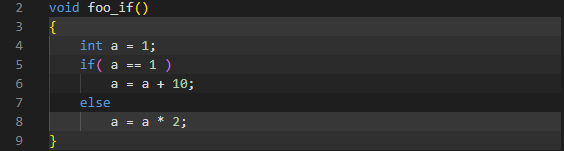
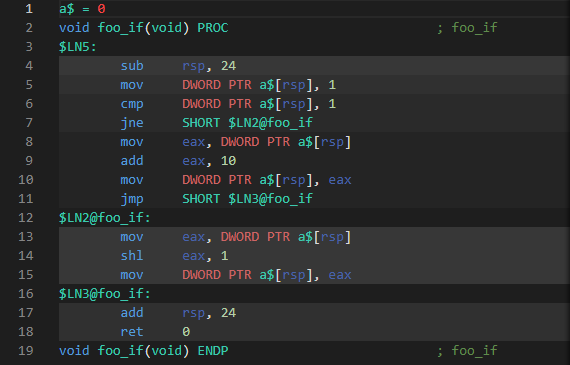
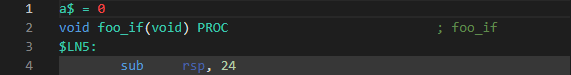
Как работает Условный оператор **If**:



А это на **Assembler**. Разбор далее.





Начало функции. “**a$**” – адрес переменной в памяти.

****

С помощью команды **MOV** копирует значение из одного места в другое. Это “*место*” может быть регистр, ячейка памяти или непосредственное значение (только как исходное значение). Задаётся как “mov приёмник, источник”. Задаётся значение для переменной “**a**” в памяти. По-другому – операнд (далее и так, и так, для удобства)

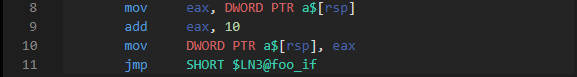
**DWORD PTR** — это командный префикс в ассемблере, который указывает, что операнд должен быть обращен к памяти в виде двухбайтного значения (32 бита).



Для команды **if** выполняется две инструкции. Первая - **CMP** — это инструкция Ассемблера, которая сравнивает 2 операнда. Основное применение инструкции — проверка условий в циклах. **CMP** вычитает один операнд от другого, затем сравнивает, являются операнды равными или нет.

Вторая - команда **JNE** в Ассемблере осуществляет условный переход по метке в случае, если значение флага **ZF** равно 0, и никакого перехода не будет, если значение флага **ZF** равно 1. С неформальной же точки зрения перед командой **JNE** чаще всего идёт команда **CMP**, которая сравнивает два числа и записывает результат сравнения в упомянутый флаг **ZF**.

Место перехода будет показано далее.



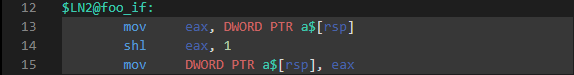
Эти команды выполняются в случае выполнения условия (то есть если перехода по метке не будет).

**EAX** - (*Accumulator register*) — аккумулятор. Применяется для хранения промежуточных данных. В него записывается текущее значение переменной “**a**”.

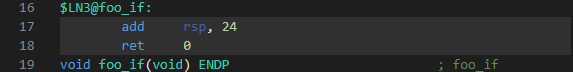
Команда **ADD** служит для реализации операций сложения чисел в различных вариациях. В качестве операндов для команд сложения могут выступать как регистры, так и ячейки памяти и конкретные значения. Команда **ADD** производит простое суммирование операндов. С помощью команды **ADD** происходит суммирование значения из ячейки памяти и числа 10.

Далее с помощью известной уже нам команды **MOV** значение переменной возвращается обратно из промежуточного хранения в “*свою*” ячейку.

Команда **JMP** — это безусловный переход на адрес, указанный с помощью операнда команды. Переход может быть как ближним, в пределах одного сегмента памяти, так и дальним, межсегментным переходом. С его помощью происходит “перескок” **else** и дальнейших его инструкций. Продолжается выполнение команд уже после конструкции **if-else**.



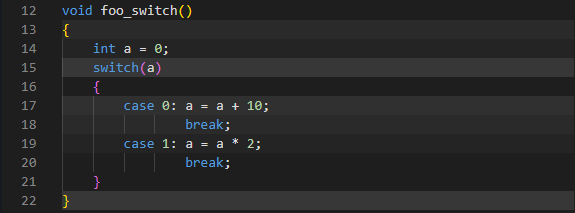
А вот, что будет выполняться, в случае перехода по метке в инструкции проверки условия (если **if** будет неверно). По командам всё точно также. Копируется в промежуточную ячейку, умножение числа на 2 выполняется побитовым сдвигом с помощью команды **SHL** (например: 2\*2 = 0010 🡪 0100). Такое действие равнозначно беззнаковому умножению исходного операнда на два, четыре и т.д. И снова записывается в исходную ячейку памяти.



