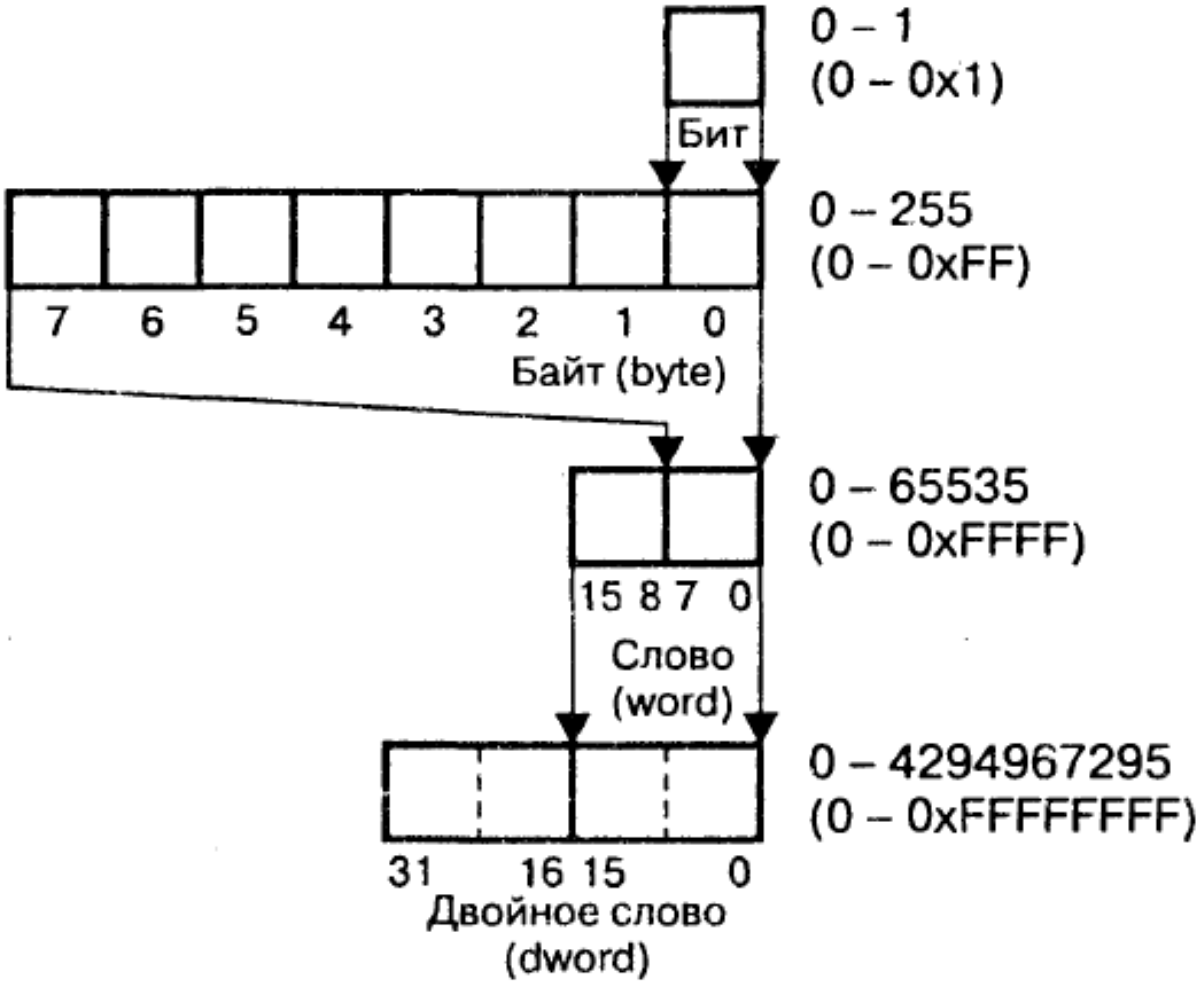


Рис. 1 От бита до двойного слова

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00		☺ 1	☻ 2	♥ 3	♦ 4	♣ 5	♠ 6	• 7	8	○ 9	10	11	12	13	14	15
10	► 16	◄ 17	18	! 19	20	21	‗ 22	23	↑ 24	↓ 25	→ 26	← 27	28	↔ 29	▲ 30	▼ 31
20	32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
30	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
40	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
50	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
60	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
70	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	⌂ 127
80	Ç 128	ü 129	é 130	â 131	ä 132	à 133	å 134	ç 135	ê 136	ë 137	è 138	ï 139	î 140	ì 141	Ä 142	Å 143
90	É 144	æ 145	Æ 146	ô 147	ö 148	ò 149	û 150	ù 151	ÿ 152	Ö 153	Ü 154	¢ 155	£ 156	¥ 157	₧ 158	ƒ 159
A0	160	í 161	ó 162	ú 163	ñ 164	Ñ 165	ª 166	º 167	¿ 168	⌐ 169	¬ 170	½ 171	¼ 172	¡ 173	« 174	» 175
B0	░ 176	▒ 177	▓ 178	│ 179	┤ 180	╡ 181	╢ 182	╖ 183	╕ 184	╣ 185	║ 186	╗ 187	╝ 188	╜ 189	╛ 190	┐ 191
C0	└ 192	┴ 193	┬ 194	├ 195	─ 196	┼ 197	╞ 198	╟ 199	╚ 200	╔ 201	╩ 202	╦ 203	╠ 204	═ 205	╬ 206	╧ 207
D0	╨ 208	╤ 209	╥ 210	╙ 211	╘ 212	╒ 213	╓ 214	╫ 215	216	┘ 217	┌ 218	▉ 219	▄ 220	▌ 221	▐ 222	▀ 223
E0	α 224	ß 225	Γ 226	π 227	Σ 228	σ 229	µ 230	τ 231	Φ 232	Θ 233	Ω 234	δ 235	∞ 236	φ 237	ε 238	∩ 239
F0	≡ 240	± 241	≥ 242	≤ 243	⌠ 244	⌡ 245	÷ 246	≈ 247	° 248	∙ 249	· 250	√ 251	ⁿ 252	² 253	■ 254	□ 255

