Машинно-зависимые языки программирования

Савельев Игорь Леонидович

План

- Арифметические операции
  - Сдвиг
  - Двоично-десятичная арифметика
  - Логические операции

0. Арифметические операции. Умножение двух больших чисел

```
1.   MOV EAX, [X]
2.   MOV EBX, EAX          ; сохраняем для шага 6
3.   MUL DWORD [Y]         ; перемножить младшие двойные слова
4.   MOV [Z],  EAX         ; сохранить младшее слово произведения
5.   MOV ECX, EDX          ; сохранить старшее двойное слово
6.   MOV EAX, EBX          ; младшее слово "X" в eax
7.   MUL DWORD [Y+4]       ; умножить младшее слово на старшее
8.   ADD EAX, ECX
9.   ADC EDX, 0            ; добавить перенос
10.  MOV EBX, EAX          ; сохранить частичное произведение
11.  MOV ECX, EDX
12.  MOV EAX, [X+4]
13.  MUL DWORD [Y]         ; умножить старшее слово на младшее
14.  ADD EAX, EBX          ; сложить с частичным произведением
15.  MOV  [Z+4], EAX
16.  ADC ECX, EDX
17.  MOV EAX,  [X+4]
18.  MUL DWORD [Y+4]       ; умножить старшие слова
19.  ADD EAX, ECX          ; сложить с частичным произведением
20.  ADC EDX, 0            ; и добавить перенос
21.  MOV  [Z+8], EAX
22.  MOV  [Z+12], EDX
23.  ret
```

```
section .data
   X DQ 0x0000000100000002
   Y DQ 0x0000000300000004
   Z DQ 0, 0
```

```
   1 2
   3 4
   ---
   0 8
 0 6
 0 6     (AX+ CX)
 0 6     (+ CF)
 0 4
 0 A     (AX + BX)
 0 A     (+ CF)
0 3
0 3      (AX + CX)
0 3      ( + CF)
---
0 3 A 8
```

```
[L:1] AX = 2
[L:2] BX = 2
[L:3-5] CX = DX = 0, AX = 8 (2*4)
[L:7]DX = 0, AX = 6 (2*3)
[L:8](0 6 + 0 0)
[L:9]CX = DX = 0, BX = AX  = 6
[L:13] DX = 0, AX = 4 (1 * 4)
[L:14-15]
[L:16] CX = 0
[L:18] DX = 1, AX = 3 (1*3)
[L:19]
[L:20]
[L:21-22]
```

0 – результат умножения, 6 – результат сложения

Жирный шрифт (**8, A, 0, 3**) – сохранение результата

[L:20] – номер строки кода

1. Сдвиговые операции

Команды сдвига перемещает все биты в поле данных либо вправо, либо влево, работая либо с байтами, либо со словами.

Каждая команда содержит два операнда:
первый операнд – **поле данных** – может быть либо регистром, либо ячейкой памяти;
второй операнд – **счетчик сдвигов**.
Его значение может быть равным 1, или быть произвольным. В последнем случае это значение необходимо занести в регистр **CL**, который указывается в команде сдвига. Число в **CL** может быть в пределах 0-255, но его практически имеющие смысл значения лежат в пределах 0-16.

Общая черта всех команд сдвига – установка **флага переноса**. Бит, попадающий за пределы операнда, сохраняется во флаге переноса.

1. Сдвиговые операции

Всего существует 8 команд сдвига: 4 команды обычного сдвига и 4 команды циклического сдвига.

команды логического сдвига вправо **SHR** и влево **SHL**;
команды арифметического сдвига вправо **SAR** и влево **SAL**;
команды циклического сдвига вправо **ROR** и влево **ROL**;
команды циклического сдвига вправо **RCR** и влево **RCL** с переносом;

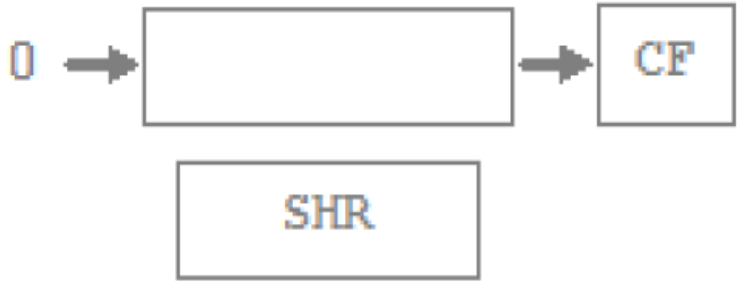
Команды циклического сдвига переносят появляющийся в конце операнда бит в другой конец, а в случае обычного сдвига этот бит пропадает.

1. Сдвиговые операции

Значение, вдвигаемое в операнд, зависит от типа сдвига.
При **логическом** сдвиге вдвигаемый бит всегда 0, **арифметический** сдвиг выбирает вдвигаемый бит таким образом, чтобы **сохранить знак** операнда.

Команды циклического сдвига с переносом и без него отличаются трактовкой флага переноса. **Первые** рассматривают его как **дополнительный 9-ый или 17-ый** бит в операции сдвига, а **вторые нет**.

