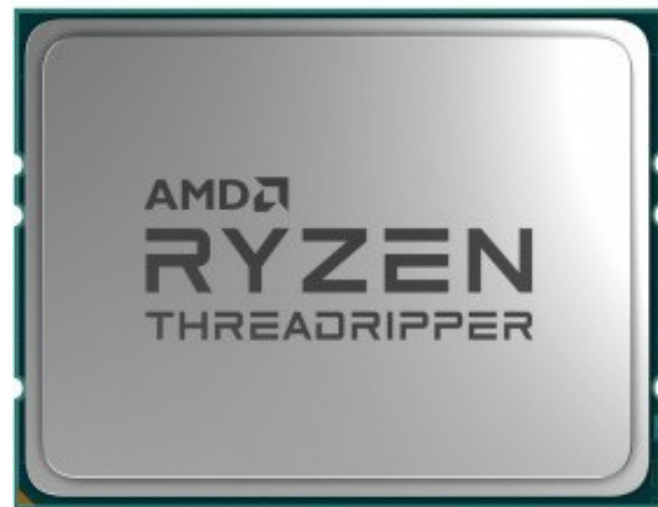
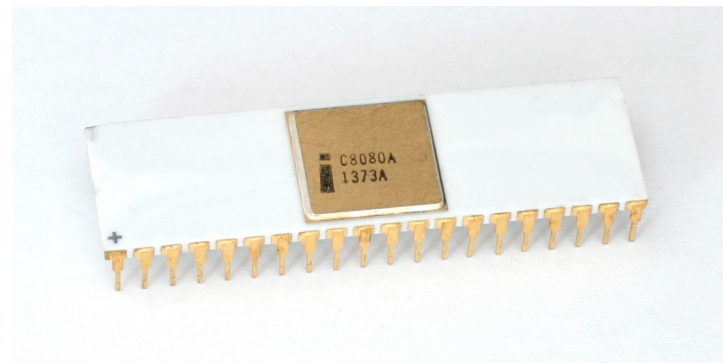


Архитектура процессоров Intel



Архитектура семейства Intel X86. Предтеча

Intel 8080 — 8-битный микропроцессор, выпущенный компанией Intel в апреле 1974 года. Представляет собой усовершенствованную версию процессора Intel 8008. По заверениям Intel, этот процессор обеспечивал десятикратный прирост производительности по сравнению с микропроцессором Intel 8008.

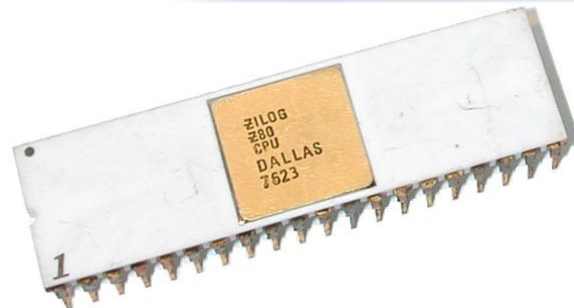


Функциональный аналог микропроцессора Intel i8080A Также существовал более ранний вариант микропроцессора K580ИК80, выпускавшийся с 1977 по середину 1990-х

