Introduction to Computer System Organization

SZU Review

Chapter5

ISA(LC-3)

Question: ISA 的结构有哪些?

1. ISA Structure

   (1). 内存组织 2^k × m bit (e.g. 2^{16} × 16 bit)

   (2). 寄存器组织 2^k × m bit (e.g. LC-3: 2^3 × 16 bit)

   (3). 以上(1),(2) 确定下来之后 --> (设计) 指令集合

2. 指令格式和寻址方式

   LC-3 指令 --> (更改指令,仿照现有的指令格式) --> 新的指令 (unfamiliar)

3. LC -3 的主要指令格式/助记符号

   指令的机器语言格式与助记符号之间的转换.

LC-3 指令集

一条指令分为两个部分：

1.操作码（做什么）

2.操作数（对谁操作）

Definition:

(Instruction set of ISA)

1.操作码的集合

2.数据类型

3.寻址模式

LC-3的ISA结构定义了15条指令，每条指令对应一个操作码（指令的bit[15:12]）

容易注意到理论上应存在16种不同的opcode，但事实上在LC-3中操作码值1101没有定义 我们将其预留（以后在定义）。

所有指令可以分为三类：

1.operate(运算)

2.data movement(数据搬移)

3.control(控制)

Operate

1.运算操作：

ADD/SUB/MUL/DIV

2.逻辑操作:

AND/OR/NOT/XOR

特别需要注意的是，在LC-3中仅支持三种操作指令:

ADD,AND,NOT

NOT

opcode = 1001

Grammar

```
NOT targetR,originalR
```

(targetR = ! (originalR))

e.g.

R5 = 0101000011110000

```
NOT R3,R5
```

R3 = 1010111100001111

(按位取反)

(R3 = ! R5)

ADD (binary instruction)

opcode = 0001

Grammar1

condition code = 0

```
ADD targetR,originalR,R
```

(targetR = originalR + R)

e.g.

R4 = 6

R5 = -18

```
ADD R1,R4,R5
```

R1 = -12

(R1 = R4 + R5)

Grammar2

condition code = 1

```
ADD targetR,originalR,#number
```

(targetR = originalR + number)

e.g.

R4 = 6

```
ADD R1,R4,#-2
```

R1 = 4

(R1 = R4 - 2)

AND (binary instruction)

opcode = 0101

Grammar1

condition code = 0

```
AND targetR,originalR,R
```

(targetR = originalR & R)

e.g.

R5 = 1101000011110010

R6 = 1010111100001110

```
AND R4,R5,R6
```

R4 = 1000000000000010

(R4 = R5 & R6)

Grammar2

condition code = 1

```
AND targetR,originalR,#number
```

(targetR = original & number)

e.g.

```
AND R2,R2,0
```

(R2 寄存器清零)

(R2 = R2 & 0)

Data Movement

PC相对寻址:(PC - relative)

LD (opcode = 0010)

(Load Direct)

Function:

从直接地址加载数据到寄存器(通过PC相对偏移寻址)。

Composition:

opcode:LD

target register:DR

base index:PCoffset9

Grammar:

LD DR, PCoffset9

Operation Steps:

DR <- Mem[PC+PCoffset9]

e.g.

```
LD R2, 0xFF ; 将地址(PC+0xFF)处的数据加载到R2
```

ST (opcode = 0011)

(Store Direct)

Function:

将寄存器的值直接存储到内存地址(PC相对寻址)。

Grammar:

ST SR, PCoffset9

Operation Steps:

计算目标地址：PC+PCoffset9

将寄存器SR的值存储到该地址

e.g.

```
ST R0, 0x20 ; 将R0的值存储到(PC+0x20)
```

Operation Process:

(Assume that PC=0x3000, R0=0x1234):

计算地址：0x3000+0x20 = 0x3020

存储数据：Mem[0x3020] = 0x1234

Appliance:

存储全局变量或静态数据(state data)

间接寻址:

LDI (opcode = 1010)

(Load Indirect)

Function:

间接加载数据。先从指令指定的地址读取一个指针，再通过该指针读取实际数据，最后存入目标寄存器(target register)。

Composition:

opcode:LDI

target register:DR

base index:PC offset 9(相当于PC的偏移地址)

Grammar:

`LDI DR, PCoffset9`

Operation Steps:

1. 计算地址：`Mem[PC+PCoffset9]`获取指针
2. 加载数据：`DR <- Mem[Mem[PC+PCoffset9]]`

e.g.

```
LDI R0, 0x30 ; 从地址(PC+0x30)读取指针，再通过指针加载数据到R0
```

STI (opcode = 1011)

(Store Indirect)

Function:

间接存储数据。先将数据写入指令指定的地址指向的指针位置。

Composition:

opcode:STI

original register:SR

base index:PCoffset9

Grammar:

STI SR, PCoffset9

Operation Steps:

计算地址：Mem[PC+PCoffset9]获取指针

存储数据：Mem[Mem[PC+PCoffset9]] <- SR

e.g.

```
STI R0, 0x20 ; 将R0的值存储到(PC+0x20)
```

基址偏移寻址:(Base + Offset)

LDR (opcode = 0110)

(Load Base+Offset)

Function:

通过基址寄存器+偏移量加载数据。

Composition:

opcode:LDR

target register:DR

base register:BaseR

index:offset6(6位有符号偏移量)

Grammar:

LDR DR, BaseR, offset6

Operation Steps:

DR <- Mem[BaseR+offset6]

e.g.

```
LDR R3, R4, -10 ; 从地址(R4-10)加载数据到R3
```

STR (opcode = 0111)

(Store Base + Offset)

Function:

将寄存器的值存储到基址寄存器+偏移量指定的内存地址。

Grammar:

STR SR, BaseR, offset6

Operation Steps:

计算目标地址：BaseR + offset6(符号扩展)

将SR的值存储到该地址

e.g.

```
STR R1, R2, -4 ; 将R1的值存储到(R2-4)
```

Operation Process:

(Assume that R2=0x4000, R1=0x5678):

计算地址：0x4000 + (-4) = 0x3FFC

存储数据：Mem[0x3FFC] = 0x5678

Appliance:

存储到局部变量或数组元素(如R2指向栈帧基址)。

立即数寻址

LEA (opcode = 1110)

(load effective address)

Explaination:被装入寄存器的数值的获取是“立即的”（直接从当前指令片段中抽取，而不需要任何内存访问）

Function:

将有效地址(非数据)加载到寄存器。

Composition:

opcode:LEA

target register:DR

base index:PCoffset9

Grammar:

LEA DR, PCoffset9

Operation Steps:

DR <- PC+PCoffset9 (直接计算地址并存储)

e.g.

```
LEA R1, LABEL ; 将LABEL的地址存入R1
```

Control

条件跳转指令 (opcode = 0000)

(BR Instruction)

指令格式	含义	等效条件
`BRn LABEL`	结果为负时跳转	N=1
`BRz LABEL`	结果为零时跳转	Z=1
`BRp LABEL`	结果为正时跳转	P=1
`BRnz LABEL`	结果为负或零时跳转	N=1 或 Z=1
`BRnp LABEL`	结果为负或正时跳转（非零）	N=1 或 P=1
`BRzp LABEL`	结果为零或正时跳转	Z=1 或 P=1
`BRnzp LABEL`	无条件跳转	总是跳转

(JSR/JSRR Instruction)(Jump to Subroutine)

格式1: JSR LABEL

格式2: JSRR BaseR

功能: 跳转到子程序,将返回地址保存在R7

示例:

```
JSR  SUB ;调用子程序SUB

JSRR R6  ;调用R6指向的子程序
```

(JMP Instruction)

格式: JMP BaseR

功能: 无条件跳转到BaseR指定的地址

示例: JMP R7; 跳转到R7中的地址.

(RET Instruction)

格式: RET

功能: 从子程序返回 (equals to JMP R7)

示例: RET ;返回到调用者