Рис. 2

Таблица кодов ASCII

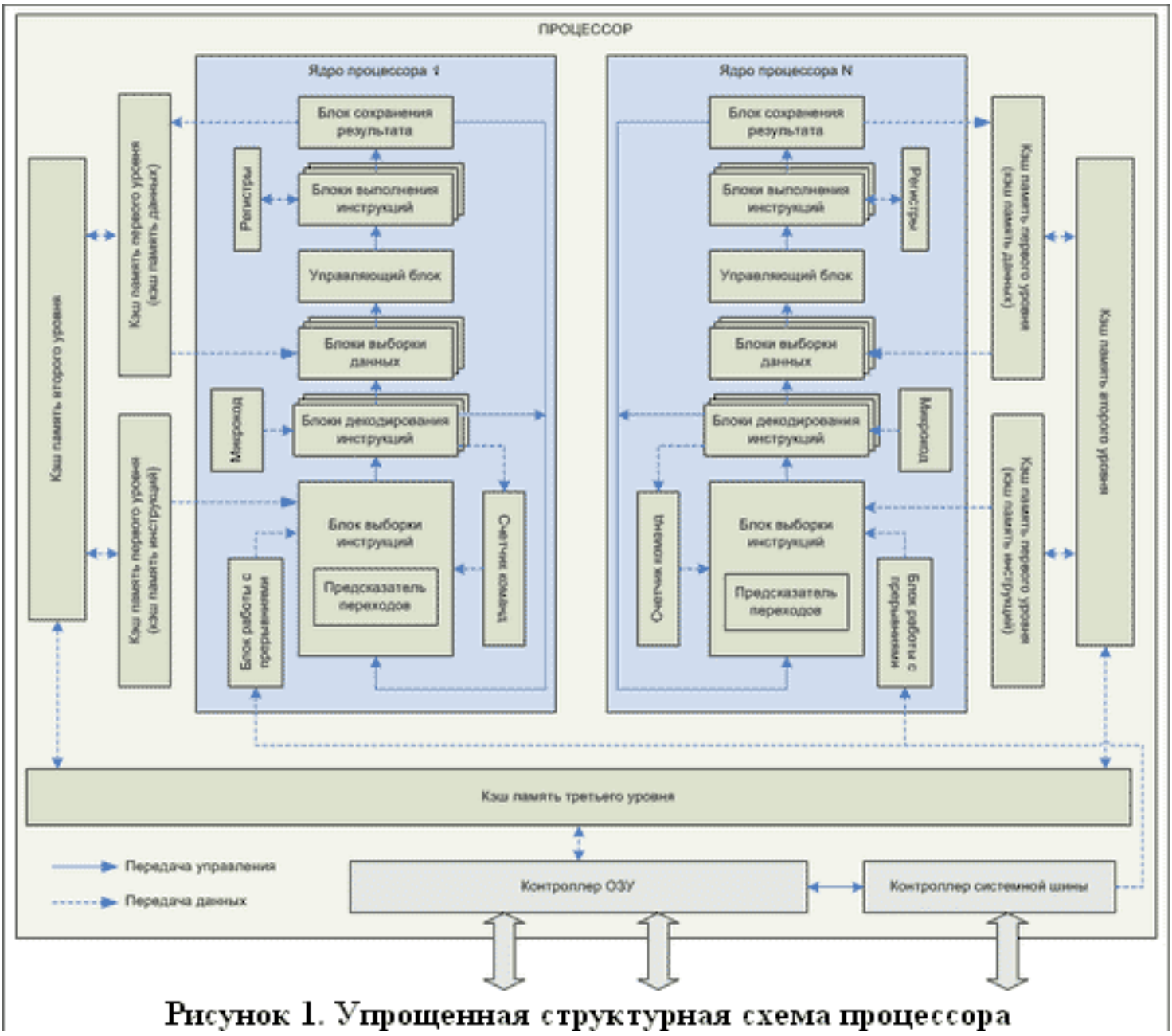