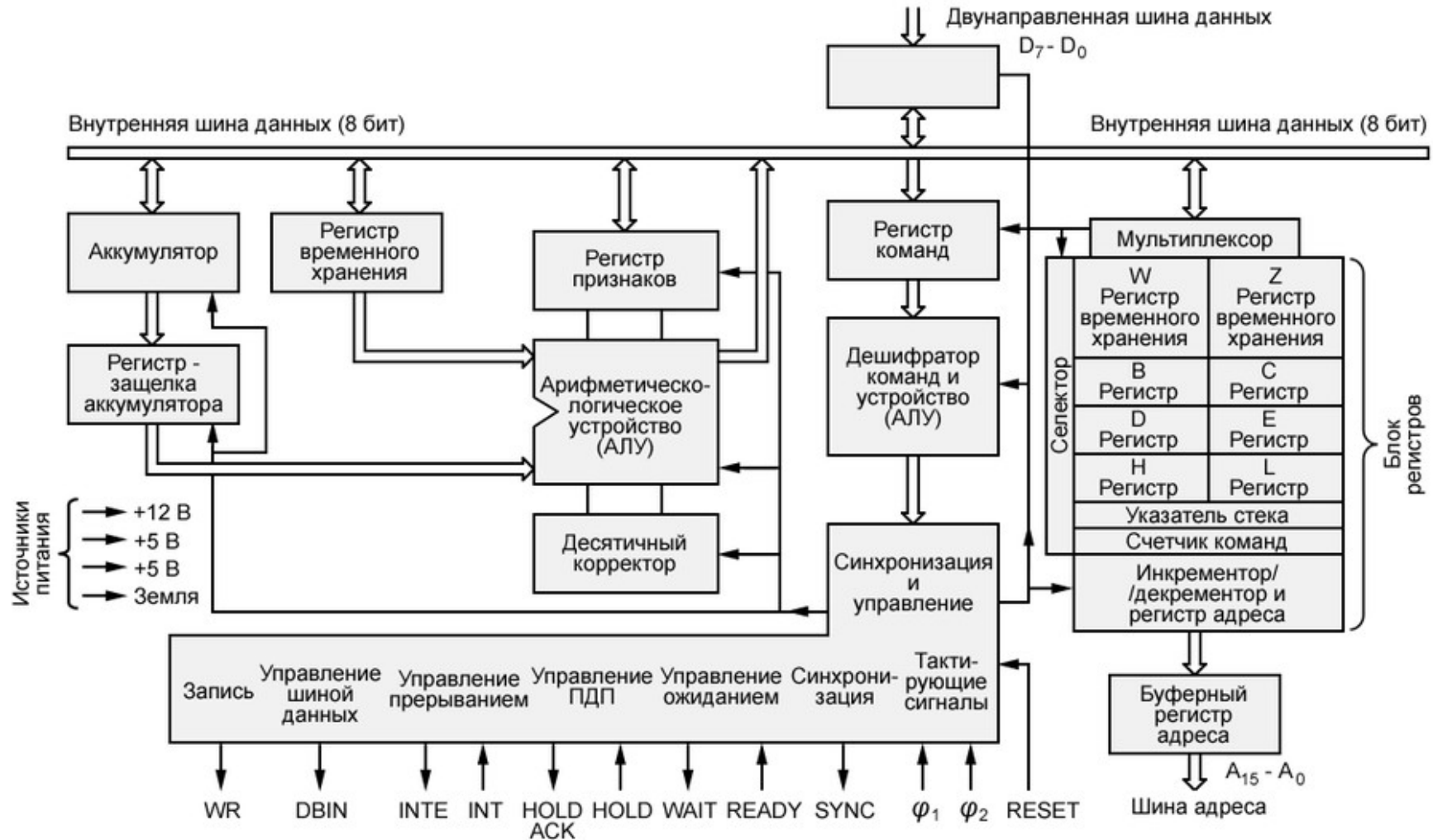


Копии i8080 производили Mitsubishi, National Semiconductor, NEC, Siemens, AMD и некоторые другие.

Zilog Z80: существенно большая производительность, расширенный набор команд, питание только 5 вольт, значительно меньше микросхем поддержки.



Процессор Intel 8080. Структура



Процессор Intel 8080. Регистры

Семь 8-разрядных доступных регистров:

- **A** — аккумулятор — предназначен для обмена с внешними устройствами. Служит при этом источником операнда и приемником результата.
- **B, C, D, E, H, L** — блок регистров общего назначения (РОН). Могут хранить как данные так и адреса. Используются как 8-разрядные регистры и как 16-разрядные регистровые пары.
- **W, Z** — используются как буферные и программно недоступны. Для временного хранения второго и третьего байтов команд перехода.
- **T** — также программно не доступен и служит для временного хранения второго операнда арифметических и логических операций.
- **SP** — указатель стека, содержит адрес памяти, начиная с которой ее можно использовать как стек.
- **PC** — счетчик команд. Содержит адрес очередной исполняемой команды.

В качестве адреса можно использовать содержимое любой регистровой пары. При выдаче адреса содержимое соответствующей регистровой пары переходит в регистр адреса, в буфер и на шину адреса.

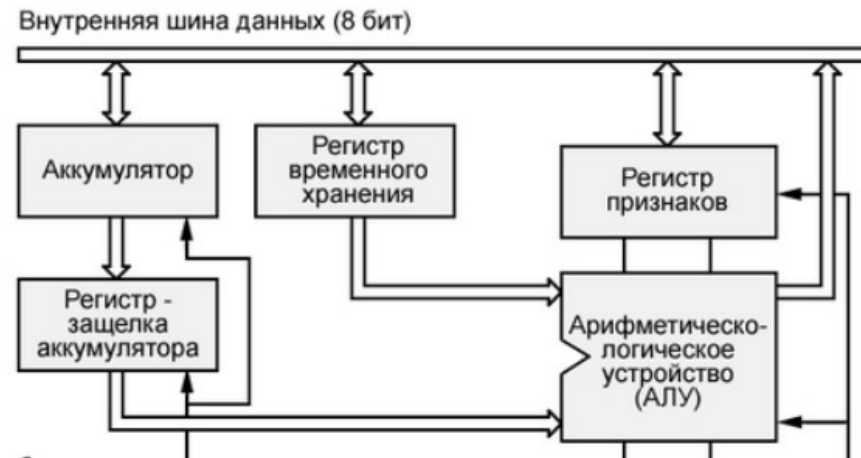
Процессор Intel 8080. Арифметико-логическое устройство (АЛУ)

АЛУ — 8-разрядная комбинационно-логическая схема, реализующая четыре арифметические операции, четыре логические, и четыре циклических сдвига.

При выполнении операций одним из операндов служит содержимое аккумулятора.

Сдвиг также выполняется над аккумулятором.

Предусмотрена возможность выполнения арифметических операций над 10-ичными числами при помощи дополнительного блока десятичной коррекции. Она осуществляется при помощи специальной команды DAA.



Процессор Intel 8080. Регистр флагов (признаков)

Регистр флагов (F) используется для хранения признаков.