1. Сдвиговые операции

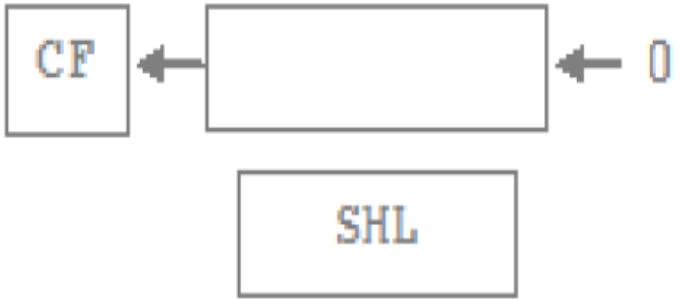



```
MOV CL, 03                  AX:
MOV AX,  10110111B ; 10110111
SHR AX, 1          ; 01011011        ; Сдвиг вправо на 1 , CF = 1
SHR AX, CL         ; 00001011        ; Сдвиг вправо на 3, CF = 0
```

Первая команда SHR сдвигает содержимое регистра AX вправо на 1 бит.
Выдвинутый в результате один бит попадает в флаг CF, а самый левый бит регистра AX заполняется нулем.

Вторая команда сдвигает содержимое регистра AX еще на три бита.
При этом флаг CF последовательно принимает значения 1,1, 0, а в три левых бита в регистре AX заносятся нули.

1. Сдвиговые операции




```
MOV CL, 04                  AX:
MOV AX,  10110111B ; 10110111
SHL AX, 1          ; 01101110      ; Сдвиг влево на 1 , CF = 1
SHL AX, CL         ; 11100000      ; Сдвиг влево на 4, CF = 0
```

1. Сдвиговые операции


```
MOV CL, 03                  AX:
MOV AX, 10110111B           ; 10110111
SAR AX, 1                   ; 11011011          ; Сдвиг вправо на 1, CF = 1
SAR AX, CL                  ; 11111011          ; Сдвиг вправо на 3, CF = 0

MOV CL, 03
MOV AX, 00110111B           ; 00110111
SAR AX, 1                   ; 00011011          ; Сдвиг вправо на 1, CF = 1
SAR AX, CL                  ; 00000011          ; Сдвиг вправо на 3, CF = 0
```

Команда **SAR** имеет важное отличие от команды **SHR**:
Для заполнения левого бита используется знаковый бит. Таким образом, положительные и отрицательные величины сохраняют свой знак.

1. Сдвиговые операции


```
MOV CL, 04                  AX:
MOV AX,  10110111B ; 10110111
SAL AX, 1          ; 01101110       ; Сдвиг влево на 1 , CF = 1
SAL AX, CL         ; 11100000       ; Сдвиг влево на 4, CF = 0
```

При сдвигах влево правые биты заполняются нулями. Таким образом, результат команд сдвига **SHL** и **SAL** идентичен.

1. Сдвиговые операции

SHL/SAL r	2	1 1 0 1 0 0 v w	mod * 0 0 r/m	Логический влево (Рг)
SHL/SAL mem	15+*E*			Логический влево (П)
SHR r	2	1 1 0 1 0 0 v w	mod 1 0 1 r/m	Логический вправо (Рг)
SHR mem	15+*E*			Логический вправо (П)
SAR r	2	1 1 0 1 0 0 v w	mod 1 1 1 r/m	Арифметический вправо (Рг)
SAR mem	15+*E*			Арифметический вправо (П)

D0	SAL *r/m8*,1	Арифметический сдвиг *r/m8* влево (*r/m8* = *r/m8* * 2)	8086	sal al,1
D2	SAL *r/m8*,CL	Арифметический сдвиг *r/m8* влево (*r/m8* = *r/m8* * 2^CL)	8086	sal ah,cl
C0	SAL *r/m8,imm8*	Арифметический сдвиг *r/m8* влево (*r/m8* = *r/m8* * 2^*imm8*)	8086	sal dh,4
D1	SAL *r/m16*,1	Арифметический сдвиг *r/m16* влево (*r/m16* = *r/m16* * 2)	8086	sal ax,1
D3	SAL *r/m16*,CL	Арифметический сдвиг *r/m16* влево (*r/m16* = *r/m16* * 2^CL)	8086	sal ax,cl
C1	SAL *r/m16,imm8*	Арифметический сдвиг *r/m16* влево (*r/m16* = *r/m16* * 2^*imm8*)	8086	sal dx,4

1. Сдвиговые операции

Сдвиг влево часто используется для **удваивания** чисел, а сдвиг вправо - для **деления** на **2**.

1000 = 8
0100 = 4
0010 = 2
0001 = 1

1111 = 15
0111 = 7 ; 15 = 7 * 2 + CF(1)
0011 = 3 ; 7 = 3 * 2 + CF(1)
0001 = 1 ; 3 = 1 * 2 + CF(1)

Эти операции осуществляются значительно **быстрее**, чем команды умножения или деления. Деление пополам нечетных чисел (например, 5 или 7) образует меньшие значения (2 или 3, соответственно) и устанавливает флаг CF в 1.
Кроме того, если необходимо выполнить сдвиг на 2 бита, то использование двух команд сдвига более эффективно, чем использование одной команды с загрузкой регистра CL значением 2.

