High-level languages like C are better suited than assembly for clearly expressing even the simplest of algorithms. So it’s a common practice when programming in assembly to sketch out a solution in a language like C and then to use it as a guide for developing a corresponding solution in assembly.

In C (and other high-level languages), “if” statements provide a mechanism for conditional execution of a statement (or group of statements) based on the value of a condition – i.e., the statement is either executed or not, depending on the value of the condition. In assembly, however, the only action that can be controlled by a condition is to branch (the equivalent of a “goto” statement) – i.e., you can only either branch or not, depending on the value of the condition.

Although “goto” statements are part of the C language, programmers are taught to avoid them *(or never introduced to them in the first place)* because their use often leads to programs that are difficult to understand. Unfortunately, we have no choice in assembly. Therefore, translating C into assembly is much easier if the C code is written in a manner that more closely resembles how it must ultimately be written in assembly. Fundamentally, this means that the only thing that an if statement should control is a goto, and also means that “else” statements should be avoided. Shown below are examples of how if-then and if-then-else statements can be rewritten using goto’s.

**if statements must only control goto statements:**

|  |  |
| --- | --- |
| if (*condition*) ***statement* ;** | if (!*condition*) goto L1 ;  *statement* ;  L1: |

*(Note that in the above example, we must use the opposite of the original condition!)*

**If statements must not have an associated else statement:**

|  |  |
| --- | --- |
| if (*condition*) *then-statement* ;  **else *else-statement* ;** | if (!*condition*) goto L1 ;  *then-statement* ;  goto L2 ;  L1: *else-statement ;*  L2: … |
| *// The ternary selection operator is an if-then-else*  *variable* = (*condition*) **?** *then-expression1*  **:** *else-expression2* ; | if (!*condition*) goto L1 ;  *variable = then-expression1* ;  goto L2 ;  L1: *variable = else-expression2 ;*  L2: … |

Compound conditionals are another example of high-level language constructions that do not have a direct equivalent in assembly. When there are multiple conditions, the assembly code must evaluate the first condition and branch or not, then evaluate the second condition and branch or not, etc. This means that we should avoid compound conditionals in if statements, and requires splitting the solution into multiple if statements – each based on a single condition:

**Don’t use compound conditionals in “if” statements:**

|  |  |
| --- | --- |
| if (***condition1* || *condition2***) *statement*; | if (*condition1*) goto L1 ;  if (*condition2*) goto L1 ;  goto L2 ;  L1: *statement* ;  L2:    *Simplify!*  if (*condition1*) goto L1 ;  if (!*condition2*) goto L2 ;  L1: *statement* ;  L2: |

*(Note that the simplified solution uses the opposite of the second condition!)*

|  |  |
| --- | --- |
| if (***condition1* && *condition2***) *statement*; | if (!(*condition1* && *condition2*)) goto L1;  *statement* ;  L1:  *Apply DeMorgan’s Law*  if (!*condition1 ||* !*condition2****)*** goto L1 ;  *statement* ;  L1:  *Simplify!*  if (!*condition1*) goto L1 ;  if (!*condition2*) goto L1 ;  *statement* ;  L1: … |

Compound conditionals sometimes also appear in for loops, while loops, and do-while loops, and must be rearranged in a similar manner:

**Don’t use compound conditionals to control “for” loops:**

|  |  |
| --- | --- |
| for (…; ***condition1* || *condition2***; …)  {  …  } | for (…;true;…)  {  if (*condition1*) goto L1 ;  if (!*condition2*) break ;  L1:  …  } |
| for (…; ***condition1* && *condition2***; …)  {  …  } | for (…;*condition1*;…)  {  if (!*condition2*) break ;  …  } |

**Don’t use compound conditionals to control “while” loops:**

|  |  |
| --- | --- |
| while (***condition1* || *condition2***)  {  …  } | while (true)  {  if (*condition1*) goto L1 ;  if (!*condition2*) break ;  L1: …  } |
| while (***condition1* && *condition2***)  {  …  } | while (*condition1*)  {  if (!*condition2*) break ;  …  } |

**Don’t use compound conditionals to control “do-while” loops:**

|  |  |
| --- | --- |
| do  {  …  } while (***condition1* || *condition2***) ; | do  {  …  if (*condition1*) continue ;  } while (*condition2*) ; |
| do  {  …  } while (***condition1* && *condition2***) ; | do  {  …  if (!*condition1*) break ;  } while (*condition2*) ; |