7	6	5	4	3	2	1	0
S	Z		AC		P		CY

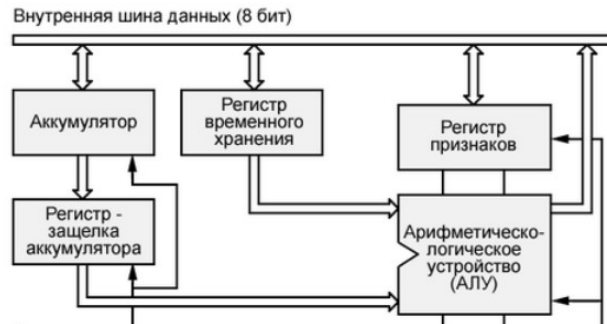
CY — бит переноса/заема. Если перенос был, то устанавливается в 1.

P — признак четности количества единиц результата. Если четное, то 1.

AC — признак переноса из 3 разряда в 4. Если такой был, то устанавливается в 1.

Z — признак нулевого результата. 0, если результат ненулевой, и 1 если результат = 0.

S — признак отрицательного результата. 1 при отрицательном и 0 при положительном.



Процессор Intel 8080. Форматы команд

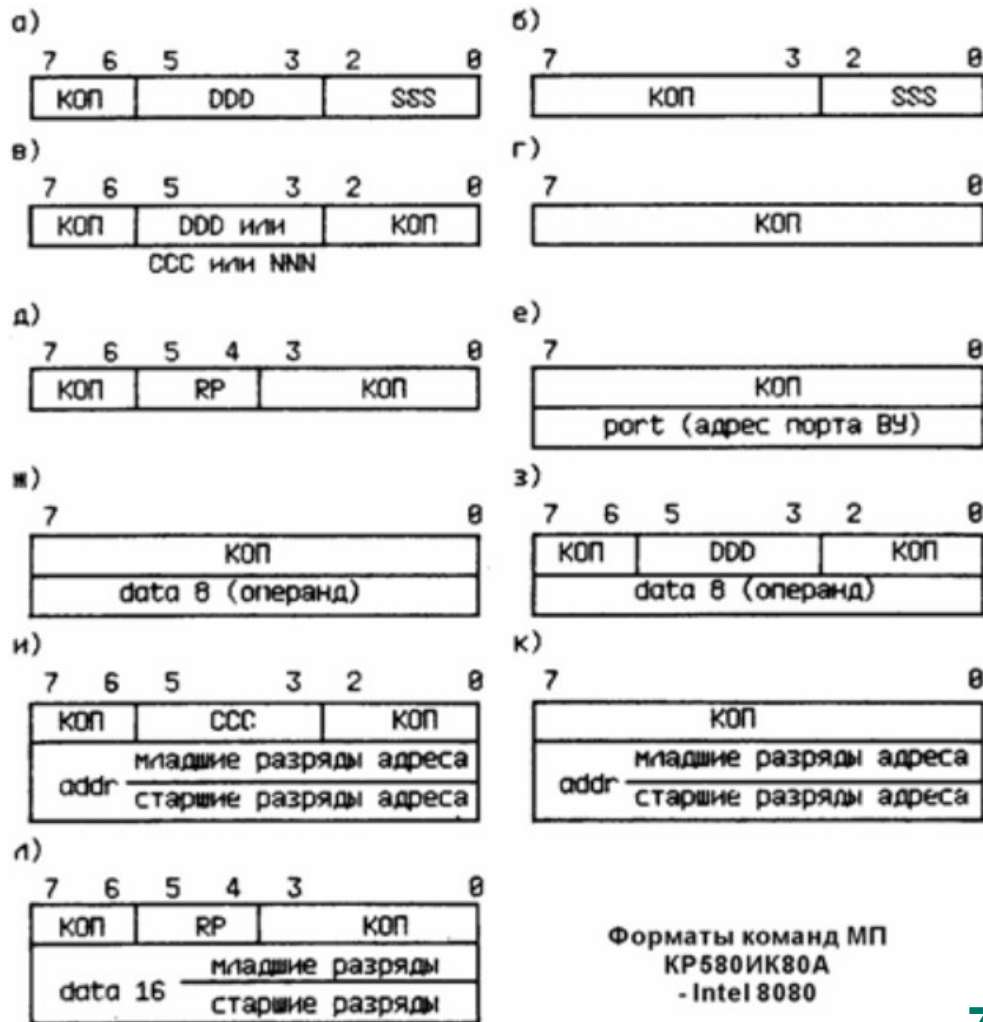
11 форматов команд, коды операций (КОП) которых имеют различную длину (2, 5, 6 или 8 бит) и часто состоят из двух частей. В зависимости от способа адресации команды могут быть одно-, двух- или трехбайтовыми.

Используются четыре способа адресации:

- регистровая (MOV r1, r2; ADD r; PCHL; ...);
- косвенно-регистровая (MOV M, r; ADD M; PUSH; POP; ...);
- непосредственная (MVI r, data 8; ADI data 8; ...);
- прямая (LDA addr; SHLD addr; IN port; ...).

Трехбитовые поля адресации источника и приемника: **SSS** и **DDD** соответственно. В мнемонических изображениях двухадресных команд **приемник** указывается на **первом месте**, а **источник** — на **втором**.

Для обозначения содержимого регистра или ячейки памяти используется запись в скобках: (r1), (r), (H), (M) и т. п.