1. Сдвиговые операции

SHL/SAL r	2	1 1 0 1 0 0 v w	mod * 0 0 r/m	Логический влево (Рг)
SHL/SAL mem	15+E			Логический влево (П)
SHR r	2	1 1 0 1 0 0 v w	mod 1 0 1 r/m	Логический вправо (Рг)
SHR mem	15+E			Логический вправо (П)
SAR r	2	1 1 0 1 0 0 v w	mod 1 1 1 r/m	Арифметический вправо (Рг)
SAR mem	15+E			Арифметический вправо (П)

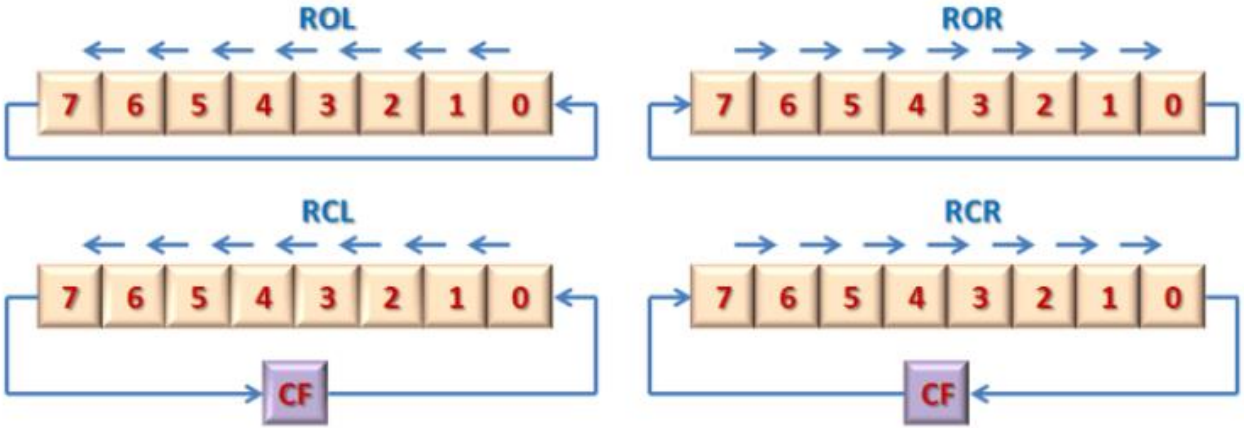
MUL src	71+E 124+E	1 1 1 1 0 1 1 w	mod 1 0 0 r/m	AX←AL источник (при w=0) DX, AX ← AX источник (при w=1)
IMUL src	90+E 144+E	1 1 1 1 0 1 1 w	mod 1 0 0 r/m	AX ← AL источник (при w=0) DX, AX ← AX источник (при w=1) со знаком

MOV r , data	4	data L 1 0 1 1 w reg data H(w=1)	data H(w=1) data L	Рг←Д

SAL AX, 1 ; 2
SAL AX, 1 ; 2

MOV CL, 2 ; 4
SAL AX, CL ; 2

1. Сдвиговые операции



Циклический сдвиг представляет собой операцию сдвига, при которой выдвинутый бит занимает освободившийся разряд.

Команды циклического сдвига:

ROR ————— Циклический сдвиг вправо
ROL ————— Циклический сдвиг влево
RCR ————— Циклический сдвиг вправо с переносом
RCL ————— Циклический сдвиг влево с переносом

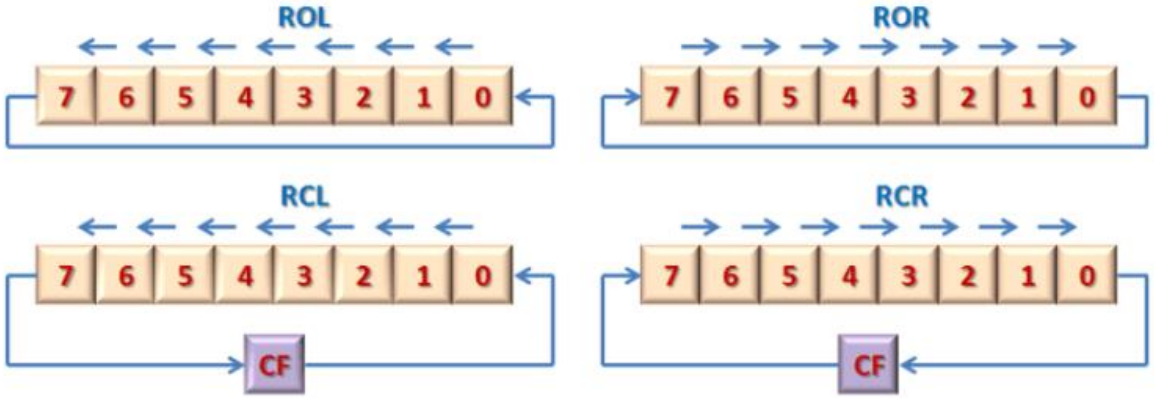
1. Сдвиговые операции

```
MOV CL, 03              BX:
MOV BX, 10110111B  ; 10110111
ROR BX, 1          ; 11011011          ; Сдвиг вправо на 1
ROR BX, CL         ; 01111011          ; Сдвиг вправо на 3
```

Первая команда **ROR** при выполнении циклического сдвига переносит правый единичный бит регистра **BX** в освободившуюся левую позицию. Вторая команда **ROR** переносит таким образом три правых бита.

```
MOV CL, 04              BX:
MOV BX, 10110111B  ; 10110111
ROR BX, CL         ; 01111011          ; Сдвиг вправо на 4
```

