Readings on Arrays and Indirect Addressing (for Lab 3)

Arrays

The ability to be able to access an array is fundamental to many computer programs. This section will describe how you can access the elements of an array using the LC3 instruction set. We will only deal with arrays of 16-bit data (the word size of LC3), but the idea can be generalize to arrays of more complicated structures, such as an array of records.

Assuming that you have memory set aside in your program, the first thing you need to do before you can manipulate an array is to get its base address (that is, the address of the start of the array). This can be done as follows:

```
LEA R1, Data   ; get the address of the array Data an put
               ;it into R1
```

The `LEA` instruction loads the address of the label DATA and puts it into the specify register. This will give you the address of the first element of the array.

Now that you have the address of the start of the array in R1, you access individual elements of the array by adding an **offset** to R1. An offset is simply an indexing mechanism. You can just use an ADD instruction to add the base and offset to get the final address of the array element in question and then use `LDR` to read, `STR` to store.

To store the value 0 in Data[1], you could do something like

```
LEA R1, Data          ;get address of Data
AND R2,R2,#0          ;R2 <- 0
STR R2,R1,#1          ;Data[1] <- 0
```

Alternatively you can access elements by modifying the base register and keeping the offset at 0

```
LEA R1, Data          ;get address of Data
AND R2,R2,#0          ;R2 <- 0
ADD R1,R1,#1
STR R2,R1,#0          ;Data[1] <- 0
```

To read the value in Data[4] and store it in register R3, you could do something like

```
LEA R1, Data          ;get address of Data
LDR R3,R1,#4          ;R3 <- Data[4]
```

There are several important things to remember when using an offset:

1) The offset must be between –32 and 31. This means you have to update the base to go beyond this range (or simply update the base register).
2) The value must be read into a register, and the value to be stored must come from a register.
3) When in doubt, consult the documentation.

Indirect Addressing

In indirect addressing, the final address (Effective address) is stored at the memory location given by adding the address generator to the current value of the PC. This is like a pointer to a pointer. The instructions `LDI` and `STI` are used for this addressing mode.

For example: Suppose you have two labels Data and Num at addresses 0x3010 and 0x3010 respectively which looks like

```
address
x3010          Data .fill     0x3011
x3011          Num  .fill     #10
```

The instruction `LDI R1,Data` would load the value #10 into `R1`. This is because, the instruction tells the computer to look at label `Data`, get its value (0x3011) and load the data (#10) at address given by this value.

The instructions

```
AND R2,R2,#0
STI R2,DATA
```

would store the value #0 at address x3011.

You can use labels to get the same effect without having to know the addresses

```
address
x3010          Data .fill     Num
x3011          Num  .fill     #10
```