Форматы команд МП
КР580ИК80А
- Intel 8080

Архитектура семейства Intel X86

История развития

1978 г. Intel 8086

Размер машинного слова: 2 байта (16 разрядов)

Расположение байт: “little endian”.

Реальный режим работы (*real mode*): адресация до $2^{20}=1$ МБайт ОЗУ

Число основных команд: 133 основные

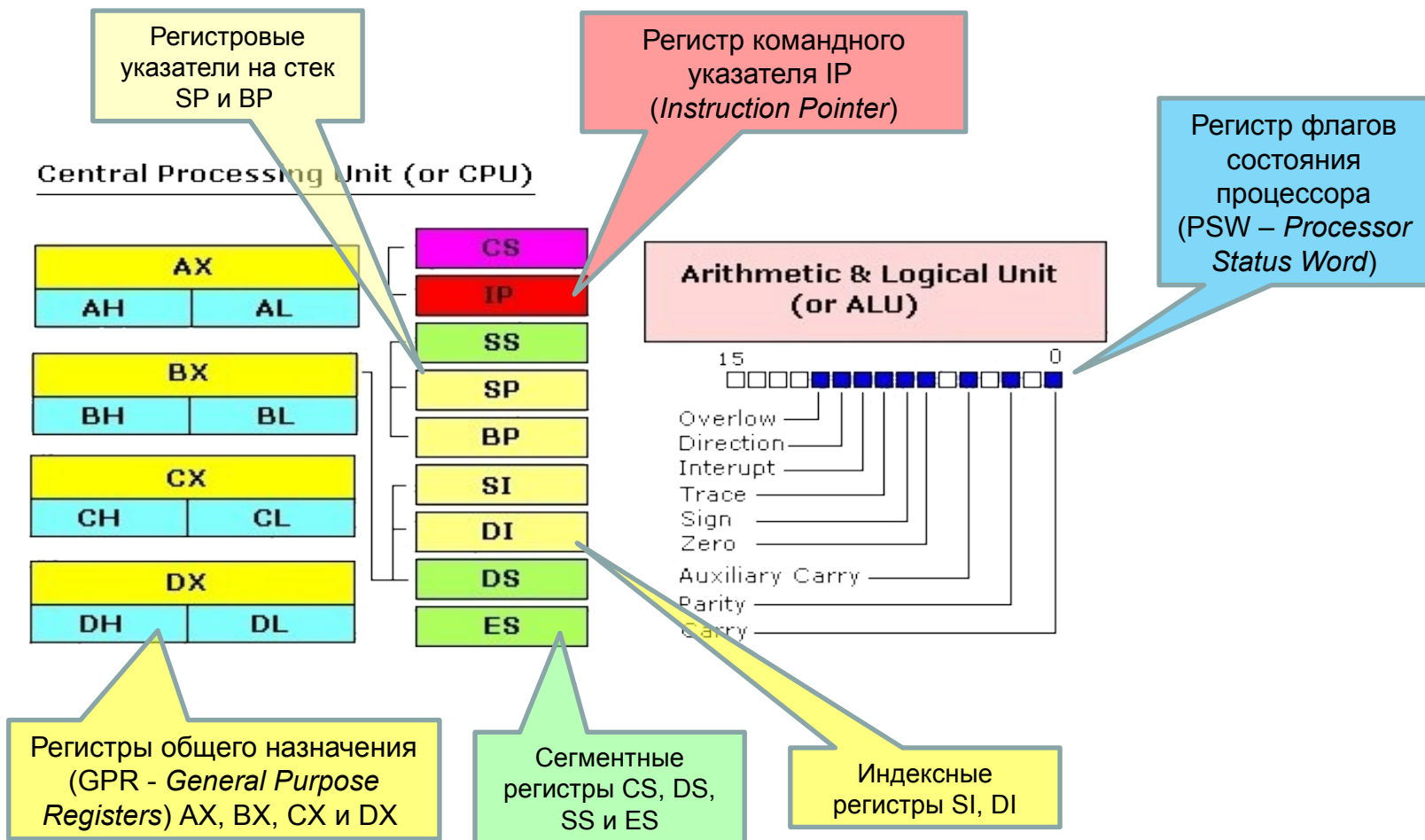
Длина команды: 1-6 байт

1982 г. Intel 80286

Размер машинного слова: 2 байта (16 разрядов)

Защищённый режим работы (*protected mode*): защита памяти, контексты задач, средства для организации виртуальной памяти. Адресуемая память – 16 Мбайт.