1. Сдвиговые операции

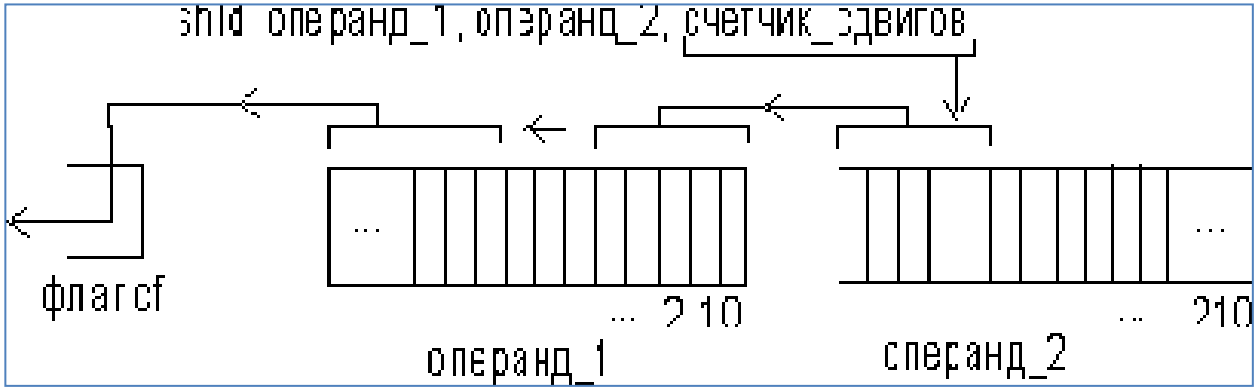


RCL / RCR – аналоги для **SHL / SHR**, но используют дополнительный (9й) бит, как ADC для ADD .

```
MOV CL, 3
MOV BX, 10110111B          ; 10110111
MOV AX, 00000000B          ; 00000000
SHL BX, 1                  ; 10110110   ; CF = 1
RCL  AX, 1                 ; 00000001
; RCL  AX, CL              ; 00000100
```

1. Сдвиговые операции

SHLD операнд_1, операнд_2, счетчик_сдвигов — сдвиг влево двойной точности. Команда shld производит замену путем сдвига битов операнда **операнд_1** влево, заполняя его биты справа значениями битов, вытесняемых из **операнд_2 с**огласно схеме. Количество сдвигаемых бит определяется значением **счетчик_сдвигов**, которое может лежать в диапазоне **0...31**. Это значение может задаваться непосредственным операндом или содержаться в регистре **cl.** Значение **операнд_2** не изменяется.



1. Сдвиговые операции

SHRD операнд_1,операнд_2,счетчик_сдвигов — сдвиг вправо двойной точности. Команда производит замену путем сдвига битов операнда **операнд_1** вправо, заполняя его биты слева значениями битов, вытесняемых из **операнд_2**. Количество сдвигаемых бит определяется значением **счетчик_сдвигов**, которое может лежать в диапазоне **0...31**. Это значение может задаваться непосредственным операндом или содержаться в регистре **cl**. Значение **операнд_2** не изменяется.

```
.data
pole_l  dd     0b21187f5h
pole_h  dd     45ff6711h
.code
;...
.386
      mov     cl,16   ;загрузка счетчика сдвига в cl
      mov    eax, pole_l
      shld    pole_h, eax, cl ; pole_h=6711b211h
      shl     pole_l, cl          ; pole_l=87f50000h,
```

1. Сдвиговые операции

				(П)
ROL r	8	1 1 0 1 0 0 v w	mod 0 0 0 r/m	Циклический влево (Рг)
ROL mem	20+*E*			Циклический влево (П)
ROR r	8	1 1 0 1 0 0 v w	mod 0 0 1 r/m	Циклический вправо (Рг)
ROR mem	20+*E*			Циклический вправо (П)
RCL r	8	1 1 0 1 0 0 v w	mod 0 1 0 r/m	Циклический через *CF* влево (Рг)
RCL mem	20+*E*			Циклический через *CF* влево (П)
RCR r	8	1 1 0 1 0 0 v w	mod 0 1 1 r/m	Циклический через *CF* вправо (Рг)
RCR mem	20+*E*			Циклический через *CF* вправо (П)

1. Сдвиговые операции

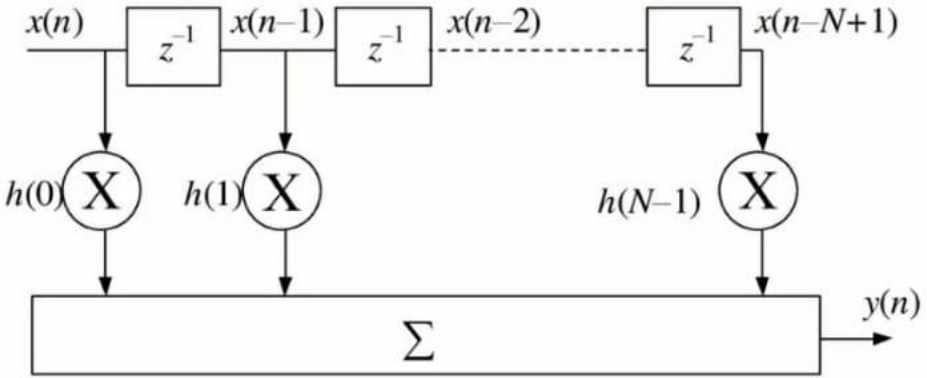
170 (0xAA) * 85 (0x55) = 14450 (0x3872)

```
        1010.1010
      *
        0101.0101
  ---------------
        1010.1010
      0.0000.000
+     10.1010.10
    1010.1010.
  1.0101.010
  ---------------
  11.1000.0111.0010 = 0x3872
```

1. Сдвиговые операции

Нерекурсивный фильтр (КИХ)

$$y(n) = \sum_{m=0}^{N-1} h(m)\,x(n-m)$$



Цифровой фильтр с конечной импульсной характеристикой

2. Двоично-десятичная арифметика

Двоично-десятичный код отличается от шестнадцатеричной системы тем, что он использует только первые десять комбинаций.

