ГБПОУ ВО "Борисоглебский техникум промышленных и информационных технологий"

Основы Flat Assembler

Материалы по дисциплине «Микропроцессорные системы»

Специальность «Компьютерные системы и комплексы»

Составитель: Торгашин Р.Г

Оглавление

Основные понятия языка

Компьютерная программа, записанная на машинном языке, состоит из машинных инструкций, каждая из которых представлена в машинном коде в виде т. н. опкода — двоичного кода отдельной операции из системы команд машины. Каждая модель процессора имеет свой собственный набор команд, хотя во многих моделях эти наборы команд сильно перекрываются. Говорят, что процессор A совместим с процессором B, если процессор A полностью «понимает» машинный код процессора B. Если процессоры A и B имеют некоторое подмножество инструкций, по которым они взаимно совместимы, то говорят, что они одной «архитектуры» (имеют одинаковую архитектуру набора команд).

Программа «Hello, world!» для процессора архитектуры x86 (ОС MS DOS, вывод при помощи BIOS прерывания int 10h) выглядит следующим образом (в шестнадцатеричном представлении):

BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD 20 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21[1]

В двоичном представлении только первые 5 байт будут выглядеть следующим образом:

1011 1011 0001 0001 0000 0001 1011 1001 0000 1101

Для удобства вместо кодов обычно используется символический язык (язык ассемблера), в котором каждая команда процессора представляется символическим именем, и именами регистров, которые в ней используются. В этом случае программа записывается хоть и длиннее, но более понятно:

```
.386
model tiny                      ;Указание модели памяти
Code segment use16              ;Начало описания сегмента кода
ASSUME cs:Code, ds:Code     ;Ассоциация регистров с сегментом
   org    100h                  ;Генерация смещения на 256 байт
start:                          ;Метка начала программы
   push    cs                   ;Запись регистра CS в стек
   pop    ds                    ;Загрузка регистра DS значением из стека
   mov    dx, offset mess       ;Помещение в DS смещения строки mess
   mov    ah, 09h               ;Запись в AH номера функции вывода строки
   int    21h                   ;Вызов сервиса MS-DOS
   int    20h                   ;Завершение COM программы в MS-DOS
mess    db    'Hello world!','$‘ ;Объявление строки
Code ends                       ;Завершение описания строки
end start
```
[2]

Ассемблер - транслятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке.

Как и сам язык, ассемблеры, как правило, специфичны для конкретной архитектуры, операционной системы и варианта синтаксиса языка. Вместе с тем существуют мультиплатформенные или вовсе универсальные (точнее, ограниченно-универсальные, потому что на языке низкого уровня нельзя написать аппаратно-независимые программы) ассемблеры, которые могут работать на разных платформах и операционных системах. Среди последних можно также выделить группу кросс-ассемблеров, способных собирать машинный код и исполняемые модули (файлы) для других архитектур и ОС.

Зарезервированные слова

Слова, значения которых заранее определены в семантическом пространстве ассемблера. Они могут использоваться только для соответствующих целей.

Использование зарезервированного слова не по назначению приведет к тому, что ассемблер выдаст сообщение об ошибке.

Эли слова делятся четыре категории:

---

[1] https://ru.wikipedia.org/wiki/Машинный_код

[2] http://learnprogramm.ucoz.ru/index/hello_world_na_assemblere_dlja_dos_com_exe_i_windows/0-101

Инструкции — указывают на операции, которые может выполнять компьютер. Например ADD, MOV;
Директивы — используются для изменения поведения транслятора ассемблера. Например SEGMENT, END;
Операторы — используются в составе выражений. Например FAR, SIZE;
Предопределенные символы — возвращают информацию при трансляции. Например @Data, @Model.

Комментарии

Комментарии упрощают понимание написанного кода. Особенно это актуально для программ на Ассемблере, где назначение наборов инструкций не всегда понятно. Например, ясно, что MOV AH,10h помещает 10h в регистр AH. Однако зачем это делается - не понятно. Еще более важно комментировать код при написании программ для микроконтроллеров, так как в этом случае значение наборов инструкций может зависеть от архитектуры микроконтроллера.

Комментарий начинается с символа **« ; ».** Все символы, стоящие в строке после этого символа воспринимаются как комментарий.

Например:

```
; подпрограмма получения данных об оборудовании
PUSH DS          ; сохранить адрес DS в стеке
MOV AX, 0040     ; получить адрес сегмента
MOV DS, AX       ; поместить адрес сегмента в DS
MOV AX,[100]     ; поместить данные из 40:10 в AX
POP DS           ; восстановить адрес в DS из стека
IRET             ; вернуться в исходную программу
```

Обратите внимание на оформление кода. Написание инструкций и регистров заглавными символами не является обязательным с точки зрения синтаксиса ассемблера. Однако это улучшает читаемость кода и является общепринятым стандартом оформления. То же относится к выравниванию комментариев в единый столбец при помощи «Tab» и написанию их строчными символами.

Комментарии не включаются в состав генерируемого кода. Поэтому объем комментариев в листинге исходного кода не влияет на объем результирующего кода.