Процессор Intel 8086. Внутренние регистры



Процессор Intel 8086. Расширение адресного пространства

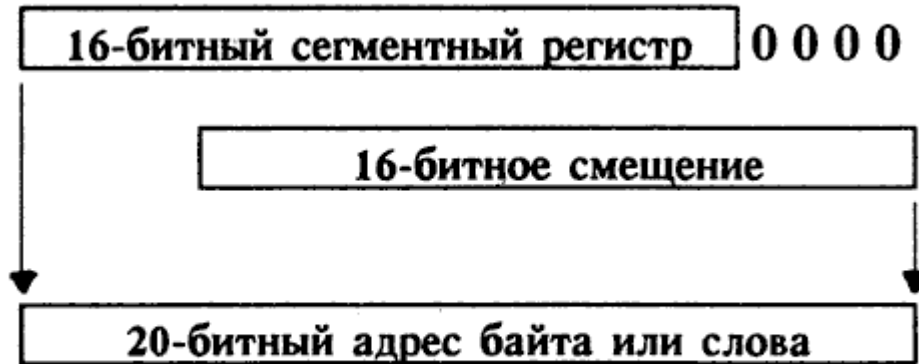
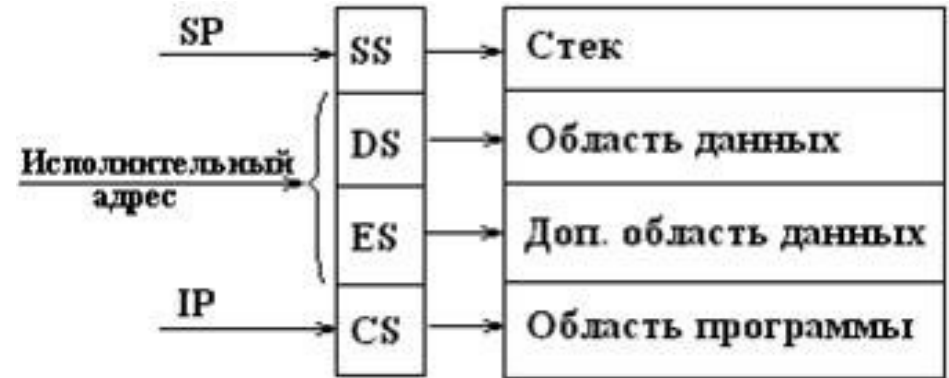


Рис.4 Формирование адресов байта или слова



Модели памяти Intel 8086

Использовались 6 моделей памяти

Tiny (крохотная). Минимальный размер (64К). CS, DS, SS и ES устанавливаются на один и тот же адрес. Обычно для системных программ.

Small (малая). Для небольших прикладных программ. CS, DS расположены отдельно друг от друга и не перекрываются. (64К кода программы + 64К данных и стека).

Medium (средняя). Для больших программ, с небольшим объемом данных. Для кода, используются указатели far. Данные + стек ограничены размером 64К. Код может занимать до 1М.

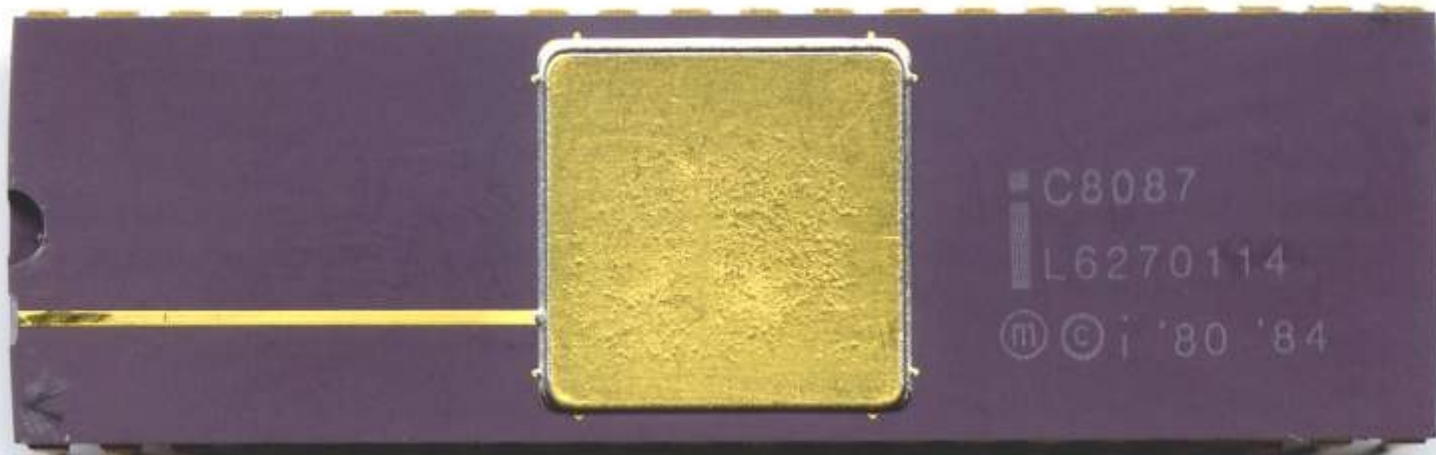
Compact (компактная). Размер кода невелик, адресация большого объема данных. Указатели far используются для данных, но не для кода. Код ограничен 64К. Предельный размер данных - 1 Мб.

Large (большая). Модели large и huge только в очень больших программах. Дальние указатели используются как для кода, так и для данных, что дает предельный размер 1 Мб для обоих.

Huge (огромная). Дальние указатели используются как для кода, так и для данных. Обычно размер статических данных ограничивается 64К. Модель памяти huge отменяет это ограничение, позволяя статическим данным занимать более 64К.



Математический сопроцессор Intel 8087





<https://xakep.ru/issues/xb/002/>

Введение в Assembler

Изучаем низкоуровневое программирование с нуля

Знать ассемблер раньше было обязательно для каждого хакера. Сейчас — только для лучших в своей профессии. Понимать язык машины не только полезно, но и крайне увлекательно: освоив ассемблер, ты научишься программировать без помощи операционной системы и общаться с «железом» напрямую.