Значения BCD приведены в таблице.

В BCD используются только первые 10 комбинаций, которыми можно закодировать 10 цифр от 0 до 9.
Остальные комбинации в BCD являются запрещёнными.

Десятичное	HEX	BIN	BCD
0	0	0000	0000
1	1	0001	0001
2	2	0010	0010
3	3	0011	0011
4	4	0100	0100
5	5	0101	0101
6	6	0110	0110
7	7	0111	0111
8	8	1000	1000
9	9	1001	1001
10	A	1010	Запрещено
11	B	1011	Запрещено
12	C	1100	Запрещено
13	D	1101	Запрещено
14	E	1110	Запрещено
15	F	1111	Запрещено

Преимущества

1.Удобно использовать для вывода на индикаторы с одной цифрой. Например, в часах каждый индикатор отображает десятичное число в двоично-десятичной системе (от 0 до 9).

2.Упрощён вывод чисел на индикацию - вместо последовательного деления на 10 требуется просто вывести на индикацию каждый полубайт. По этой же причине проще ввод данных с цифровой клавиатуры.

3.Для **дробных чисел** (как с фиксированной, так и с плавающей запятой) при переводе в человекочитаемый десятичный формат и наоборот не теряется точность.

4.Упрощены умножение и деление на 10, а также округление.

По этим причинам двоично-десятичный формат применяется в калькуляторах и других устройствах, которые выводят данные на семисегментные или другие индикаторы, где каждый отдельный индикатор отображает только одну цифру.

Недостатки

1.Требует больше памяти.

2.Усложнены арифметические операции.

2. Двоично-десятичная арифметика

Так как в BCD данных используются только 10 возможных комбинаций 4-х битового поля вместо 16, существуют запрещённые комбинации битов. Поэтому, при сложении и вычитании чисел BCD действуют следующие правила:

1. При сложении двоично-десятичных чисел каждый раз, когда происходит перенос бита в старший полубайт, необходимо к полубайту, от которого произошёл перенос, добавить поправочное значение **0110** = 6_{10} = (16_{10} - 10_{10}) - разница количеств комбинаций полубайта и используемых значений, то есть всего комбинаций в тетраде 16, из них разрешённых 10, а запрещенных 6.

2. При сложении двоично-десятичных чисел каждый раз, когда встречается недопустимая для полубайта комбинация (число, большее 9), необходимо к каждой недопустимой комбинации добавить поправочное значение **0110** с разрешением переноса в старшие полубайты.

3. При вычитании двоично-десятичных чисел, для каждого полубайта, получившего заём из старшего полубайта, необходимо провести поправку, отняв значение **0110**.

2. Двоично-десятичная арифметика

Пример операции сложения двоично-десятичных чисел:

Требуется: Найти число X = Y + Z, где Y = 0929, Z = 1538

Решение: Представим числа Y и Z в двоично-десятичной форме:

Y = 929 (DEC) = 0000 1001 0010 1001 (BCD)
Z = 1538 (DEC) = 0001 0101 0011 1000 (BCD)

Ответ: 2467

2. Двоично-десятичная арифметика

Суммируем числа Y и Z по правилам двоичной арифметики:

```
          **          *
  0000 1001 0010 1001        (929)
+ 0001 0101 0011 1000        (1538)
 ____________________
= 0001 1110 0110 0001 - Двоичная сумма (7777)
+      0110      0110 - Поправка (по правилу 1 и правилу 2)
 ____________________
  0010 0100 0110 0111 - сумма BDC (2467)
```

Красным жирным в сумме выделены запрещённые комбинации.
Синим жирным выделены поправки по правилу 2.
Чёрным жирным выделены поправки по правилу 1.

* — тетрада, из которой был перенос в старшую тетраду
** — тетрада с запрещённой комбинацией битов (в сумме)

2. Двоично-десятичная арифметика

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

```
char dig0 = '0'; // == 0x30
char dig1 = '1'; // == 0x31
char dig9 = '9'; // == 0x39

dig9 == dig0 + 0x09 == '0' + 9
9 == dig9 – dig0 == dig9 – '0'
```

2. Двоично-десятичная арифметика

```
; A = 4 + 8
MOV BX, 4
MOV AX, 8
ADD AX, BX   ; AX = 12 = 0x0C

; A = '4' + '8'
MOV BX, '4'          ; 0x34
MOV AX, '8'          ; 0x38
ADD AX, BX           ; AX = 108 = 0x6C    //  не '12' = 0x3132
```

2. Двоично-десятичная арифметика

AAA (ASCII Adjust for Addition - коррекция для сложения ASCII-кода)
AAD (ASCII Adjust for Division - коррекция для деления ASCII-кода)
AAM (ASCII Adjust for Multiplication - коррекция для умножения ASCII-кода)
AAS (ASCII Adjust for Subtraction - коррекция для вычитания ASCII-кода)

Синтаксис: AAA
Операнды: Нет
Назначение: ASCII-коррекция после сложения
Флаги: Флаги CF и AF устанавливаются в 1, если произошел перенос из AL в AH, в противном случае они равны нулю. Значения флагов OF, SF, ZF и PF не определены.
Комментарий: Команда AAA корректирует сумму двух неупакованных двоично-десятичных чисел в регистре AL. Если коррекция приводит к десятичному переносу, регистр AH увеличивается на 1.
Эту команду лучше использовать сразу после команды сложения двух таких чисел.