Идентификаторы

Идентификаторы — это имена. назначаемые элементам программы, на которые нужно сослаться. Понятие идентификатора в языке ассемблера ничем не отличается от понятия идентификатора в других языках.

Есть два типа идентификаторов

Имя — ссылается на адрес или элемент данных. Например COUNTER DB 0.

Метка — ссылается на адрес инструкции. Например MAIN PROC FAR или B30: ADD BL,25

Идентификаторы могут содержать буквы латинского алфавита, цифры (но не могут начинаться с цифры), знаки ? , _ , $, @ , точка (но не могут начинаться с точки)[3].

Собственные идентификаторы начинающиеся с @ использовать не рекомендуется.

По умолчанию ассемблер не различает строчные и заглавные буквы.

Максимальная длинна идентификатора: до 31 символа в MASM 6.0 и до 247 и более в поздних версиях. Рекомендуется использовать значимые, осмысленные идентификаторы.

Предложения — основная структурная единица программы на ассемблере. Существует два типа предложений: инструкции и директивы.

Таблица 1 Структура предложения. В квадратных скобках указаны необязательные элементы

[<метка>]	***<операция>***	***[<операнд (ы)>]***	***[; <комментарий>]***

[3] Ограничения, связанные с использованием точек, цифр и т.п. в именах идентификаторов могут различаться в зависимости от диалекта ассемблера.

```
        ADD     ax, bx      ; суммирование значений ax=ax+bx
;Start: MOV     ax, bx      ; перемещение данных из bx в ax
        ret                 ; возвращение из подпрограммы
```

Все структурные части предложения разделаются хотя бы одним пробелом или символом табуляции.

В MASM 6.0 максимальная длинна строки 132 символа. В более поздних до 512. Оптимальной считается длинна строки не более 80 символов. Использование длинных строк ухудшает читаемость кода.

Примеры предложений:

Директивы объявления данных (объявления переменных).

В ассемблере существуют директивы объявления данных.

Размер (в байтах)	**Объявление**	**Резервирование**
1	db	rb
2	dw du	rw
4	dd	rd
6	dp df	rp rf
8	dq	rq
10	dt	rt
N	file	

За **директивой резервирования данных** должно следовать одно числовое выражение, значение которого определяет количество резервируемых ячеек установленного размера. Она создает не инициализированные данные. Их можно использовать в программе для хранения временного или промежуточного значения. Фактически под переменную просто резервируется место в памяти.

С неинициализированными переменными следует быть внимательным. Не надо рассчитывать, что по умолчанию значение будет нулевым или ещё каким-то, иначе это может привести к ошибке.

Директива file включает цепь байтов из файла. В качестве параметра за ней должно идти в кавычках имя файла, далее, опционально, двоеточие и числовое выражение, указывающее начало цепочки байтов, далее, также опционально, запятая и числовое выражение, определяющее количество байтов в этой цепочке (если этот параметр не определен, то будут включены все данные до конца файла).

Синтаксис объявления данных

Объявление
[<имя>] <директива> <значение>
Резервирование
[<имя>] <директива> <число ячеек>

Объявлять данные очень просто — например, чтобы объявить байт со значением 5 достаточно написать:

x db 5

где x — название нашей переменной или константы, db — директива объявления байта, а 5 — значение. С помощью названия в программе можно будет обращаться к ячейке памяти, содержащей наш байт.

```
x db 5
y dw 5
MOV al, x      ;переместить данные из байта x в ax
MOV bx, y      ;переместить данные из байта y в bx
```

Обратите внимание, что содержимое X мы помещаем в регистр AL а не AX. Дело в том, что мы определяли X директивой DB и размер X - 1 байт. Разрядность регистра AX - 16 bit, то есть 2 байта и помещение в него одного байта может привести к ошибке. За такими моментам нужно постоянно следить. Для подобного действия есть специальная команда movzx. Например *movzx bx,[x] ;*

Вообще, название не обязательно и можно его не писать, если оно не требуется:

db 5

Объявление последовательностей (массивов)

Иногда в программе требуется объявить массив, то есть несколько переменных одинакового размера, расположенных в памяти друг за другом. Например, чтобы объявить массив из 5 двухбайтных чисел можно написать:

array1 dw 1,2,3,4,5

где array1 — название массива, 1,2,3,4,5 — значения элементов. Вместо array1 компилятор FASM будет подставлять в программу адрес начала массива, то есть адрес первого элемента.

Дамп памяти будет выглядеть следующим образом (обратите внимание, младший байт каждого слова расположен перед старшим):

Для объявления повторяющихся элементов можно использовать такую запись (объявляем массив из 5 байтов, равных 1):

array2 db 5 dup(1)

Объявление строк

Строка представляет собой массив байтов-символов и записывается в одинарных кавычках:

str1 db 'Hello'

Для обозначения конца строки используется специальный символ. Обычно это нулевой байт, но для функций DOS используется символ ’$’.

```
str2 db 'Hello',0    ;Обычно так
str3 db 'Hello$'     ;Для DOS
```

Запись чисел

По умолчанию, число в программе воспринимается ассемблером как десятичное.
Чтобы обозначить двоичное число, необходимо к нему в конце добавить символ ’b’.
Восьмеричное число обозначается аналогично с помощью символа ’o’.
Для записи шестнадцатеричного числа FASM поддерживает 3 формы записи:

- перед числом записываются символы ’0x’ (как в C/C++);
- перед числом записывается символ ’$’ (как в Pascal);
- после числа записывается символ ’h’.