Купить этот выпуск:

590 руб

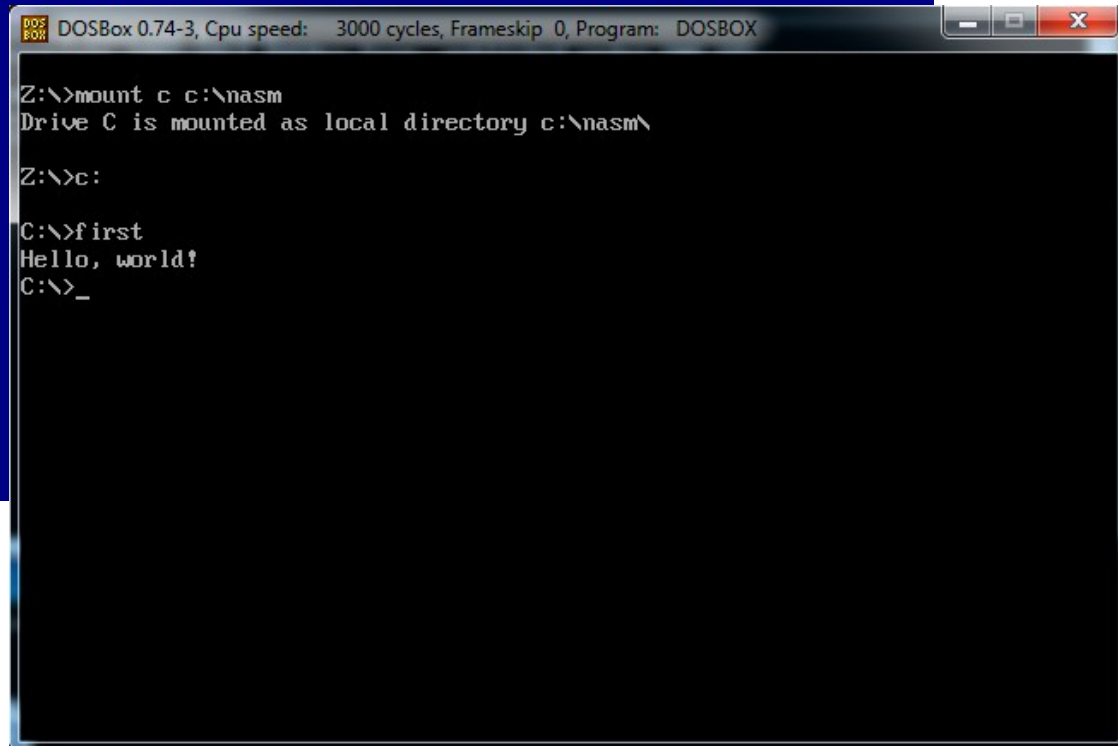
один раз и навсегда

подборка статей

Hello World на Intel 8086

```
org      0x100
@@start:
mov      bx, string
@@repeat:
mov      al, [bx]
cmp      al, 0
je       @@end
push     bx
mov      ah, 0x0E
mov      bx, 0x000F
int      0x10
pop      bx
inc      bx
jmp      @@repeat
@@end:
int      0x20

string:
db       "Hello, world", '?', 0
```



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Z:\>mount c c:\nasm
Drive C is mounted as local directory c:\nasm\
Z:\>c:
C:\>first
Hello, world!
C:\>_
```



```

; =====
; Упомянутая программа "Hello, world!"
; =====

    org 0x100                ; Резервируем первые 100 байт сегмента кода под
                             ; системные нужды (бюрократия для com-файла)

@@start:
    mov     bx, string       ; Помещаем в регистр BX адрес переменной string
@@repeat:
    mov     al, [bx]         ; Помещаем в AL очередной символ из string
    cmp     al, 0            ; AL = 0 ?
    je      @@end            ; Да – строка кончилась. Выходим (jump if zero)
    push    bx               ; Сохраняем на стеке указатель на текущий символ

;----- Выводим символ на экран, пользуясь прерыванием BIOS: 0x10
    mov     ah, 0x0E         ; 0E – функция которая выводит символ на экран
    mov     bx, 0x000F       ; BH – нулевая страница; BL – цвет символа
    int     0x10

;-----

    pop     bx               ; Восстанавливаем указатель на текущий символ
    inc     bx               ; и прокручиваем его к следующему символу
    jmp     @@repeat         ; Идём обрабатывать этот новый символ