Примеры:

```
mov     ax,0004h
add     ax,0008h        ;AX=000Ch
aaa                     ;AX=0102h
```

Примеры:

```
mov     ax,0034h
add     ax,0038h        ;AX=006Ch
aaa                     ;AX=0102h
add     ax, 0x3030      ; AX=3132h
```

2. Двоично-десятичная арифметика

Синтаксис: AAS
Операнды: Нет
Назначение: ASCII-коррекция после вычитания
Флаги: Флаги CF и AF устанавливаются в 1, если произошел заем из AL в AH, в противном случае они равны нулю. Значения флагов OF, SF, ZF и PF не определены.
Комментарий: Команда AAS корректирует разность двух неупакованных двоично-десятичных чисел в регистре AL сразу после команды SUB или SBB. Если операция приводит к займу, регистр AH уменьшается на 1.
Ограничения: Нет
Примеры:

```
mov ax, 0101h
sub  al,    06h        ;AX=01FBh
Aas                    ;AX=0005h
```

```
mov       ax, 0x38
sub       al,  0x34   ;AX=04h
aas                   ;AX=04h
```

```
mov       ax, 0x34
sub       al,  0x38   ;AX=00FCh
aas                   ;AX=FF06h = -4 (! dec)
```

2. Двоично-десятичная арифметика

Синтаксис: AAM

Операнды: Нет

Назначение: ASCII-коррекция после умножения

Флаги: Флаги SF, ZF и PF устанавливаются в соответствии с результатом. Значения флагов OF, AF и CF не определены.

Комментарий: Команда AAM корректирует результат умножения неупакованных двоично-десятичных чисел, который находится в регистре AX после выполнения команды MUL, преобразовывая его в пару неупакованных двоично-десятичных чисел в регистрах AH и AL.

Ограничения: Нет

Примеры:

```
mov     al,05h
mov     bl,05h
mul     bl          ;AX=0019h
aam                 ;AX=0205h
```

ASCII !!!

Синтаксис: AAD
Операнды: Нет
Назначение: ASCII-коррекция перед делением
Флаги: Флаги SF, ZF и PF устанавливаются в соответствии с результатом. Значения флагов OF, AF и CF не определены.
Комментарий: Команда AAD выполняет коррекцию неупакованного двоично-десятичного числа, находящегося в регистре AX, так, чтобы последующее деление привело к десятичному результату.
Ограничения: Нет
Примеры:

```
mov     ax,0205h
mov     bl,05h
aad                 ;AX=0019h = 25 dec
div     bl          ;AX=0005h
```

ASCII !!!

2. Двоично-десятичная арифметика

DAA (Decimal Adjustment for Addition - десятичная коррекция для сложения)
DAS (Decimal Adjustment for Subtraction - десятичн. коррекция для вычит.)

01020304h = 1234h

2. Двоично-десятичная арифметика

```
mov AL, 71H      ; AL = 0x71h
add AL, 44H       ; AL = 0x71h + 0x44h = 0xB5h
daa              ; AL = 0x15h
                 ; CF = 1 - перенос является частью результата 71 + 44 = 115
```

```
mov AL, 71H      ; AL = 0x71h
sub AL, 44H      ; AL = 0x71h - 0x44h = 0x2Dh
das              ; AL = 0x27h
                 ; CF = 0 - заем (перенос) является частью результата
```

2. Двоично-десятичная арифметика

				со знаком
DAA	4	00100111		*AL*←скорректированное *AL* (сложение, 2—10)
DAS	4	00101111		*AL*←скорректированное *AL* (вычитание, 2—10)
AAA	4	00110111		*AL*←скорректированное *AL* (сложение, ASC11)
AAS	4	00111111		*AL*←скорректированное *AL* (вычитание, ASC11)
AAM	83	11010100	00001010	*AX*←скорректированное *AX* (умножение)
AAD	60	11010101	00001010	*AX*←скорректированное *AX* (деление)

3. Логические операции

Логические операции являются важным элементом в проектировании микросхем и имеют много общего в логике программирования.
Команды **AND**, **OR**, **XOR** и **TEST** - являются командами логических операций.

Эти команды используются для сброса и установки отдельных бит.

Все эти команды обрабатывают один байт или одно слово в регистре или в памяти, и устанавливают флаги CF, OF, PF, SF, ZF.

AND: Если оба из сравниваемых битов равны 1, то результат равен 1; во всех остальных случаях результат - 0.

OR: Если хотя бы один из сравниваемых битов равен 1, то результат равен 1; если сравниваемые биты равны 0, то результат - 0.

XOR: Если один из сравниваемых битов равен 0, а другой равен 1, то результат равен 1; если сравниваемые биты одинаковы (оба - 0 или оба - 1) то результат - 0.

TEST: действует как **AND-устанавливает** флаги, но не изменяет биты.

3. Логические операции

Первый операнд в логических командах указывает на один байт или слово в регистре или в памяти и является единственным значением, которое может изменятся после выполнения команд.