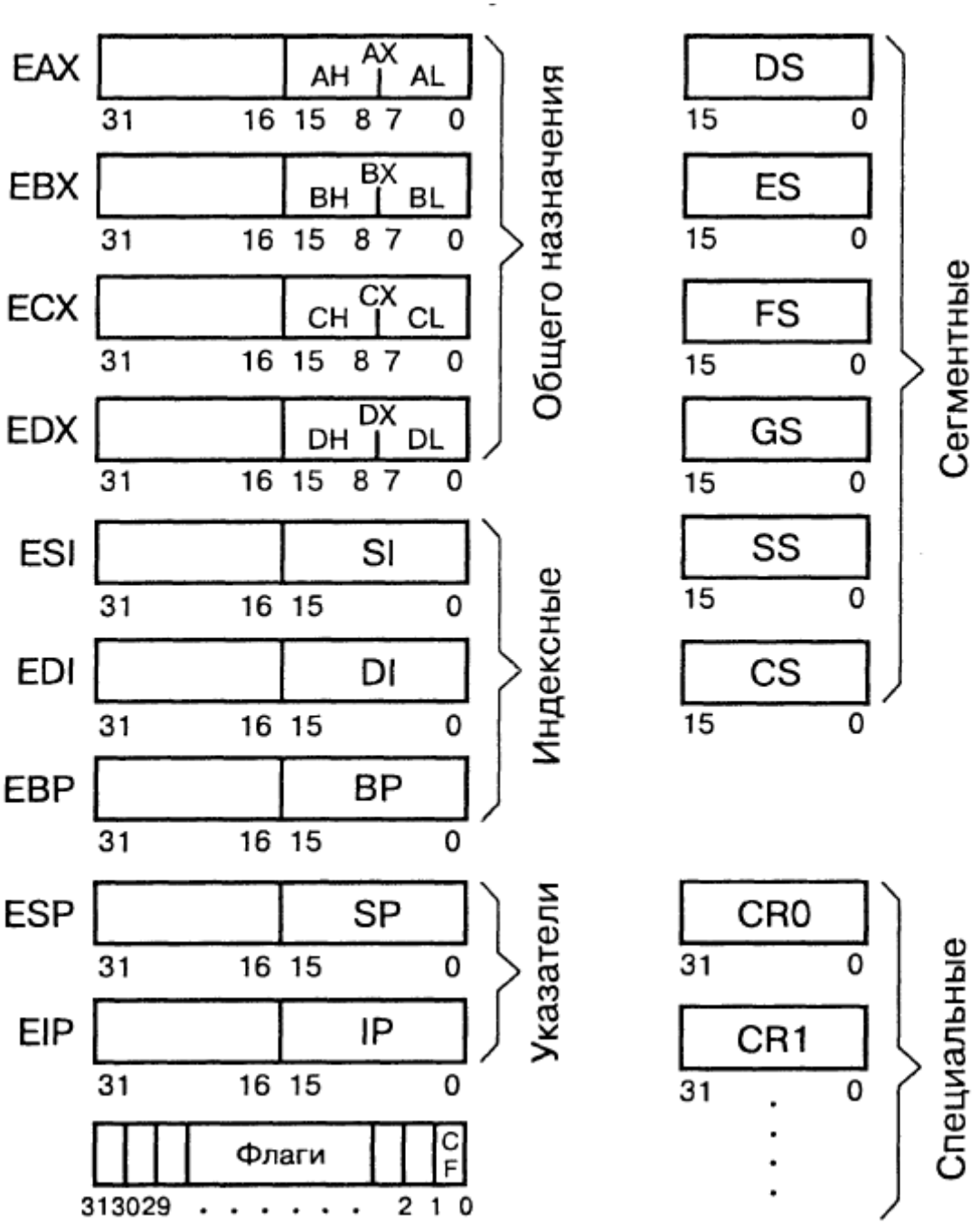