@@end:
    int     0x20             ; Выходим в командную строку

string:
    db      "Hello, world", '?', 0

```


Архитектура семейства Intel X86

История развития

1982 г. Intel 80386

Размер машинного слова: 4 байта (32 разряда)

Адресуемая память $2^{32} \approx 4$ Гбайт.

Защищенный и реальный режимы работы. Виртуальный 8086.

Внешний сопроцессор с плавающей точкой.

1989 г. Intel 80486

Появление встроенного сопроцессора с плавающей точкой.



Советует свою методичку,
которую написал 50 лет
назад

Твой билет на долг осеино

Попадает по ярлыку с пятого
раза, не может найти кнопку
пуск

Оставляет недружелюбные
следы слюны на
программе, которую ты
сдал на листочке

Преподает, потому что
никто кроме студента
его марам не станет
слушать

Облизывает волосы,
растущие из носа

Стойкий
запах могилы

Забил, что воротник
рубашки когда-то не
был желтым

Горой стоит за
ВУЗ, ведь больше
его нигде не
возьмут

Вспоминает как
было круто
писать код на
перфокартах

Облизывает волосы,
растущие из носа

Архитектура x86-64

x86-64 (также **AMD64/Intel64/EM64T**) — 64-битная версия (изначально — расширение) архитектуры x86, разработанная компанией AMD и представленная в 2000 году, позволяющая выполнять программы в 64-разрядном режиме.

Это расширение архитектуры x86, а ныне — версия архитектуры x86, почти полностью обратно совместимая с 32-разрядной версией архитектуры x86, известной ныне как IA-32.

Особенности:

- **16** целочисленных 64-битных регистров общего назначения (**RAX**, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8 — R15);
- **8** 80-битных регистров с плавающей точкой (ST0 — ST7);
- **8** 64-битных регистров Multimedia Extensions (MM0 — MM7, имеют общее пространство с регистрами ST0 — ST7);
- **16** 128-битных регистров SSE (XMM0 — XMM15);
- **64**-битный указатель RIP и 64-битный регистр флагов RFLAGS.

X86-64. Режимы работы

- 1) ***Long mode*** («длинный» режим);
- 2) ***Legacy mode*** («наследственный» режим).

«Long Mode»

Основной для AMD64. Позволяет воспользоваться всеми дополнительными преимуществами архитектуры. Для работы требуется 64-разрядная ОС.

Выполняются 64-разрядные программы.

Доступна обратная совместимость: поддержка 32-разрядных приложений.

«Legacy Mode»

Дает возможность AMD64 работать с инструкциями для x86-процессоров.

Полная совместимость с 32-разрядным кодом и соответствующими ОС.

Дополнительные функции, доступные под AMD64, становятся неактивны.

64-битные программы и соответствующие операционные системы функционировать не будут.

Таблица регистров x86-64

ZMM0	YMM0	XMM0	ZMM1	YMM1	XMM1	ST(0)	MM0	ST(1)	MM1	AL AH AX EAX RAX	R8B R8W R8D R8	R12B R12W R12D R12	MSW CR0	CR4					
ZMM2	YMM2	XMM2	ZMM3	YMM3	XMM3	ST(2)	MM2	ST(3)	MM3	BL BH BX EBX RBX	R9B R9W R9D R9	R13B R13W R13D R13	CR1	CR5					
ZMM4	YMM4	XMM4	ZMM5	YMM5	XMM5	ST(4)	MM4	ST(5)	MM5	CL CH CX ECX RCX	R10B R10W R10D R10	R14B R14W R14D R14	CR2	CR6					
ZMM6	YMM6	XMM6	ZMM7	YMM7	XMM7	ST(6)	MM6	ST(7)	MM7	DL DH DX EDX RDX	R11B R11W R11D R11	R15B R15W R15D R15	CR3	CR7					
ZMM8	YMM8	XMM8	ZMM9	YMM9	XMM9					BPL BP EBP RBP	DI EI EDI RDI	IP EIP RIP	MXCSR	CR8					
ZMM10	YMM10	XMM10	ZMM11	YMM11	XMM11	CW	FP_IP	FP_DP	FP_CS	SIL SI ESI RSI	SPL SP ESP RSP			CR9					
ZMM12	YMM12	XMM12	ZMM13	YMM13	XMM13	SW								CR10					
ZMM14	YMM14	XMM14	ZMM15	YMM15	XMM15	TW								CR11					
ZMM16	ZMM17	ZMM18	ZMM19	ZMM20	ZMM21	ZMM22	ZMM23	FP_DS						CR12					
ZMM24	ZMM25	ZMM26	ZMM27	ZMM28	ZMM29	ZMM30	ZMM31	FP_OPC	FP_DP	FP_IP	CS	SS	DS	GDTR	IDTR	DR0	DR6	CR13	
											ES	FS	GS	TR	LDTR	DR1	DR7	CR14	
																DR2	DR8	CR15	
																DR3	DR9		
																DR4	DR10	DR12	DR14
																DR5	DR11	DR13	DR15

■ 8-bit register ■ 32-bit register ■ 80-bit register ■ 256-bit register
■ 16-bit register ■ 64-bit register ■ 128-bit register ■ 512-bit register