Если шестнадцатеричное число начинается с буквы, необходимо добавить в начале ноль (иначе непонятно, число это или имя метки).

Этот синтаксис используется как при объявлении данных, так и в командах. Вот примеры записи чисел во всех четырёх системах:

```
mov ax,537        ;Десятичная система

mov bl,11010001b    ;Двоичная система

mov ch,57o        ;Восьмеричная система

mov dl,$C2        ;\
mov si,0x013A      ; \
mov ah,18h        ; / Шестнадцатеричная система
mov al,0FFh        ;/

mov al,FFh        ;Ошибка!
```

Для записи отрицательного числа в программе на ассемблере используется символ ‘-‘, например:

```
x db -5
```

Это работает и с числами в других системах счисления, и даже с символами

```
y db -25h
z db -77o
k db -101b
s db -'a'
```

Со знаковыми и беззнаковыми числами нужно быть внимательным! Один и тот же байт может интерпретироваться по-разному, в зависимости от того со знаком число или без. Например, числу со знаком -5 соответствует число без знака 251:

Диапазоны значений чисел со знаком и без

Размер переменной	Число без знака		Число со знаком	
	min	max	min	max

байт	00000000	11111111	10000000	01111111
	0	255	-128	127
слово	00000000 00000000	11111111 11111111	10000000 00000000	01111111 11111111
	0	65 535	-32 768	32 767
Двойное слово	0000…0000	1111…1111	1000…0000	0111…1111
	0	4 294 967 295	-2 147 483 648	2 147 483 647
и т.д.	…	…	…	…

Система команд

В общем случае система команд процессора включает в себя следующие четыре основные группы команд:

- команды пересылки данных;
- арифметические команды ;
- логические команды ;
- команды переходов.

Команды пересылки данных не требуют выполнения никаких операций над операндами. Операнды просто пересылаются (точнее, копируются) из источника (Source) в приемник (Destination). Источником и приемником могут быть внутренние регистры процессора, ячейки памяти или устройства ввода/вывода. АЛУ в данном случае не используется.

Арифметические команды выполняют операции сложения, вычитания, умножения, деления, увеличения на единицу (инкрементирования), уменьшения на единицу (декрементирования) и т.д. Этим командам требуется один или два входных операнда. Формируют команды один выходной операнд.

Логические команды производят над операндами логические операции, например, логическое И, логическое ИЛИ, исключающее ИЛИ, очистку, инверсию, разнообразные сдвиги (вправо, влево, арифметический сдвиг, циклический сдвиг). Этим командам, как и арифметическим, требуется один или два входных операнда, и формируют они один выходной операнд.

Команды переходов предназначены для изменения обычного порядка последовательного выполнения команд. С их помощью организуются переходы на подпрограммыи возвраты из них, всевозможные циклы, ветвления программ, пропуски фрагментов программ и т.д. Команды переходов всегда меняют содержимое счетчика команд. Переходы могут быть условными и безусловными. Именно эти команды позволяют строить сложные алгоритмы обработки информации.

В соответствии с результатом каждой выполненной команды устанавливаются или очищаются биты регистра состояния процессора (PSW). Но надо помнить, что не все команды изменяют все имеющиеся в PSW флаги. Это определяется особенностями каждого конкретного процессора.

Имеется основной набор команд, который поддерживается всеми процессорами одного семейства. С усовершенствованием процессоров список команд расширяется. Добавляются команды, позволяющие использовать особенности архитектуры данного процессора но, обычно, сохраняется поддержка более ранних наборов команд. Поэтому программы, использующие «старые» наборы команд также могут функционировать. Но с меньшей производительностью.

При этом, на системах с различной (микро)архитектурой может быть реализована одна и та же система команд. Например, Intel Pentium и AMD Athlon имеют почти идентичные версии системы команд x86, но имеют радикально различный внутренний дизайн.

В то же время существуют процессоры с сокращенным набором команд (так называемые RISC-процессоры), в которых за счет максимального сокращения количества команд достигается увеличение эффективности и скорости их выполнения.

Команды пересылки данных

Команды пересылки данных занимают очень важное место в системе команд любого процессора. Они выполняют следующие важнейшие функции:

- загрузка (запись) содержимого во внутренние регистры процессора;
- сохранение в памяти содержимого внутренних регистров процессора;
- копирование содержимого из одной области памяти в другую;
- запись в устройства ввода/вывода и чтение из устройств ввода/вывода.

В некоторых процессорах (например, Т-11) все эти функции выполняются одной единственной командой MOV (для байтовых пересылок — MOVB) но с различными методами адресации операндов.

Команда MOV

Копирует operand2 в operand1.

Команда MOV не может:

записывать данные в регистры CS и IP.