Рисунок 1. Упрощенная структурная схема процессора



Индексные регистры

К регистрам общего назначения иногда относят и индексные регистры процессора 80386 —• ESI, EDI и EBP (или SI, DI и BP для 16-разрядных действий).

Обычно эти регистры используются для адресации памяти: обращения к массивам, индексирования и т.д. Отсюда их имена: индекс источника (Source Index), индекс приемника (Destination Index), указатель базы (Base Pointer).

Но хранить в них только адреса совсем необязательно: регистры ESI, EDI и EBP могут содержать произвольные данные. Эти регистры программно доступны, то есть их содержание может быть изменено программистом.

У регистров ESI, EDI и EBP существуют только в 16-разрядная и 32-разрядная версии.

Сегментные регистры

В программной модели микропроцессора имеется шесть сегментных регистров: cs, ss, ds, es, gs, fs.

Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых сегментами. Соответственно такая организация памяти называется сегментной.

Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены сегментные регистры. Фактически (с небольшой поправкой) в этих регистрах содержатся адреса памяти, с которыхначинаются соответствующие сегменты. Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку неявно используются адреса во вполне определенных сегментных регистрах.

Микропроцессор поддерживает следующие типы сегментов.

1. Сегмент кода. Содержит команды программы. Для доступа к этому сегменту служит регистр cs (code segment register) – сегментный регистр кода. Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (т. е. эти команды загружаются в конвейер микропроцессора).

2. Сегмент данных. Содержит обрабатываемые программой данные. Для доступа к этому сегменту служит регистр ds (data segment register) – сегментный регистр данных, который хранит адрес сегмента данных текущей программы.

3. Сегмент стека. Этот сегмент представляет собой область памяти, называемую стеком. Работу со стеком микропроцессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этому сегменту служит регистр ss (stack segment register) – сегментный регистр стека, содержащий адрес сегмента стека.

4. Дополнительный сегмент данных. Не явно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре ds. Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре ds, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в регистрах es, gs, fs (extension data segment registers).

CS — Сегмент кода (используется для IP);
DS — Сегмент данных (используется для MOV);
ES — дополнительный сегментный регистр данных или экстракодов (Целевой сегмент (используется для MOVS и т.д.));
SS — сегмент стека (используется для SP).

