Conditional Structures

Conditional structures in programming allow you to make decisions based on conditions. You've likely used these in high-level languages like C++. These structures involve evaluating a condition and executing different sets of instructions based on whether the condition is true or false.

Block-Structured IF Statements: In the context of C++ and similar languages, an IF statement consists of a boolean expression followed by two sets of statements. One set of statements executes when the expression is true, and the other set when the expression is false. The "else" part is optional, meaning you can have just an "if" without an "else."

Translation to Assembly: When you translate high-level code into assembly language, you need to perform two main steps:

Evaluate the Boolean Expression: This step involves comparing values in registers or memory locations in such a way that it affects a CPU status flag. In your example, they use the CMP instruction to compare op1 and op2. This sets flags in the CPU.

Conditional Jumps: Based on the CPU status flags set by the comparison, you decide which branch of the code to execute. In your example, they use JNE (Jump if Not Equal) to skip ahead if op1 is not equal to op2.

Code Example: The code example you provided in C++ checks if op1 is equal to op2. If they are equal, it sets X to 1 and Y to 2.

Now, let's look at the assembly code they provided:

```
340 mov eax, op1    ; Move op1 into the EAX register
341 cmp eax, op2    ; Compare op1 and op2
342 jne L1          ; Jump if not equal (if op1 != op2)
343 mov X, 1        ; Assign 1 to X
344 mov Y, 2        ; Assign 2 to Y
345 L1:
```

mov eax, op1: This line moves the value of op1 into the EAX register for comparison.

cmp eax, op2: It then compares the values in EAX (which now holds op1) and op2.

jne L1: If the comparison shows that op1 is not equal to op2, it jumps to the label L1.

mov X, 1 and mov Y, 2: Inside L1, it assigns 1 to X and 2 to Y. This code allows for efficient execution by letting the code "fall through" when the boolean condition is true. It avoids unnecessary jumps, making it more compact.

The alternative implementation using JE (Jump if Equal) would be slightly less compact but functionally equivalent.

..it jumps to the label L1. why not say it jumps to the procedure L1?

ChatGPT

You're correct; the term "procedure" is often used in programming to refer to a block of code that has a specific purpose or function. In the context of assembly language, especially in the example you provided, "L1" is more commonly referred to as a "label."

In assembly language, labels are used to mark specific locations in the code, typically for branching purposes. So, when I mentioned "jumps to the label L1," it means that the program's execution flow will transfer to the point in the code marked by the label "L1."

While the term "procedure" is indeed used in programming, it's usually associated with higher-level languages like C++ or C# and refers to a named block of code that can be called or invoked with specific parameters. In assembly language, labels are more commonly used to mark positions in the code for control flow.

So, you could also say "jumps to the procedure L1" if you were discussing this in a higher-level language context, but in the context of assembly language, "jumps to the label L1" is the more accurate terminology.

accurate terminology.

In high-level programming languages, labels and procedures are distinct concepts:

• **Labels:** Labels are not typically used in high-level languages like C++, C#, Python, etc. In these languages, you don't typically define labels explicitly. Instead, control flow is managed using constructs like functions, loops, and conditional statements. Labels are more commonly associated with assembly language or lower-level languages.

• **Procedures (Functions):** Procedures and functions are an integral part of high-level programming languages. They are named blocks of code that can be called or invoked with specific parameters. Functions in high-level languages serve various purposes, such as performing a specific task, returning values, or modifying data. They encapsulate a set of instructions and can be reused throughout the program.

So, in summary:

• In high-level languages, you use functions (or procedures) to encapsulate code with a specific purpose and make your code more modular and readable.

• Labels are more commonly associated with low-level languages like assembly, where they are used as markers for control flow and branching.

• The terminology and usage can vary between different programming languages, but this distinction is generally applicable.

We want to implement these conditional structures in real life:

Example 2: Setting Cluster Size Based on Volume Capacity

In this example, the pseudocode is used to set the cluster size of a disk volume in the NTFS file system. The cluster size depends on the volume's capacity, specifically, whether it's less than 16 terabytes.

Here's the assembly language implementation:

```
mov clusterSize, 8192   ; Assume a larger cluster size

cmp terrabytes, 16      ; Compare terrabytes with 16
jae next                ; If greater or equal, jump to 'next'

mov clusterSize, 4096   ; Set clusterSize to 4096 for smaller volume

next:
```

mov clusterSize, 8192: Initially, the code assumes a larger cluster size of 8192.

cmp terrabytes, 16: It compares the variable terrabytes with 16.

jae next: If terrabytes is greater than or equal to 16, it jumps to the next label.

mov clusterSize, 4096: Inside the next label, it sets clusterSize to 4096, indicating a smaller cluster size.

The code effectively changes the cluster size based on the volume size, as described in the pseudocode.

Example 3: Conditional Routine Calls

In this example, the pseudocode involves calling different routines based on a condition, specifically, whether op1 is greater than op2.

Here's the assembly language implementation:

```
360 mov eax, op1        ; Move op1 to a register
361 cmp eax, op2        ; Compare op1 and op2
362 jg A1               ; If op1 > op2, jump to A1 (call Routine1)
363 call Routine2       ; Otherwise, call Routine2
364 jmp A2              ; Jump to A2 (exit the IF statement)
365
366 A1:
367 call Routine1
368
369 A2:
```

mov eax, op1: It moves the value of op1 into the EAX register.

cmp eax, op2: It compares the values in EAX (which now holds op1) and op2.

jg A1: If op1 is greater than op2, it jumps to A1, which calls Routine1.

call Routine2: If the comparison is false, it calls Routine2.

jmp A2: After executing either Routine1 or Routine2, it jumps to A2, which marks the exit point for the IF statement.

The code effectively calls different routines based on the condition provided in the pseudocode.