копировать данные из одного сегментного регистра в другой сегментный регистр (сначала нужно скопировать данные в регистр общего назначения).

копировать непосредственное значение в сегментный регистр (сначала нужно скопировать данные в регистр общего назначения).

Пример:

```
#make_COM#
ORG 100h
MOV AX, 0B800h    ; установить AX = B800h (память VGA).
MOV DS, AX        ; копировать значение из AX в DS.
MOV CL, 'A'       ; CL = 41h (ASCII-код).
MOV CH, 01011111b ; CL = атрибуты цвета.
MOV BX, 15Eh      ; BX = позиция на экране.
MOV [BX], CX      ; w.[0B800h:015Eh] = CX.
RET               ; вернуться в операционную систему.
```

Часто выделяются специальные команды для сохранения в стеке и для извлечения из стека (PUSH — сохранить в стеке, POP — извлечь из стека). Эти команды выполняют пересылку с автоинкрементной и с автодекрементной адресацией (даже если эти режимы адресации не предусмотрены в процессоре в явном виде).

Иногда в систему команд вводится специальная команда MOVS для строчной (или цепочечной) пересылки данных (например, в процессоре 8086). Эта команда пересылает не одно слово или байт, а заданное количество слов или байтов (MOVSB), то есть инициирует не один цикл обмена по магистрали, а несколько. При этом адрес памяти, с которым происходит взаимодействие, увеличивается на 1 или на 2 после каждого обращения или же уменьшается на 1 или на 2 после каждого обращения. То есть в неявном виде применяется автоинкрементная или автодекрементная адресация.

Команда MOVSB

Копирует байт из DS:[SI] в ES:[DI]. Изменяет регистры SI и DI.

Алгоритм:

ES:[DI] = DS:[SI]

если DF = 0 то

- *SI = SI + 1*
- *DI = DI + 1*
- *иначе*
- *SI = SI - 1*
- *DI = DI - 1*

Пример:

```
#make_COM#
ORG 100h
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP MOVSB
RET
```

Команда MOVSW
Копирует слово из DS:[SI] в ES:[DI]. Изменяет регистры SI и DI.
Алгоритм:
- *ES:[DI] = DS:[SI]*
- *если DF = 0 то*
  - *SI = SI + 2*
  - *DI = DI + 2*

  иначе
  - *SI = SI - 2*
  - *DI = DI - 2*

Пример:

```
#make_COM#
ORG 100h
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP MOVSW
RET
```

В некоторых процессорах (например, в процессоре 8086) специально выделяются функции обмена с устройствами ввода/вывода. Команда IN используется для ввода (чтения) информации из устройства ввода/вывода, а команда OUT используется для вывода (записи) в устройство ввода/вывода. Обмен информацией в этом случае производится между регистром-аккумулятором и устройством ввода/вывода.

Команда IN

Помещает данные из порта в AL или AX.

Второй операнд - это номер порта. Если требуется доступ к порту с номером более 255, то нужно использовать регистрDX.

Пример:

```
IN AX, 4  ; получить состояние светофора.
IN AL, 7  ; получить состояние шагового дигателя.
```

Команда OUT

Выводит данные из регистра AL или AX в порт. Первый операнд - номер порта. Если требуется доступ к порту, номер которого превышает 255, то должен быть использован регистр DX.

Пример:

```
MOV AX, 0FFFh ; Включить все
OUT 4, AX     ; светофоры.
MOV AL, 100b  ; Включть третий магнит
OUT 7, AL     ; шагового двигателя.
```

Арифметические команды

Арифметические команды рассматривают коды операндов как числовые двоичные или двоично-десятичные коды.

Команда ADD

Сложение

Алгоритм:

operand1 = operand1 + operand2

Пример:

```
MOV AL, 5   ; AL = 5
ADD AL, -3  ; AL = 2
RET
```

Команда SUB

Вычитание
Алгоритм:
operand1 = operand1 - operand2
Пример:
MOV AL, 5
SUB AL, 1 ; AL = 4
RET

Команда MUL

Беззнаковое умножение.
Алгоритм:
если операнд - byte: AX = AL * операнд.
если операнд - word: (DX AX) = AX * операнд.
Пример:
MOV AL, 200 ; AL = 0C8h
MOV BL, 4
MUL BL ; AX = 0320h (800)
RET

Команда DIV

Беззнаковое деление.
Алгоритм:
если операнд - это байт:
AL = AX / операнд
AH = остаток (модуль)
если операнд - это слово:
AX = (DX AX) / операнд
DX = остаток (модуль)
Пример:
MOV AX, 203 ; AX = 00CBh
MOV BL, 4
DIV BL ; AL = 50 (32h), AH = 3
RET

Команды очистки (CLR) предназначены для записи нулевого кода в регистр или ячейку памяти. Эти команды могут быть заменены командами пересылки нулевого кода, но специальные команды очистки обычно выполняются быстрее, чем команды пересылки. Команды очистки иногда относят к группе логических команд, но суть их от этого не меняется.