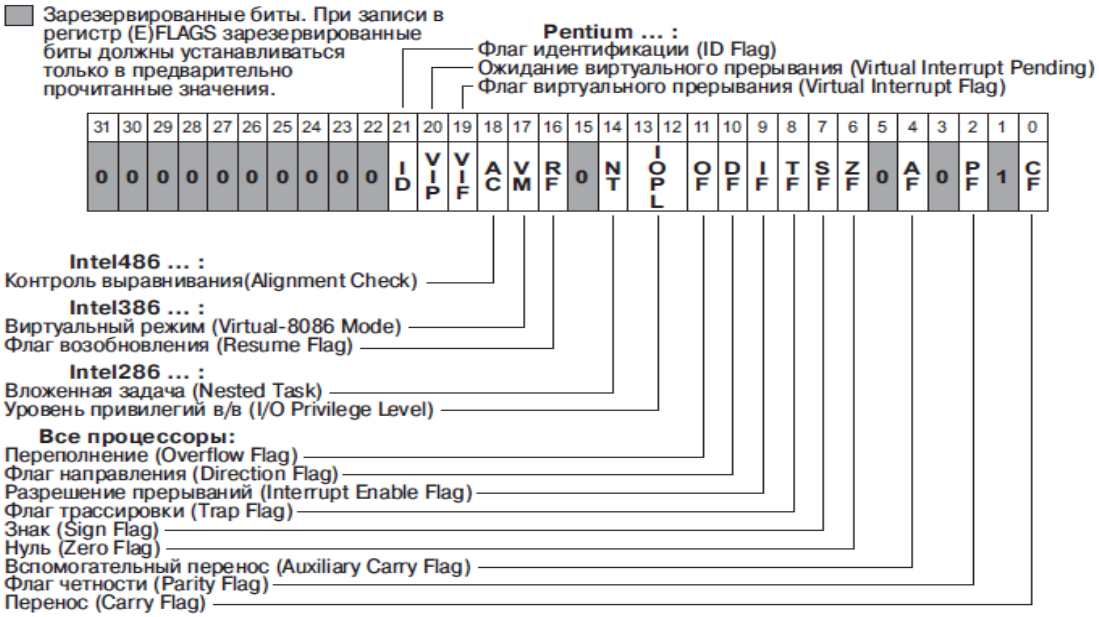
Эту группу регистров можно отнести к регистрам состояния. Регистры из этой группы используются при вычислении реального адреса (адреса, который будет передан на шину адреса). Процесс вычисления реального адреса зависит от режима процессора (реальный или защищенный). Сегментные регистры только 16-разрядные, такие же, как в 80286.

Названия этих регистров соответствуют выполняемым функциям: CS (Code Segment, сегмент кода) вместе с EIP (IP) определяют адрес памяти, откуда нужно прочитать следующую инструкцию; аналогично регистр SS (Stack Segment, сегмент стека) в паре с ESP (SS:SP) указывают на вершину стека. Сегментные регистры DS, ES, FS, и GS (Data, Extra, F и G сегменты) используются для адресации данных в памяти.

Регистры состояния и управления

Регистр ESP (SP) — это указатель памяти, который указывает на вершину стека (x86-совместимые процессоры не имеют аппаратного стека). Также программно не может быть изменен регистр EIP (IP, Instruction Pointer) — указатель команд. Этот регистр указывает на инструкцию, которая будет выполнена следующей. Значение этого регистра изменяется непосредственно контроллером процессора согласно инструкциям, полученным из памяти.

Регистр флагов (иногда его называют регистром признаков) — EFLAGS. Он состоит из одноразрядных флагов, отображающих в основном текущее состояние арифметико-логического устройства.



