**Подробно про безусловные переходы:**

Безусловные переходы в ассемблере — это команды, которые передают управление на указанный адрес или метку независимо от текущего состояния флагов процессора. Основной командой для безусловных переходов является jmp.

Формы команды jmp

Команда jmp имеет несколько форм, в зависимости от типа перехода:

1. Внутрисегментный прямой переход:
   * jmp метка: Переход к метке в пределах текущего сегмента памяти.
   * Пример: jmp m1, где m1: — метка.
2. Внутрисегментный косвенный переход:
   * jmp регистр: Переход к адресу, хранящемуся в регистре.
   * Пример: jmp rbx, где rbx содержит адрес перехода.
3. Внутрисегментный косвенный переход через переменную:
   * jmp переменная: Переход к адресу, хранящемуся в переменной.
   * Пример: jmp retPtr, где retPtr — переменная, содержащая адрес перехода.
4. Межсегментный прямой переход:
   * jmp far метка: Переход к метке в другом сегменте памяти.
   * Пример: jmp far m1, где m1: — метка в другом сегменте.
5. Межсегментный косвенный переход:
   * jmp far [переменная]: Переход к адресу, хранящемуся в переменной, в другом сегменте.
   * Пример: jmp far [Old], где [Old] — переменная, содержащая адрес перехода в другом сегменте.

Примеры использования

Пример 1: Внутрисегментный прямой переход

.code  
main proc  
 mov rdx, 11  
 jmp m1 ; Переход к метке m1  
 mov rdx, 22 ; Не выполняется  
m1:  
 mov rax, rdx  
 ret  
main endp  
end

Пример 2: Внутрисегментный косвенный переход

.code  
main proc  
 mov rbx, offset m1 ; Загрузка адреса метки m1 в rbx  
 jmp rbx ; Переход к адресу из rbx  
 mov rdx, 22 ; Не выполняется  
m1:  
 mov rax, rdx  
 ret  
main endp  
end

Пример 3: Внутрисегментный косвенный переход через переменную

.code  
main proc  
 retPtr qword offset m1 ; retPtr хранит адрес метки m1  
 jmp retPtr ; Переход к адресу из retPtr  
 mov rdx, 22 ; Не выполняется  
m1:  
 mov rax, rdx  
 ret  
main endp  
end

Пример 4: Межсегментный прямой переход

.code  
main proc  
 jmp far m1 ; Переход к метке m1 в другом сегменте  
 ; Дополнительный код, который не будет выполнен  
m1:  
 ; Код в другом сегменте  
 ret  
main endp  
end

Пример 5: Межсегментный косвенный переход

.code  
main proc  
 Old dd 0 ; Для логического адреса перехода  
 mov word ptr [Old], SEG m1 ; Установка сегмента  
 mov dword ptr [Old+2], offset m1 ; Установка смещения  
 jmp far [Old] ; Переход к адресу в другом сегменте  
 ; Дополнительный код, который не будет выполнен  
m1:  
 ; Код в другом сегменте  
 ret  
main endp  
end

**Подробно про условные переходы:**

Важные моменты

* Условные переходы могут выполнять только ближние прыжки (в пределах -128 до +127 байт от текущей команды).
* Для больших переходов используются комбинации условных и безусловных переходов.
* Условные переходы замедляют выполнение программы из-за предсказания переходов в конвейере процессора.

Основные команды условных переходов

Условные переходы основаны на состоянии флагов процессора, таких как флаг нуля (ZF), флаг переноса (CF), флаг переполнения (OF), флаг знака (SF), флаг четности (PF). Вот основные команды условных переходов:

* ZF (Флаг нуля):
  + je/jz метка: Переход, если результат равен нулю (ZF=1).
  + jne/jnz метка: Переход, если результат не равен нулю (ZF=0).
* CF (Флаг переноса):
  + jc/jnae/jb метка: Переход, если есть перенос (CF=1).
  + jnc/jae/jnb метка: Переход, если нет переноса (CF=0).
* OF (Флаг переполнения):
  + jo метка: Переход, если есть переполнение (OF=1).
  + jno метка: Переход, если нет переполнения (OF=0).
* SF (Флаг знака):
  + js метка: Переход, если знак отрицательный (SF=1).
  + jns метка: Переход, если знак положительный (SF=0).
* PF (Флаг четности):
  + jp метка: Переход, если четность четная (PF=1).
  + jnp метка: Переход, если четность нечетная (PF=0).
* Сравнения для беззнаковых чисел:
  + ja/jnbe метка: Переход, если выше (CF=0 и ZF=0).
  + jbe/jna метка: Переход, если ниже или равно (CF=1 или ZF=1).
  + jb/jnae метка: Переход, если ниже (CF=1).
  + jnb/jae метка: Переход, если выше или равно (CF=0).
* Сравнения для знаковых чисел:
  + jg/jnle метка: Переход, если больше (ZF=0 и SF=OF).
  + jle/jng метка: Переход, если меньше или равно (ZF=1 или SF<>OF).
  + jl/ngt метка: Переход, если меньше (SF<>OF).
  + jge/jnl метка: Переход, если больше или равно (SF=OF).

Пример использования

text

mov eax, 5

cmp eax, 10

jl less\_than ; Переход, если eax меньше 10

; Код, если eax не меньше 10

jmp end

less\_than:

; Код, если eax меньше 10

end:

; Дальнейший код

В этом примере управление передается на метку less\_than, если значение в eax меньше 10.

**Подробно про циклы:**

**Типы Циклов**

**1. Циклы с помощью команды LOOP**

Команда LOOP используется для организации циклов, где количество итераций задается в регистре CX (или ECX для 32-битных систем). Команда LOOP автоматически уменьшает значение в CX на единицу и выполняет переход к метке, пока CX не станет равным нулю.

Пример:

.model small  
.stack 128  
.data  
 mas db 10 dup(?)  
.code  
start:  
 mov cx, 10 ; Количество итераций  
 mov si, 0 ; Индекс для массива  
loop\_start:  
 ; Код цикла  
 inc si  
 loop loop\_start  
end start

**2. Циклы с помощью условных переходов**

Циклы также можно реализовать с помощью условных переходов (JNE, JNZ, JE, JZ и т.д.) и регистра CX. Этот метод требует ручного уменьшения значения в CX и проверки условия для выхода из цикла.

Пример:

.model small  
.stack 128  
.data  
 mas db 10 dup(?)  
.code  
start:  
 mov cx, 10 ; Количество итераций  
loop\_start:  
 ; Код цикла  
 dec cx  
 cmp cx, 0  
 jne loop\_start  
end start

**3. Циклы с помощью команд LOOPE/LOOPZ и LOOPNE/LOOPNZ**

* **LOOPE/LOOPZ**: Выполняют цикл, пока CX не равно нулю и флаг нуля (ZF) равен 1.
* **LOOPNE/LOOPNZ**: Выполняют цикл, пока CX не равно нулю и флаг нуля (ZF) равен 0.

Пример:

.model small  
.stack 128  
.data  
 mas db 10 dup(?)  
.code  
start:  
 mov cx, 10 ; Количество итераций  
loop\_start:  
 ; Код цикла  
 loopne loop\_start  
end start

**Преимущества и Недостатки**

* **Преимущества**:
  + Команда LOOP упрощает реализацию циклов, автоматически уменьшая счетчик и проверяя условие выхода.
  + Циклы с условными переходами позволяют реализовать более сложные условия для выхода из цикла.
* **Недостатки**:
  + Команда LOOP ограничена только ближними переходами (в пределах -128 до +127 байт).
  + Циклы с условными переходами требуют больше кода и могут быть менее эффективными, чем команда LOOP.

**Реализация на Практике**

Циклы широко используются в задачах, таких как обработка массивов, поиск элементов, сортировка данных и т.д. Например, для нахождения суммы элементов в массиве можно использовать цикл:

.model small  
.stack 128  
.data  
 mas db 10 dup(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
 sum dw 0  
.code  
start:  
 mov cx, 10 ; Количество элементов  
 mov si, 0 ; Индекс для массива  
 xor ax, ax ; AX для суммы  
loop\_start:  
 add ax, mas[si]  
 inc si  
 loop loop\_start  
 mov sum, ax  
end start

Таким образом, циклы являются фундаментальным инструментом в программировании на ассемблере, позволяя реализовать различные алгоритмы и задачи.