Команда инкремента INC

Инкремент.
Алгоритм:
operand = operand + 1
Пример:
MOV AL, 4
INC AL ; AL = 5
RET

Команда декремента DEC

Декремент.
Алгоритм:

operand = operand - 1
Пример:
MOV AL, 255 ; AL = 0FFh (255 или -1)
DEC AL ; AL = 0FEh (254 или -2)
RET

Команды инкремента (увеличения на единицу, INC) и декремента (уменьшения на единицу, DEC) также бывают очень удобны. Их можно в принципе заменить командами суммирования с единицей или вычитания единицы, но инкремент и декремент выполняются быстрее, чем суммирование и вычитание. Эти команды требуют одного входного операнда, который одновременно является и выходным операндом.

Команда сравнения CMP

Сравнение.
Алгоритм:
operand1 - operand2
результат никуда не записывается, флаги устанавливаются (OF, SF, ZF, AF, PF, CF) в соответствии с результатом.
Пример:
MOV AL, 5
MOV BL, 5
CMP AL, BL ; AL = 5, ZF = 1 (значит равно!)
RET

Команда сравнения (обозначается CMP) предназначена для сравнения двух входных операндов. По сути, она вычисляет разность этих двух операндов, но выходного операнда не формирует, а всего лишь изменяет биты в регистре состояния процессора (PSW) по результату этого вычитания. Следующая за командой сравнения команда (обычно это команда перехода) будет анализировать биты в регистре состояния процессора и выполнять действия в зависимости от их значений (о командах перехода речь идет в разделе 3.3.4). В некоторых процессорах предусмотрены команды цепочечного сравнения двух последовательностей операндов, находящихся в памяти (например, в процессоре 8086 и совместимых с ним).

Логические команды

Логические команды выполняют над операндами логические (побитовые) операции, то есть они рассматривают коды операндов не как единое число, а как набор отдельных битов. Этим они отличаются от арифметических команд. Логические команды выполняют следующие основные операции:

- логическое И, логическое ИЛИ, сложение по модулю 2 (Исключающее ИЛИ);
- логические, арифметические и циклические сдвиги;
- проверка битов и операндов;
- установка и очистка битов (флагов) регистра состояния процессора (PSW).

Команды логических операций позволяют побитно вычислять основные логические функции от двух входных операндов. Кроме того, операция И (AND) используется для принудительной очистки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие очистки, установлены в нуль).

Команда AND

Логическое И между всеми битами двух операндов. Результат записывается в 1-й операнд.
Действуют следующие правила:
1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0

0 AND 0 = 0
Пример:
MOV AL, 'a' ; AL = 01100001b
AND AL, 11011111b ; AL = 01000001b ('A')
RET

Команда OR

Логическое ИЛИ между всеми битами двух операндов. Результат записывается в первый операнд.

Выполняются следующие правила:
1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0
Пример:
MOV AL, 'A' ; AL = 01000001b
OR AL, 00100000b ; AL = 01100001b ('a')
RET

Операция ИЛИ (OR) применяется для принудительной установки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие установки в единицу, равны единице).

Команда XOR

Логическое XOR (Исключающее ИЛИ)между всеми битами двух операндов. Результат записывается в первый операнд.

Выполняются следующие правила:
1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0
Пример:
MOV AL, 00000111b
XOR AL, 00000010b ; AL = 00000101b
RET

Операция "Исключающее ИЛИ" (XOR) используется для инверсии заданных битов (в качестве одного из операндов при этом применяется код маски, в котором биты, подлежащие инверсии, установлены в единицу). Команды требуют двух входных операндов и формируют один выходной операнд.

Команда ROL

Циклический сдвиг (ротация) влево. Количество ротаций устанавливается во втором операнде.

Алгоритм:

самый левый бит записать во флаг CF, сдвинуть все биты влево, в самый правый бит записать флаг CF.

Пример:
MOV AL, 1Ch ; AL = 00011100b
ROL AL, 1 ; AL = 00111000b, CF=0.
RET

Команда RCL

Циклический сдвиг (ротация) влево через перенос. Количество ротаций устанавливается во втором операнде.

Если immediate больше единицы, ассемблер генерирует несколько команд RCL xx, 1, потому что 8086 имеет машинный код только для этой команды (тот же принцип работы используют все команды сдвига/ротации).

Алгоритм:

самый левый бит записать во флаг CF, сдвинуть все биты влево, значение флага CF записать в самый правый бит (бит 0).

Пример:

```
STC             ; установить перенос (CF=1).
MOV AL, 1Ch     ; AL = 00011100b
RCL AL, 1       ; AL = 00111001b,  CF=0.
RET
```

Команда SAL

Арифметический сдвиг влево. Количество сдвигов записывается во второй операнд.

Алгоритм:

- самый левый бит записать в CF, сдвинуть все биты влево,
- в самый правый бит записать ноль.

Пример:

```
MOV AL, 0E0h    ; AL = 11100000b
SAL AL, 1       ; AL = 11000000b,  CF=1.
RET
```

Команда SHL

Сдвиг влево. Количество сдвигов указывается во втором операнде. Знаковый бит рассматривается как обычный бит данных.