CF (Флаг переноса, бит 0)
Флаг переноса фиксирует значение переноса (заема), возникающего при сложении (вычитании). Иногда используется и в других ситуациях.
PF (Флаг четности, бит 2)
Фиксирует наличие четного числа единиц в младшем байте результата операции, может быть использован, например, для контроля правильности передачи данных.
AF (Флаг вспомогательного переноса, бит 4)
Фиксирует перенос (заем) из младшей тетрады, т.е. из бита 3 в старшую тетраду при сложении (вычитании). Используется только для двоично-десятичной арифметики, которая оперирует исключительно младшими байтами.
ZF (Флаг нуля, бит 6)
Сигнализирует о получении нулевого (ZF = 1) или ненулевого (ZF = 0) результата операции.
SF (Флаг знака, бит 7)
Дублирует значение старшего бита результата, который при использовании дополнительного кода соответствует знаку числа (0 – положительное число, 1 – отрицательное).
TF (Флаг трассировки, бит 8)
При TF = 1, микропроцессор переходит в пошаговый режим работы, применяемый при отладке программ, когда автоматически генерируется особая ситуация отладки (#DB) после выполнения каждой команды. Прерывание отладки начнет генерироваться, если прикладная программа установит флаг TF с помощью команд POPF, POPFD или IRET.
IF (Флаг разрешения прерываний, бит 9)
При IF = 1, микропроцессор воспринимает (распознает) и соответственно реагирует на запрос прерывания по входу INTR#; при IF = 0, прерывания по этому входу запрещаются и микропроцессор игнорирует поступающие запросы прерываний. Значение флага IF не влияет на восприятие внешних немаскируемых прерываний по входу NMI#, а также внутренних программных прерываний, выполняемых по команде INT. В защищенном режиме изменение этого флага командами CLI, STI, POPF, POPFD и IRET возможно не всегда и определяется текущим уровнем привилегий выполняемого кода (CPL) и уровнем привилегий ввода/вывода (IOPL).
DF (Флаг направления, бит 10)
Флаг направления определяет порядок обработки цепочек в соответствующих командах (строковые команды: STOS, LODS, CMPS, OUTS и т.д.) — от меньших адресов к большим (DF = 0) или от больших к меньшим (DF = 1).
OF (Флаг переполнения, бит 11)
Флаг переполнения сигнализирует о потере старшего бита результата в связи с переполнением разрядной сетки при работе со знаковыми числами. При сложении этот флаг устанавливается в 1, если происходит перенос в старший бит и нет переноса из старшего бита, или имеется перенос из старшего бита, но отсутствует перенос в него; в противном случае, флаг OF устанавливается в 0. При вычитании

он устанавливается в 1, когда возникает заем из старшего бита, но заем в старший бит отсутствует, либо имеется заем в старший бит, но отсутствует заем из него.
IOPL (Уровень привилегий ввода/вывода, биты 13 и 12: Intel286 …)
Уровень привилегий ввода/вывода используется в механизме защиты для управления доступом к адресному пространству ввода/вывода. Текущий уровень привилегий (CPL) совместно с IOPL определяет возможность по изменению поля IOPL командами POPF, POPFD и IRET.
NT (Вложенная задача, бит 14: Intel286 …)
Процессор устанавливает и проверяет этот флаг для контроля за прерванными задачами и при вызове процедур. Флаг NT влияет на действия, производимые командой IRET. Этот флаг может быть изменен командой POPF, POPFD и IRET. Некорректные изменения этого флага могут привести к возникновению различных особых ситуаций в прикладных программах.
RF (Флаг возобновления, бит 16: Intel386 …)
Флаг RF временно выключает обработку особых ситуаций отладки для того, чтобы команда, вызвавшая такую ситуацию, могла быть перезапущена и не стала бы причиной новой особой ситуации. Отладчик устанавливает этот флаг командой IRETD при возврате в прерванную программу. Команды POPF, POPFD (в режиме V86) и IRET на этот флаг не влияют.
VM (Виртуальный режим, бит 17: Intel386 …)
Установка флага VM переключает процессор в режим виртуального-8086 (специальный случай защищенного режима).
AC (Режим контроля выравнивания, бит 18: Intel486 …)
Установка флага AC и бита AM регистра CR0 включает контроль выравнивания при обращении к памяти. При этом генерируется особая ситуация контроля выравнивания (#AC), если происходит обращение к невыровненному операнду, например, к слову по нечетному адресу или к двойному слову по адресу не кратному четырем. Особая ситуация контроля выравнивания генерируется только при уровне привилегий равном 3.
VIF (Виртуальное прерывание, бит 19: Pentium …)
Этот флаг используется в специальном расширенном режиме обработки прерываний (виртуальные прерывания управляются флагом CR4.VME) и является виртуальным подобием флага IF. В зависимости от текущего значения этого флага в режиме V86 и в защищенном режиме (когда IOPL < 3, CPL = 3) процессор определенным образом обрабатывает вызовы внешних маскируемых прерываний (векторы от 32 до 255). Флаг VIF используется совместно с флагом VIP и позволяет обеспечить нормальное выполнение старого ПО, использующего команды управления маскируемыми прерываниями, в современной мультипроцессорной и мультизадачной программно-аппаратной среде.
VIP (Ожидание виртуального прерывания, бит 20: Pentium …)
Флаг VIP используется совместно с флагом VIF и позволяет прикладным программам в режиме поддержки виртуальных прерываний отслеживать внешние вызовы прерываний даже тогда, когда программа замаскирует их выполнение (индикация отложенных виртуальных прерываний). За дополнительной информацией по использованию этих флагов обратитесь к описанию прерываний и особых ситуаций.
ID (Флаг идентификации, бит 21: Pentium …)
Предназначен для проверки — поддерживается ли процессором команда CPUID. Если в программе можно установить и сбросить этот флаг, значит команда CPUID данным процессором поддерживается.

