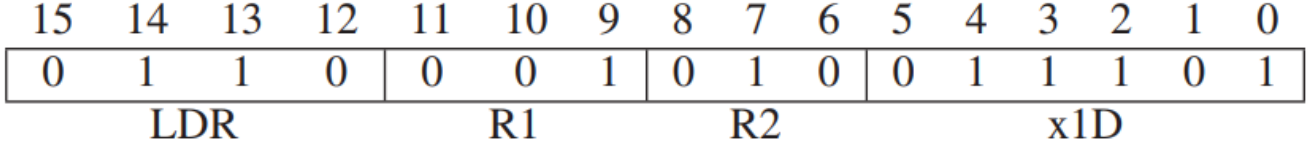


Question：Indirect 寻址能访问哪个范围内的数据？(假设 load/store 指令的地址是 x)

Any address can be accessed by indirect mode.

Base+offset Mode

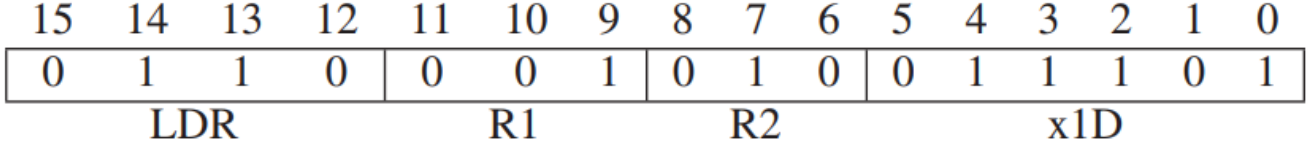
LDR (opcode=0110) and **STR** (opcode=0111) specify the **Base+offset** addressing mode.



- bit[11:9]: DR（如果指令是 STR，那么这里是 SR 源寄存器）
- bit[8:6]: Base 基寄存器
- bit[5:0]: offset。6 位需要符号扩充为 16 位，随后和 base register 相加得到要访问的内存地址。

Base+offset Mode

LDR (opcode=0110) and **STR** (opcode=0111) specify the **Base+offset** addressing mode.

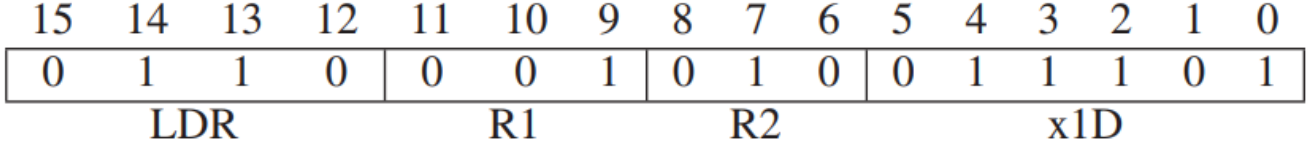


- bit[11:9]: DR（如果指令是 STR，那么这里是 SR 源寄存器）
- bit[8:6]: Base 基寄存器
- bit[5:0]: offset。6 位需要符号扩充为 16 位，随后和 base register 相加得到要访问的内存地址。

Question：Base+offset 寻址能访问哪个范围内的数据？（假设 load/store 指令的地址是 x）

Base+offset Mode

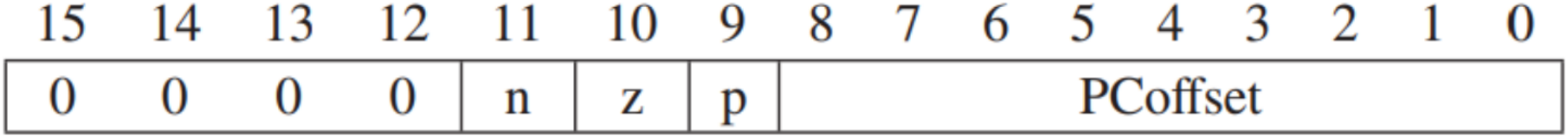
LDR (opcode=0110) and **STR** (opcode=0111) specify the **Base+offset** addressing mode.



