JMP and LOOP

The JMP and LOOP instructions in assembly language are used to implement conditional and unconditional transfers of control, respectively.

The **JMP instruction** causes an **unconditional transfer** to a destination, identified by a code label.

The **LOOP** instruction **repeats a block of statements** a specific number of times, using the **ECX register as a counter.**

Here is an example of how to use the JMP and LOOP instructions to implement a simple loop:

```
loop_start:
    ;loop body
    ...
    ;check the loop condition
    cmp ecx, 0
    je  loop_end

    dec ecx ;decrement the loop counter
    jmp loop_start ;jump to the beginning of the loop

loop_end
    ;loop's exit code
```

This loop will execute as long as the ECX register is not equal to zero. When ECX reaches zero, the loop will exit.

The LOOP instruction is a more concise and efficient way to implement a loop than using a JMP instruction.

The LOOP instruction automatically decrements the ECX register and performs the jump to the loop label, so you don't have to explicitly write these instructions yourself.

Here is an example of how to use the LOOP instruction to implement

the same loop as above:

```
loop_start:
  ; loop body
  ...

  ; loop condition
  cmp ecx, 0
  je loop_end

  ; loop
  loop loop_start

loop_end:
  ; loop exit code
```

This loop is equivalent to the previous loop, but it is more concise and efficient.

You can also use the LOOP instruction to implement more complex loops, such as loops that iterate over an array or loops that exit based on a different condition.

```
.data
    count DWORD ?

.code
    mov ecx, 100 ; set loop count
    top:
      mov count, ecx ; save the count

      ; modify ECX

      mov ecx, count ; restore loop count
    loop top

    ; loop exit code
```

The loop count is initialized to 100 by moving the value 100 into the ECX register.

The loop counter is saved in the count variable by moving the value of the ECX register to the count variable.

The ECX register is modified. The loop counter is restored by moving the value of the count variable to the ECX register.

The LOOP instruction is executed, which decrements the ECX register and jumps to the top label if the ECX register is not equal to zero.

The loop exit code is executed if the ECX register is equal to zero. The following is a more detailed explanation of each step:

```
top:
```

This label marks the beginning of the loop.

```
  loop top
```

This instruction decrements the ECX register and jumps to the top label if the ECX register is not equal to zero. This causes the loop to repeat until the ECX register is equal to zero.

==========

Nested Loops

==========

```
.data
    count DWORD ?
.code
    ; set outer loop count
    mov ecx, 100

    L1:
      ; save outer loop count
      mov count, ecx

      ; set inner loop count
      mov ecx, 20

    L2:
      ; loop body
      ...
      ; decrement inner loop counter
      dec ecx
      ; jump to beginning of inner loop if inner loop counter is not zero
      jne L2
      ; restore outer loop count
      mov ecx, count
      ; decrement outer loop counter
      dec ecx
      ; jump to beginning of outer loop if outer loop counter is not zero
      jne L1
```

The nested loop you provided is an example of a combination of an outer loop and an inner loop, a common construct in programming. Let me explain what this nested loop does:

Outer Loop (L1):

• The outer loop is controlled by the ecx register and is initialized with the value 100.

• Inside the outer loop:◇ The current value of ecx (the outer loop counter) is saved into the count variable.

• The inner loop is initialized by setting ecx to 20.

Inner Loop (L2):

• The inner loop is also controlled by the ecx register and is initialized with the value 20.Inside the inner loop:

• There is a placeholder comment for the loop body, where actual operations or instructions would be performed.

▪ dec ecx is used to decrement the inner loop counter.

▪ jne L2 is a conditional jump instruction. It checks if the inner loop counter (ecx) is not equal to zero. If it's not zero, the program jumps back to the beginning of the inner loop (L2) to continue iterating.

Outer Loop Continuation:

• After the inner loop completes (when ecx becomes zero), the program restores the original value of ecx from the count variable.

• dec ecx is used to decrement the outer loop counter.

• jne L1 is another conditional jump instruction. It checks if the outer loop counter (ecx) is not equal to zero. If it's not zero, the program jumps back to the beginning of the outer loop (L1) to continue iterating.

In summary, this nested loop structure is designed to execute the inner loop 20 times for each iteration of the outer loop, resulting in a total of 100 * 20 = 2000 iterations in total.

The actual operations or instructions within the loop body are not provided in the code snippet, but they would be executed repeatedly as part of the loop's functionality.

```
;NB: Don't indent assembly code like me, I just want clarity
.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.data
    intarray DWORD 10000h,20000h,30000h,40000h

.code
    main PROC
      ; 1: EDI = address of intarray
      mov edi, OFFSET intarray
      ; 3: sum = 0
      mov eax, 0
      ; 2: initialize loop counter
      mov ecx, LENGTHOF intarray
      L1:
        ; 5: add an integer
        add eax, [edi]
        ; 6: point to next element
        add edi, TYPE intarray
        ; 7: repeat until ECX = 0
        loop L1
      ; Exit with success code
      invoke ExitProcess, 0
main ENDP
END main
```

The code defines the intarray array, which contains four 16-bit integers.

This section of the code defines the main() function, which is the entry point for the program.

```
.code
main PROC
```

Move the address of the intarray array into the EDI register. The EDI

register will be used as an indexed operand to access the array elements.

Move the value 0 into the EAX register. The EAX register will be used to accumulate the sum of the array elements.

Move the length of the intarray array into the ECX register. The ECX register will be used as the loop counter.

L1: This label marks the beginning of the loop.

Add the value at the current address in the EDI register to the EAX register. The EDI register contains the address of the current array element.

Increment the EDI register by the size of an array element. This points the EDI register to the next array element.

Jump back to the L1 label if the ECX register is not equal to zero. This causes the loop to repeat until the ECX register is zero.

Invoke the ExitProcess function, which exits the program with a success code.

```
main ENDP
END main
```

These directives mark the end of the main() function and the program.