Регистры общего назначения

```
EAX = (16+AX=(AH+AL))
EBX = (16+BX=(BH+BL))
ECX = (16+CX=(CH+CL))
EDX = (16+DX=(DH+DL))
ESI = (16+SI)
EDI = (16+DI)
EBP = (16+BP)
ESP = (16+SP)
```

MOV приемник, источник

Примеры применения команды

```
mov ax,[number]       ;заносим значение переменной number
                      ;в регистр AX
mov [number],bx       ;загрузить значение регистра BX
                      ;в переменную number
mov bx,cx             ;занести в регистр BX значение
                      ;регистра CX
mov al,1              ;занести в регистр AL значение 1
mov dh,cl             ;занести в регистр DH значение
                      ;регистра CL
mov esi,edi           ;копировать значение регистра EDI
                      ;в регистр ESI
mov word [number],1   ;сохранить 16-битное значение 1
                      ;в переменную "number"

  mov [number_two], [number_one]   ;НЕПРАВИЛЬНО!!!

mov ax, [number_one]   ;загружаем в AX 16-битное
                       ;значение "number_one"
mov [number_two], ax   ;а затем копируем его в переменную
                       ;"number_two"

mov ax, bl              ;НЕПРАВИЛЬНО! – Операнды разных
                          ;размеров.

 mov ax, bx             ;загружаем BX в AX
 mov ah, 0              ;"сбрасываем" верхнюю часть
                        ;AX – записываем в нее 0

  mov ax, 0             ;AH = 0, AL = 0
  mov al, bl            ;заносим в AL значение BL
```

11111011 00000101

00000101

```
MOV r/m8,reg8
MOV r/m16,reg16
MOV r/m32,reg32
MOV reg8,r/m8
MOV reg16,r/m16
MOV reg32,r/m32
MOV reg8,imm8
MOV reg16,imm16
MOV reg32,imm32
MOV r/m8,imm8
MOV r/m16,imm16
MOV r/m32,imm32
```

Список допустимых форматов команд

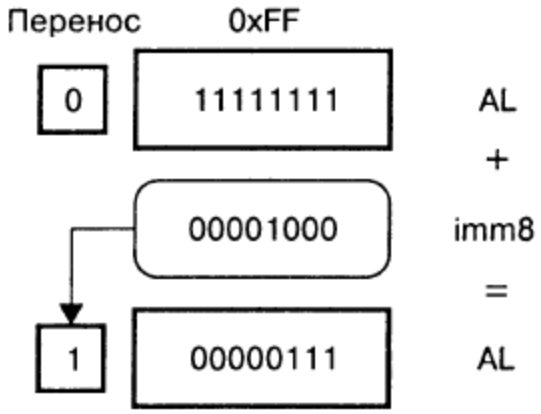
```
ADD o1, o2
SUB o1, o2
```

```
mov ax, 8          ;заносим в AX число 8
mov cx, 6          ;заносим в CX число 6
mov dx, cx         ;копируем CX в DX, DX = 6
add dx, ax         ;DX = DX + AX
```

mov ax, 8
add ax, 6

```
add eax,8                ;EAX = EAX +  8
sub ecx,ebp              ;ECX = ECX - EBP
add byte [number], 4     ;добавляем значение 4
                         ;к переменной number
                         ;размером в 1 байт
                         ;(диапазон значений 0-255)
sub word [number], 4     ;number = number - 4
                         ;размером в 2 байта
                         ;(диапазон значений 0-65535)
add dword [number], 4    ;добавляем значение 00000004
                         ;к "number"
sub byte [number], al    ;вычитаем значение регистра AL
                         ;из "number"
sub ah,al                ;вычитаем AL из AH, результат
                         ;помещаем в AH
```