Алгритм:

- Записать самый левый бит в CF, сдвинуть все биты влево.
- В самый правый бит записать ноль.

Пример:

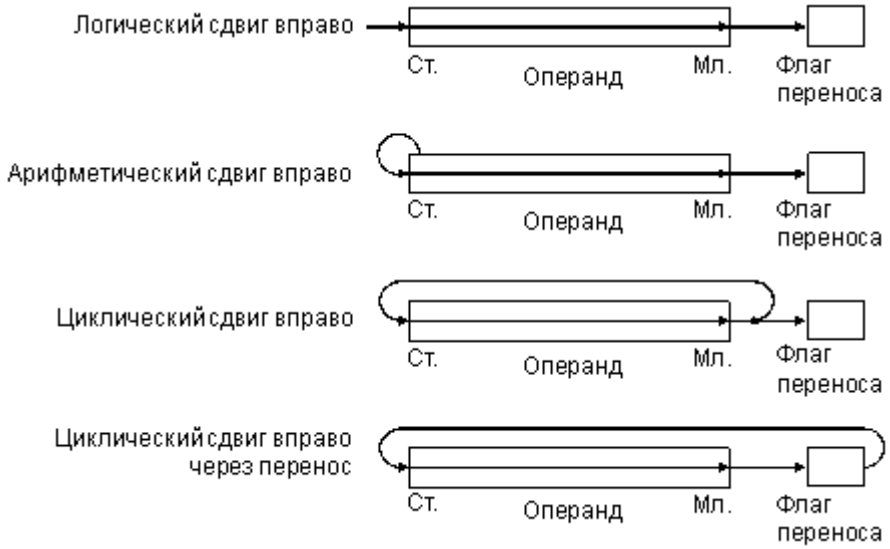
```
MOV AL, 11100000b
SHL AL, 1       ; AL = 11000000b,  CF=1.
RET
```

Команды сдвигов позволяют побитно сдвигать код операнда вправо (в сторону младших разрядов) или влево (в сторону старших разрядов). Тип сдвига (логический, арифметический или циклический) определяет, каково будет новое значение старшего бита (при сдвиге вправо) или младшего бита (при сдвиге влево), а также определяет, будет ли где-то сохранено прежнее значение старшего бита (при сдвиге влево) или младшего бита (при сдвиге вправо). Например, при логическом сдвиге вправо в старшем разряде кода операнда устанавливается нуль, а младший разряд записывается в качестве флага переноса в регистр состояния процессора. А при арифметическом сдвиге вправо значение старшего разряда сохраняется прежним (нулем или единицей), младший разряд также записывается в качестве флага переноса.

Циклические сдвиги позволяют сдвигать биты кода операнда по кругу (по часовой стрелке при сдвиге вправо или против часовой стрелки при сдвиге влево). При этом в кольцо сдвига может входить или не входить флаг переноса. В бит флага переноса (если он используется) записывается значение старшего бита при циклическом сдвиге влево и младшего бита при циклическом сдвиге вправо. Соответственно, значение бита флага переноса будет переписываться в младший разряд при циклическом сдвиге влево и в старший разряд при циклическом сдвиге вправо.

Для примера на рисунке показаны действия, выполняемые командами сдвигов вправо.

Команды проверки битов и операндов предназначены для установки или очистки битов регистра состояния процессора в зависимости от значения выбранных битов или всего операнда в целом. Выходного операнда команды не формируют. Команда проверки операнда (TST) проверяет весь код операнда в целом на равенство нулю и на знак (на значение старшего бита), она требует только одного входного операнда. Команда проверки бита (BIT) проверяет только отдельные биты, для выбора которых в качестве второго операнда используется код маски. В коде маски проверяемым битам основного операнда должны соответствовать единичные разряды.



Наконец, команды установки и очистки битов регистра состояния процессора (то есть флагов) позволяют установить или очистить любой флаг, что бывает очень удобно. Каждому флагу обычно соответствуют две команды, одна из которых устанавливает его в единицу, а другая сбрасывает в нуль. Например, флагу переноса C (от Carry) будут соответствовать команды CLC (очистка) и SEC или STC (установка).

Команды переходов

Команды переходов предназначены для организации всевозможных циклов, ветвлений, вызовов подпрограмм и т.д., то есть они нарушают последовательный ход выполнения программы. Эти команды записывают в регистр-счетчик команд новое значение и тем самым вызывают переход процессора не к следующей по порядку команде, а к любой другой команде в памяти программ.

Некоторые команды переходов предусматривают в дальнейшем возврат назад, в точку, из которой был сделан переход, другие не предусматривают этого. Если возврат предусмотрен, то текущие параметры процессора сохраняются в стеке. Если возврат не предусмотрен, то текущие параметры процессора не сохраняются.

Команды переходов без возврата делятся на две группы:

- команды безусловных переходов;
- команды условных переходов.

В обозначениях этих команд используются слова Branch (ветвление) и Jump (прыжок).

Команды безусловных переходов вызывают переход в новый адрес независимо ни от чего. Они могут вызывать переход на указанную величину смещения (вперед или назад) или же на указанный адрес памяти. Величина смещения или новое значение адреса указываются в качестве входного операнда.