Пример:	AND	OR	XOR
	0101	0101	0101
	0011	0011	0011
Результат:	0001	0111	0110

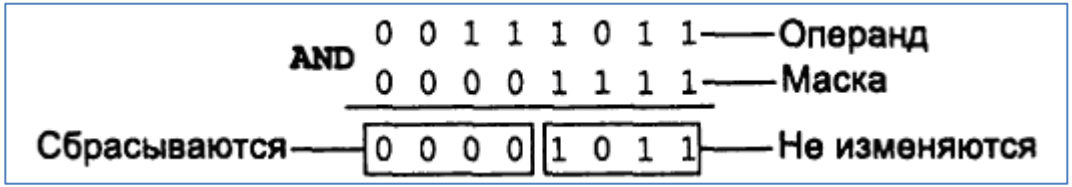
NOT – побитная инверсия

```
MOV ax, 0xAA55;
NOT ax          ; AX = 0x55AA
```

3. Логические операции

AND: Если оба из сравниваемых битов равны 1, то результат равен 1; во всех остальных случаях результат - 0.

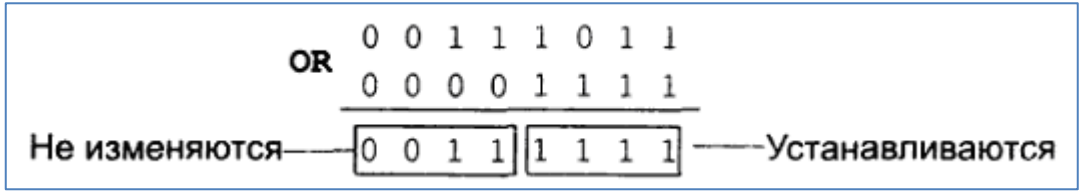
X	Y	X^Y
0	0	0
0	1	0
1	0	0
1	1	1




```
AND   AX, 111b   ; Остаток от деления на 2^3
```

OR: Если хотя бы один из сравниваемых битов равен 1, то результат равен 1; если сравниваемые биты равны 0, то результат - 0.

X	Y	X∨Y
0	0	0
0	1	1
1	0	1
1	1	1




```
mov dl, 5     ; Двоичное число

or  dl, 30h   ; Преобразуем в ASCII-код
```

3. Логические операции

С помощью команды OR можно определить, какое значение находится в регистре (**отрицательное, положительное или нуль**). Для этого вначале нужно выполнить команду OR, указав в качестве операндов один и тот же регистр, например:

OR AX, AX

Флаг нуля (ZF)	**Флаг знака (SF)**	**Значение числа**
0	0	Больше нуля
1	0	Равно нулю
0	1	Меньше нуля

XOR: Если один из сравниваемых битов равен 0, а другой равен 1, то результат равен 1; если сравниваемые биты одинаковы (оба - 0 или оба - 1) то результат - 0.

X	Y	$X \oplus Y$	$(X \oplus Y) \oplus Y$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Операция ИСКЛЮЧАЮЩЕГО ИЛИ обладает свойством реверсивности — если ее выполнить дважды с одним и тем же операндом, то значение результата инвертируется. Как показано в табл. 6, если два раза подряд выполнить операцию ИСКЛЮЧАЮЩЕГО ИЛИ между битами X и Y, то в результате получится исходное значение бита X.

BSF операнд_1,операнд_2 (Bit Scaning Forward) - сканирование битов вперед. Команда просматривает (сканирует) биты операнд_2 от младшего к старшему (от бита 0 до старшего бита) в поисках первого бита, установленного в 1. Если таковой обнаруживается, в операнд_1 заносится номер этого бита в виде целочисленного значения. Если все биты операнд_2 равны 0, то флаг нуля zf устанавливается в 1, в противном случае флаг zf сбрасывается в 0.

```
mov     al, 02h
bsf      bx,al   ;       bx=1
```

BSR операнд_1,операнд_2 (Bit Scaning Reset) — сканирование битов в обратном порядке.

Команда просматривает (сканирует) биты операнд_2 от старшего к младшему (от старшего бита к биту 0) в поисках первого бита, установленного в 1. Если таковой обнаруживается, в операнд_1 заносится номер этого бита в виде целочисленного значения.

При этом важно, что позиция первого единичного бита слева отсчитывается все равно относительно бита 0. Если все биты операнд_2 равны 0, то флаг нуля zf устанавливается в 1, в противном случае флаг zf сбрасывается в 0.

```
mov     al, 02h
bsr      bx, al   ;       bx=1
```

BT операнд, смещение_бита (Bit Test) — проверка бита.
Команда переносит значение бита в флаг cf.

```
bt      ax, 5    ;проверить значение бита 5
```

```
and  ax, 00100000b    ;проверить значение бита 5
```

BTS операнд, смещение_бита (Bit Test and Set) — проверка и установка бита.
Команда переносит значение бита в флаг cf и затем устанавливает проверяемый бит в 1.

```
mov ax, 10
bts     pole, ax  ;проверить и установить 10-й бит в pole
```

```
and  ax, 0x400    ;проверить значение бита 10
; …
or  ax, 0x400    ; установить значение бита 10
```

BTR операнд, смещение_бита (Bit Test and Reset) — проверка и сброс бита.
Команда переносит значение бита в флаг cf и затем устанавливает этот бит в 0.

```
mov ax, 10
btr     pole, ax  ;проверить и сбросить 10-й бит в pole
```

```
and  ax, 0x400    ;проверить значение бита 10
; …
mov bx, 0x400
not bx
and  ax, bx    ; сбросить значение бита 10
```