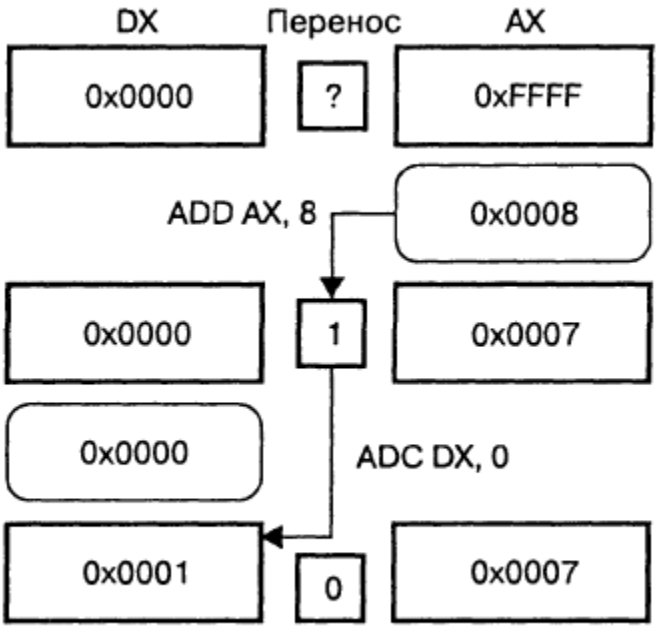
```
mov al, 255        ;заносим в AL значение 255, то есть 0xFF
add al, 8          ;добавляем 8
```



Сложение 255 (0xFF) + 8

```
ADC o1, o2    ;o1 = o1 + o2 + CF
SBB o1, o2     ;o1 = o1 – o2 – CF
```

```
mov ax, 0xffff      ;AX = 0xFFFF
mov dx, 0           ;DX = 0
add ax, 8           ;AX = AX + 8
adc dx, 0           ;добавляем 0 с переносом к DX
```



Сложение чисел 0xFFFF и 0x0008 и их сохранение в регистрах

```
mov eax, 0xffff     ;EAX = 0xFFFF
add eax, 8          ;EAX = EAX + 8
```

Использование 32-разрядных регистров процессора

```
INC o1              ;o1 = o1 + 1
DEC o1              ;o1 = o1 – 1

add al, 1           ;AL = AL + 1
inc al              ;AL = AL + 1

inc word [number]  ;мы должны указать размер
                   ;переменной – word
```

Отрицательные числа

```
mov ax,0xFFFA      ;AX = -6, то есть 65530 или 0xFFFA
mov dx,7           ;DX = 7
add ax,dx          ;AX = AX + DX

mov ax, -6          ;AX = -6
mov dx,-6           ;DX = -6
add ax,dx           ;AX = AX + DX

mov ax,[bx-1]      ;поместить в AX значение по адресу
                   ;на единицу меньшему, чем хранится в BX

NEG r/m8
NEG r/m16
NEG r/m32
```

```
neg eax             ;изменяет знак числа, сохраненного в EAX
neg bl              ;то же самое, но используется 8-битный
                    ;регистр bl
neg byte [number]   ;изменяет знак 8-битной переменной number
```

```
mov al, bh          ;AL = BH — с н а ч а л а заносим в AL второй операнд
mul c1              ;AL = AL * CL — умножаем его на CL

mov ax, 486             ; AX = 486
mul ax                  ; AX * AX -> DX:AX

imul edx , ecx          ;EDX - EDX * ECX

imul ebx, [s t l i i n g]   ;умножает 3 2-разрядную переменную "stiling" на EBX, результат
                            будет сохранен в EBX
imul ecx, б                 ;ECX - ECX * б

imul edx, ecx, l                ;EDX = ECX * 7

imul ebx, [s t h i n g] , 9 ;умножаем переменную "sthing" на 9,

imul ecx, edx, -11          ;результат будет сохранен EBX
                            ;ECX = EDX * 11
```