Команда JMP

Безусловный переход. Передает управление другому участку программы. 4-х байтовый адрес может быть введен в такой форме: 1234h:5678h, первое значение - сегмент, второе значение – смещение.

Алгоритм:

выполнить переход в любом случае

Пример:

```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
JMP label1    ; "перешагнуть" через две строки!
PRINT 'Нет перехода!'
MOV AL, 0
label1:
PRINT 'Добрались сюда!'
RET
```

Команды условных переходов вызывают переход не всегда, а только при выполнении заданных условий. В качестве таких условий обычно выступают значения флагов в регистре состояния процессора (PSW). То есть условием перехода является результат предыдущей операции, меняющей значения флагов. Всего таких условий перехода может быть от 4 до 16. Несколько примеров команд условных переходов:

- переход, если равно нулю;
- переход, если не равно нулю;
- переход, если есть переполнение;
- переход, если нет переполнения;
- переход, если больше нуля;
- переход, если меньше или равно нулю.

Если условие перехода выполняется, то производится загрузка в регистр-счетчик команд нового значения. Если же условие перехода не выполняется, счетчик команд просто наращивается, и процессор выбирает и выполняет следующую по порядку команду.

Команда JAE

Короткий переход, если первый операнд "больше или равен" второму операнду. (в результате выполнения команды CMP). Беззнаковый.

Алгоритм:

если CF = 0 то выполнить переход

Пример:

```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
CMP AL, 5
JAE label1
PRINT 'AL не больше 5'
JMP exit
label1:
PRINT 'AL больше или равен 5'
exit:
RET
```

Команда JC

Короткий переход если флаг переноса установлен в 1.
Алгоритм:
если CF = 1 то выполнить переход
Пример:

```
  include 'emu8086.inc'
  #make_COM#
  ORG 100h
  MOV AL, 255
  ADD AL, 1
  JC  label1
  PRINT 'нет переноса.'
  JMP exit
label1:
  PRINT 'имеем перенос.'
exit:
  RET
```

Специально для проверки условий перехода применяется команда сравнения (CMP), предшествующая команде условного перехода (или даже нескольким командам условных переходов). Но флаги могут устанавливаться и любой другой командой, например командой пересылки данных, любой арифметической или логической командой. Отметим, что сами команды переходов флаги не меняют, что как раз и позволяет ставить несколько команд переходов одну за другой.

Совместное использование нескольких команд условных и безусловных переходов позволяет процессору выполнять разветвленные алгоритмы любой сложности. Для примера на рис. 2 показано разветвление программы на две ветки с последующим соединением, а на рис. 3 — разветвление на три ветки с последующим соединением.

Команды переходов с дальнейшим возвратом в точку, из которой был произведен переход, применяются для выполнения подпрограмм, то есть вспомогательных программ. Эти команды называются также командами вызова подпрограмм (распространенное название — CALL).

Команда CALL

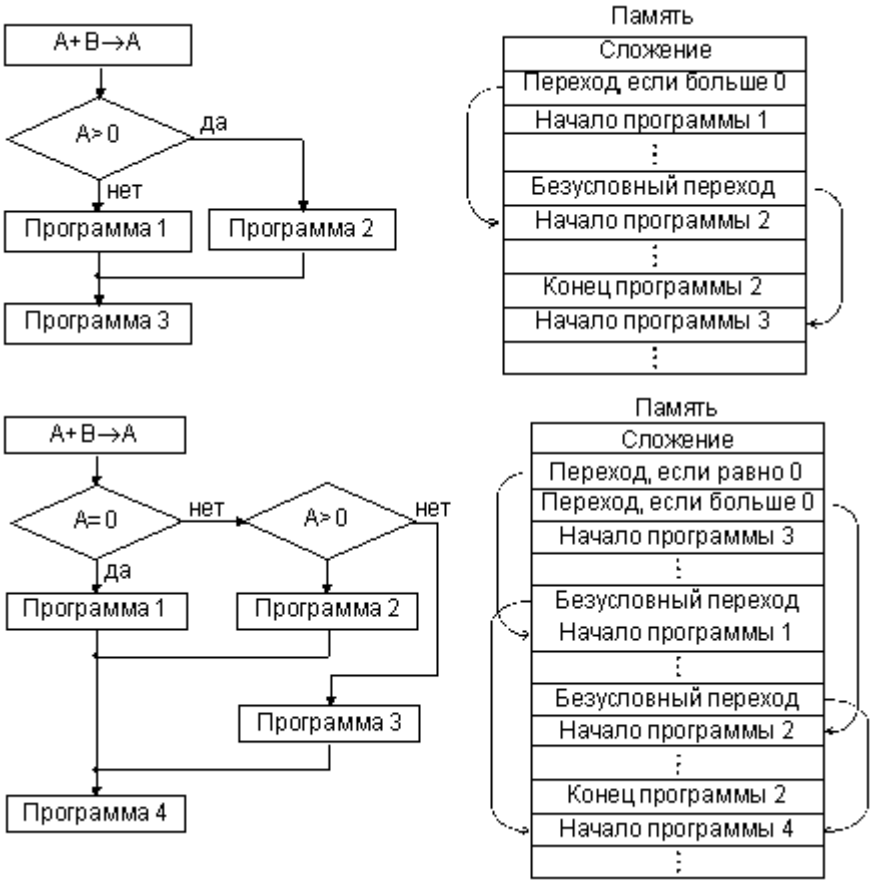
Передает управление процедуре, заносит в стек адрес следующей команды (из IP). 4-х байтовый адрес может быть введен в следующей форме: 1234h:5678h, первое значение - сегмент, второе значение - смещение (в случае дальнего вызова регистр CS также заносится в стек).