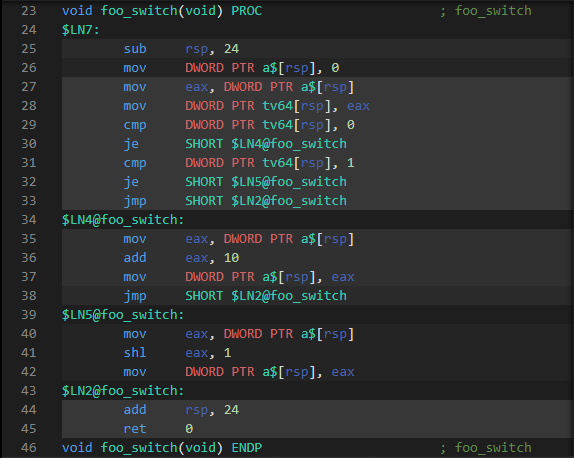
И конец конструкции **if-else**, куда переходит после выполнения инструкций при верном условии if (либо после **else**, но без перехода, команды идут подряд). Здесь могут быть дальнейшие команды программы. Но тут – конец функции.

**RET** в Ассемблере выполняет возврат из ближней процедуры. У этой команды обычно нет операндов, хотя в качестве операнда может быть чётное число (только непосредственное значение). Коротко - эта команда выполняет выход из программы или процедуры.

Условный оператор **switch**:



**Assembler**:



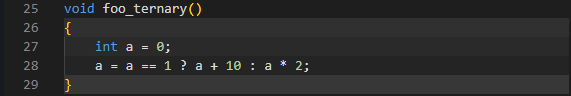
После полного разбора конструкции **if-else** общий синтаксис понятен.

Добавляется **tv64** – сюда записывается значение, которое мы будем сравнивать в дальнейшем. С помощью **CMP** происходит сравнение.

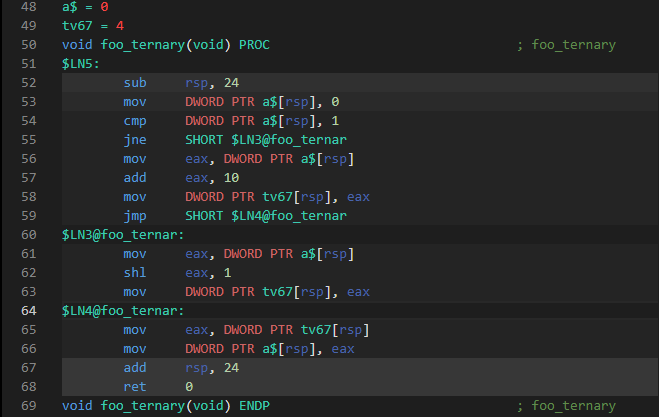
Здесь используется другая команда перехода - команда **JE** в ассемблере выполняет короткий переход, если первый операнд *РАВЕН* второму операнду при выполнении операции сравнения с помощью команды **CMP**.

Т.к. не задано **default** инструкция, то при невыполнении условий будет осуществляться переход в конец конструкции.

Всё остальное, как и в **if-else**.

**Тернарный** оператор: 

**Assembler**:



Очень похоже по структуре на **if**. Только в данном случае сама конструкция находится в назначении значения переменной “**a**”. Здесь также используется дополнительная переменная для работы условного оператора – **tv67** (в **switch** – **tv64**, предназначение у них разное), служит неким мостом между промежуточным регистром. Все остальные команды идентичны: проверка условия, переход по метке, либо продолжение инструкции и в конечном итоге запись в переменную.

Условные операторы в некотором плане схожи, код на ассемблере, а точнее команды и общая структура похожи. Даже тот же тернарный оператор, на **C++** выглядит коротко, но на **Ассемблере** такая же “*массивная*” инструкция, как и остальные условные операторы. Ещё можно заметить, что на операторе **switch** сначала идёт сравнение, а сами команды расположены уже после, переход к ним осуществляется только по отдельным командам. У других операторов если первое условие верно, то код просто продолжается и там же расположены команды для этого условия, а после просто переход в конец. А так – для использования пригодиться может любой из операторов, в различных ситуациях каждый оператор будет удобен по-своему.