- bit[11:9]: DR（如果指令是 STR，那么这里是 SR 源寄存器）
- bit[8:6]: Base 基寄存器
- bit[5:0]: offset。6 位需要符号扩充为 16 位，随后和 base register 相加得到要访问的内存地址。

Question：Base+offset 寻址能访问哪个范围内的数据？（假设 load/store 指令的地址是 x）

Control Instructions - BR



用 nzp 表示 BR 指令的 bit[11:9]，用 NZP 表示 CC 条件码，则我们跳转的条件为：$n \cdot N + z \cdot Z + p \cdot P$ 为真。

这里 CC 是由上一条能改变 CC 的指令决定。

在用 LC-3 写汇编代码时，BRz 相当于 nzp=010。需要注意的是 BR 等价于 BRnzp，即我们不允许有 nzp 均为 0 的指令。

BR Example - Loop

```
for(r0=0;r0<3;r0++){
    ...
}
```

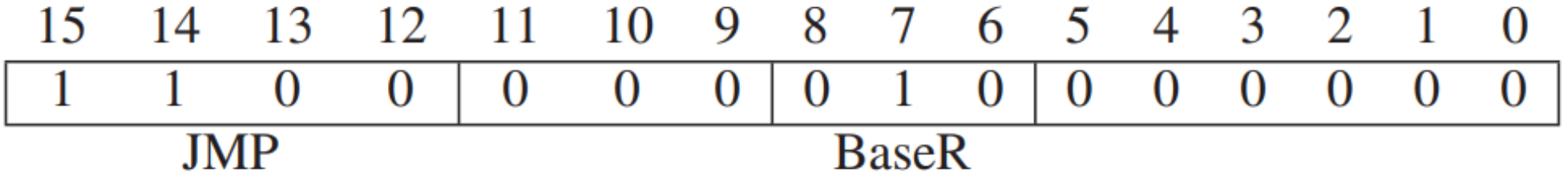
```
INIT:           AND R0,R0,0
CHECK:
                ADD R1,R0,-3
                BRzp END
LOOP:
                ...
ENDBRACE:
                ADD R0,R0,1
                BR Check
END:            HALT
```

BR Example - If

```
if (r0 > 0) {
    do sth...
}
else {
    do sth...
}
```

```
                ADD R0, R0, #0      ; 这样可以让 R0 影响 CC
                BRp IF_TAKEN
ELSE:
                ...                 ; else 内的内容
ENDELSE:
                BRnzp OUT_OF_BRACE
IF_TAKEN:       ...                 ; if 内的内容
OUT_OF_BRACE:   ...                 ; if(){}else(){} 后的内容
```

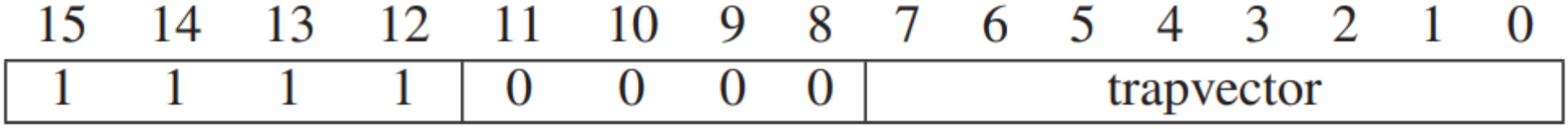
Control Instructions - JMP



- bit[15:12]: 1100
- bit[11:9]=000, bit[5:0]=000000: 默认为 0。
- bit[8:6]: BaseR.

JMP 直接跳到 BaseR 所存储的值处。可以跳到任何地方。

Control Instructions - TRAP



- bit[15:12]: 1111
- bit[11:8]=0000: 默认为 0。
- bit[7:0]: trapvector。

我们需要一些调用系统的服务（比如从键盘获取输入，输出到终端，终止程序 ...）这称为 service call 服务调用（OS 中也称为 system call 系统调用）。

在 LC-3 内核部分（x3000 前面的一部分）放有很多处理程序，用来执行这些系统调用。如果我们想要调用他们，就需要利用 trap 指令。

TRAP (Cont.)

对于这些系统服务，他们都有我们分配的 trapvector，其中

- Input a character from the keyboard (trapvector = x23)
- Output a character to the monitor (trapvector = x21)
- Halt the program (trapvector = x25)

对于 trap 指令，我们会把 trapvector 零扩充为 16 位作为地址，取出对应地址的值。这个值就是 trap 处理程序的地址，接下来我们跳到处理程序的地方即可。

TRAP (Cont.)

对于这些系统服务，他们都有我们分配的 trapvector，其中

- Input a character from the keyboard (trapvector = x23)
- Output a character to the monitor (trapvector = x21)
- Halt the program (trapvector = x25)

对于 trap 指令，我们会把 trapvector 零扩充为 16 位作为地址，取出对应地址的值。这个值就是 trap 处理程序的地址，接下来我们跳到处理程序的地方即可。

e.g. HALT 的 trapvector 为 0x25，可以在 LC-3 中看到 0x25 的值是 0x0366，这就是 HALT 处理程序的地址。因此在调用 HALT 时，我们会跳到 0x0366。

谢谢大家

Question?