BTC операнд,смещение_бита (Bit Test and Convert) — проверка и инвертирование бита.
Команда переносит значение бита в флаг cf и затем инвертирует значение этого бита.

```
mov ax, 10
btc     pole, ax  ;проверить и инвертировать 10-й бит в pole
```

```
and  ax, 0x400    ;проверить значение бита 10
; …
mov bx, 0x400
not bx
and  ax, bx    ; сбросить значение бита 10
; …
or  ax, 0x400    ; установить значение бита 10
```

3. Логические операции

AND r1 , r2	3	0 0 1 0 0 0 d w	mod reg r/m	Pr←Pr∧Pr
AND r , mem	9+*E*			Pr←Pr∧П
AND mem , r	16+*E*			П←П∧Pr
AND r , data	4	1 0 0 0 0 0 0 w data L	mod 1 0 0 r/m data H(w=1)	Pr←Pr∧Д
AND mem , data	17+*E*			П←П∧Д
AND a , data	4	0 0 1 0 0 1 0 w data H(w=1)	data L	A←A∧Д
OR r1 , r2	3	0 0 0 0 1 0 d w	mod reg r/m	Pr←Pr∨Pr
OR r , mem	9+*E*			Pr←Pr∨П
OR mem , r	16+*E*			
OR r , data	4	1 0 0 0 0 0 0 w data L	mod 0 0 1 r/m data H(w=1)	Pr←Pr∨Д
OR mem , data	17+*E*			П←П∨Д
OR a , data	4	0 0 0 0 1 1 0 w data H	data L	A←A∨Д
XOR r1 , r2	3	0 0 1 1 0 0 d w	mod reg r/m	Pr←Pr+Pr
XOR r , mem	9+*E*			Pr←Pr+П
XOR mem , r	16+*E*			П←П+Пr
XOR r , data	4	1 0 0 0 0 0 0 w data L	mod 1 1 0 r/m data H(w=1)	Pr←Pr+Д

3. Логические операции

Команды	Флаги состояния					
	OF	CF	AF	SF	ZF	PF
MUL IMUL	+	+	?	?	?	?
DIV IDIV	?	?	?	?	?	?
DAA DAS	?	+	+	+	+	+
AAA AAS	?	+	+	?	?	?
AAM AAD	?	?	?	+	+	+
AND OR XOR TEST	0	0	?	+	+	+
SHL SHR	+	+	?	+	+	+
SHL SHR	?	+	?	+	+	+
SAR	0	+	?	+	+	+
ROL ROR RCL RCR	+	+	—	—	—	—
ROL ROR RCL RCR	?	+	—	—	—	—

2. Двоично-десятичная арифметика

```
section .data
        msg db '186'
        num dw 0
section .text
global CMAIN
CMAIN:
        xor eax, eax
        xor ebx, ebx
        mov bl, byte [msg + 2]
        sub bl, 0x30
        mov al, byte [msg + 1]
        sub al, 0x30
        mov cl, 10
        mul cl
        add bx, ax
        mov al, byte [msg]
        sub al, 0x30
        mov cl, 100
        mul cl
        add bx, ax
        mov [num], bx
```

3. Логические операции

IP адрес: 192.168.224.1 Маска: 255.255.255.0 (/24)

C0 A8 E0 01 FF FF FF 00 = 11111111.11111111.11111111.00000000

C0 A8 E0 01 **AND** FF FF FF 00 == C0 A8 E0 00

C0 A8 E0 02 **AND** FF FF FF 00 == C0 A8 E0 00

C0 A8 E0 03 **AND** FF FF FF 00 == C0 A8 E0 00

C0 A8 **E1** 03 **AND** FF FF FF 00 == C0 A8 E**1** 00 **!=** C0 A8 E0 00 (!!!)

3. Логические операции

```
IPv4 таблица маршрута
===========================================================================
Активные маршруты:
Сетевой адрес          Маска сети      Адрес шлюза       Интерфейс  Метрика
          0.0.0.0          0.0.0.0      192.168.1.1     192.168.1.2     25
        127.0.0.0        255.0.0.0         On-link       127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link       127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link       127.0.0.1    306
      192.168.1.0    255.255.255.0         On-link     192.168.1.2    281
      192.168.1.2  255.255.255.255         On-link     192.168.1.2    281
    192.168.1.255  255.255.255.255         On-link     192.168.1.2    281
     192.168.56.0    255.255.255.0         On-link    192.168.56.1    266
     192.168.56.1  255.255.255.255         On-link    192.168.56.1    266
   192.168.56.255  255.255.255.255         On-link    192.168.56.1    266
        224.0.0.0        240.0.0.0         On-link       127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link    192.168.56.1    266
        224.0.0.0        240.0.0.0         On-link     192.168.1.2    281
```

3. Логические операции

AX < -- > BX

```
PUSH AX
MOV AX, BX
POP BX
```

```
MOV var, AX
MOV AX, BX
MOV BX, var
```

```
XCHG AX, BX
```

```
XOR AX, BX
XOR BX, AX
XOR AX, BX
```

```
MOV CX, AX
MOV AX, BX
MOV BX, CX
```

MOV AX, 0

MOV r , data	4	1 0 1 1 w reg data H(w = 1)	data L	Pr←Д

SUB AX, AX

SUB r1 , r2 SUB r , mem	3 9+E	0 0 1 0 1 0 d w	mod reg r/m	Pr←Pr—Pr Pr←Pr—П

XOR AX, AX

XOR r1 , r2	3	0 0 1 1 0 0 d w	mod reg r/m	Pr←Pr+Pr

Спасибо