Пример:

```
#make_COM#
ORG 100h  ; для COM-файла.
CALL p1
ADD AX, 1
RET       ; вернуться в операционную систему.

p1 PROC    ; объявление процедуры.
   MOV AX, 1234h
   RET     ; возвращение в программу.
p1 ENDP
```

Использование подпрограмм позволяет упростить структуру основной программы, сделать ее более логичной, гибкой, легкой для написания и отладки. В то же время надо учитывать, что широкое использование подпрограмм, как правило, увеличивает время выполнения программы.



Все команды переходов с возвратом предполагают безусловный переход (они не проверяют никаких флагов). При этом они требуют одного входного операнда, который может указывать как абсолютное значение нового адреса, так и смещение, складываемое с текущим значением адреса. Текущее значение счетчика команд (текущий адрес) сохраняется перед выполнением перехода в стеке.

Команда RET

Возврат из ближней процедуры.
Алгоритм:

- Получить из стека:
  - IP
- если имеется операнд immediate: SP = SP + операнд

Пример:

```
#make_COM#
ORG 100h  ; для COM-файла.
CALL p1
ADD AX, 1
RET       ; вернуться в операционную систему.

p1 PROC    ; объявление процедуры.
   MOV AX, 1234h
   RET     ; вернуться в программу.
p1 ENDP
```

Для обратного возврата в точку вызова подпрограммы (точку перехода) используется специальная команда возврата (RET или RTS). Эта команда извлекает из стека значение адреса команды перехода и записывает его в регистр-счетчик команд.

Команда INT

Выполняет прерывание программы и передает управление функции, указанной в immediate byte (0..255).

Алгоритм:

Поместить в стек:

- флаговый регистр
- CS
- IP

• IF = 0

• Передать управление процедуре прерывания

Пример:

```
MOV AH, 0Eh  ; телетайп.
MOV AL, 'A'
INT 10h      ; Прерывание BIOS.
RET
```

Особое место среди команд перехода с возвратом занимают команды прерываний (распространенное название — INT). Эти команды в качестве входного операнда требуют номер прерывания (адрес вектора). Обслуживание таких переходов осуществляется точно так же, как и аппаратных прерываний. То есть для выполнения данного перехода процессор обращается к таблице векторов прерываний и получает из нее по номеру прерывания адрес памяти, в который ему необходимо перейти. Адрес вызова прерывания и содержимое регистра состояния процессора (PSW) сохраняются в стеке. Сохранение PSW — важное отличие команд прерывания от команд переходов с возвратом.

Команды прерываний во многих случаях оказываются удобнее, чем обычные команды переходов с возвратом. Сформировать таблицу векторов прерываний можно один раз, а потом уже обращаться к ней по мере необходимости. Номер прерывания соответствует номеру подпрограммы, то есть номеру функции, выполняемой подпрограммой. Поэтому команды прерывания гораздо чаще включаются в системы команд процессоров, чем обычные команды переходов с возвратом.

Команда IRET

Возврат из обработки прерывания.

Алгоритм:

Выгрузить из стека:

- IP
- CS
- регистр флагов

Для возврата из подпрограммы, вызванной командой прерывания, используется команда возврата из прерывания (IRET или RTI). Эта команда извлекает из стека сохраненное там значение счетчика команд и регистра состояния процессора (PSW).

Отметим, что у некоторых процессоров предусмотрены также команды условных прерываний, например, команда прерывания при переполнении.

Конечно, в данном разделе мы рассмотрели только основные команды, наиболее часто встречающиеся в процессорах. С более полным списком команд процессора 8086 можно ознакомится здесь: полный набор команд процессора 8086

У конкретных процессоров могут быть и многие другие команды, не относящиеся к перечисленным группам команд. Но изучать их надо уже после того, как выбран тип процессора, подходящий для задачи, решаемой данной микропроцессорной системой.

Источники

1. Учебный курс программирования на ассемблере FASM - Url: http://asmworld.ru/uchebnik/
2. FLAT ASSEMBLER 1.64 - МАНУАЛ ПРОГРАММЕРА - Url: http://flatassembler.narod.ru/fasm.htm
3. Полный набор команд процессора 8086 — Url http://www.avprog.narod.ru/progs/emu8086/8086_instruction_set.html
4. Основы микропроцессорной техники. Новиков Юрий Витальевич, Скоробогатов Петр Константинович — Url: http://www.intuit.ru/studies/courses/3/3/info