# Register Indirect Addressing in Assembly Language

In assembly language, register indirect addressing allows accessing data from memory by holding the memory address in a register. This method is particularly useful for iterating over data placed in consecutive memory locations, which can help avoid repeated instructions when dealing with a large amount of data. The iAPX88 architecture has four registers that can hold memory addresses: BX, BP, SI, and DI. In this topic, we will explore how to use these registers for indirect addressing, starting with a simple example of adding three numbers using the BX register.

**Example 2.6: Adding Three Numbers Using Indirect Addressing**

```
[org 0x100]
mov bx, num1    ; point bx to first number
mov ax, [bx]    ; load first number in ax
add bx, 2       ; advance bx to second number
add ax, [bx]    ; add second number to ax
add bx, 2       ; advance bx to third number
add ax, [bx]    ; add third number to ax
add bx, 2       ; advance bx to result
mov [bx], ax    ; store sum at num1+6
mov ax, 0x4c00  ; terminate program
int 0x21
num1: dw 5, 10, 15, 0
```

In this program, we use the BX register to point to the memory locations of the numbers. BX holds the address of the first number (num1), and we use the instruction 'mov ax, [bx]' to load the value at that address into the AX register. By incrementing BX by 2, we move to the next word in memory and repeat the same operation. This allows us to add multiple numbers using the same instructions without hardcoding the memory addresses. The sum is stored back in memory at the address held

by BX, after it has been advanced to the correct location.

**Example 2.7: Adding Ten Numbers Using a Loop**

```
[org 0x0100]
mov bx, num1    ; point bx to first number
mov cx, 10      ; load count of numbers in cx
mov ax, 0       ; initialize sum to zero
l1: add ax, [bx]  ; add number to ax
add bx, 2       ; advance bx to next number
sub cx, 1       ; numbers to be added reduced
jnz l1          ; if numbers remain, add next
mov [total], ax ; write back sum in memory
mov ax, 0x4c00  ; terminate program
int 0x21
num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0
```

This program demonstrates how to sum multiple numbers using register indirect addressing and a loop. The BX register is used to hold the address of the current number, while CX is used as a counter for the loop. The loop continues until CX becomes zero, at which point the program exits. Each iteration of the loop adds the current number to AX, increments BX to point to the next number, and decrements CX by 1. Once all numbers are added, the sum is stored in memory at the location labeled 'total'.

**Register Indirect Addressing Explained**

Register indirect addressing is a method where the address of the data is stored in a register. In this case, we use BX to hold the address of the numbers in memory, and by modifying the value of BX, we can access different memory locations. The square brackets around BX indicate that we want to

use the contents of the memory location pointed to by BX. This technique is especially useful for working with arrays or sequences of data, as it allows us to iterate over the data without explicitly coding the address of each element.

**Looping and Conditional Jumps**

In assembly language, looping is achieved using conditional jumps. In the second example, we use the 'jnz' (jump if not zero) instruction to repeat the addition process until CX reaches zero. The subtraction instruction 'sub cx, 1' reduces the value of CX by 1 on each iteration, and 'jnz' checks if CX is zero. If it is not, the program jumps back to the label 'l1', repeating the addition and incrementing the address in BX. This process continues until all numbers are added, providing a powerful mechanism for repeating instructions without duplicating code.