Архитектура x86-64. Регистры

64-битовый	32-битовый	16-битовый	Нижний 8-битовый	Верхний 8-битовый	Комментарий
rax	eax	ax	al	ah	
rbx	ebx	bx	bl	bh	
rcx	ecx	cx	cl	ch	
rdx	edx	dx	dl	dh	
rsi	esi	si	sil	-	
rdi	edi	di	dil	-	
rbp	ebp	bp	bpl	-	Указатель базы (адреса)
rsp	esp	sp	spl	-	Указатель стека
r8	r8d	r8w	r8b	-	
r9	r9d	r9w	r9b	-	
r10	r10d	r10w	r10b	-	
r11	r11d	r11w	r11b	-	
r12	r12d	r12w	r12b	-	
r13	r13d	r13w	r13b	-	
r14	r14d	r14w	r14b	-	
r15	r15d	r15w	r15b	-	

Архитектура x86-64. Регистры

IP (Instruction Pointer) — регистр, указывающий на смещение (адрес) инструкций в сегменте кода (1234:0100h сегмент/смещение).

IP — 16-битный (младшая часть EIP)

EIP — 32-битный аналог (младшая часть RIP)

RIP — 64-битный аналог

Сегментные регистры — регистры, указывающие на сегменты.

Все сегментные регистры — 16-разрядные

CS (Code Segment), **DS** (Data Segment), **SS** (Stack Segment), **ES** (Extra Segment), **FS**, **GS**

В реальном режиме работы процессора сегментные регистры содержат адрес начала 64Kb сегмента, смещённый вправо на 4 бита.

В защищённом режиме работы процессора сегментные регистры содержат селектор сегмента памяти, выделенного ОС.

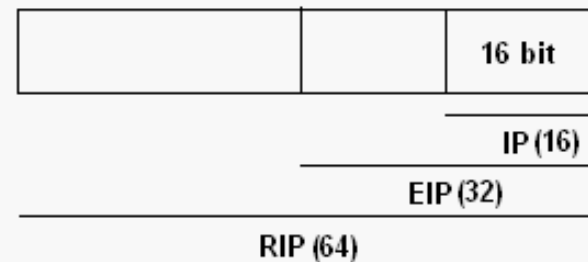
CS — указатель на кодовый сегмент.

Связка **CS:IP** (**CS:EIP/CS:RIP** — в защищённом/64-битном режиме) указывает на адрес в памяти следующей команды.

В 64-разрядном режиме сегментные регистры **CS**, **DS**, **ES** и **SS** в формировании линейного (непрерывного) адреса не участвуют, поскольку сегментация в этом режиме не поддерживается.



Регистр - указатель инструкций



Архитектура x86-64. Регистры

IP (Instruction Pointer) — регистр, указывающий на смещение (адрес) инструкций в сегменте кода (1234:0100h сегмент/смещение).

IP — 16-битный (младшая часть *EIP*)

EIP — 32-битный аналог (младшая часть *RIP*)

RIP — 64-битный аналог

Сегментные регистры — регистры, указывающие на сегменты.

Все сегментные регистры — 16-разрядные

CS (Code Segment), **DS** (Data Segment), **SS** (Stack Segment), **ES** (Extra Segment), **FS**, **GS**

В реальном режиме работы процессора сегментные регистры содержат адрес начала 64Kb сегмента, смещённый вправо на 4 бита.

В защищённом режиме работы процессора сегментные регистры содержат селектор сегмента памяти, выделенного ОС.

CS — указатель на кодовый сегмент.

Связка **CS:IP** (**CS:EIP/CS:RIP** — в защищённом/64-битном режиме) указывает на адрес в памяти следующей команды.

В 64-разрядном режиме сегментные регистры **CS**, **DS**, **ES** и **SS** в формировании линейного (непрерывного) адреса не участвуют, поскольку сегментация в этом режиме не поддерживается.



Архитектура x86. Регистры

RAX, RCX, RDX, RBX, RSP, RBP, RSI, RDI, R8-R15 — 64-битные (**registry AX**)

EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, R8D-R15D — 32-битные (**extended AX**)

AX (**Accumulator**),
CX (**Count Register**),
DX (**Data Register**),
BX (**Base Register**),
SP (**Stack Pointer**),
BP (**Base Pointer**),
SI (**Source Index**),
DI (**Destination Index**),
R8W-R15W — 16-битные

AH, AL, CH, CL, DH, DL, BH, BL, SPL, BPL, SIL, DIL, R8B-R15B — 8-битные (половинки 16-битных регистров)

Пример:

AH — high AX — старшая половина 8 бит

AL — low AX — младшая половина 8 бит

Архитектура x86. Регистры

Регистр флагов FLAGS (16 бит) / **EFLAGS** (32 бита) / **RFLAGS** (64 бита) — содержит текущее состояние процессора.

Системные регистры GDTR, LDTR и IDTR введены в процессорах начиная с Intel286 и предназначены для хранения базовых адресов таблиц дескрипторов — важнейших составляющих системной архитектуры при работе в защищённом режиме.

Регистр GDTR содержит 32-битный (24-битный для Intel286) базовый адрес и 16-битный предел глобальной таблицы дескрипторов (**GDT**).

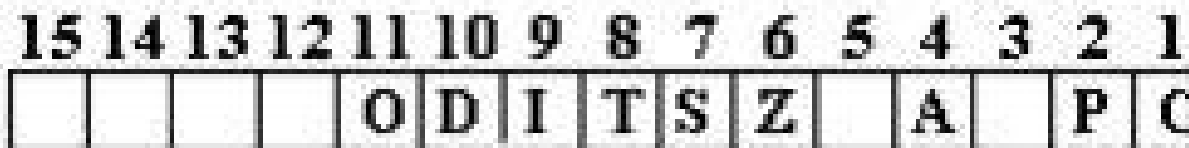
Видимая часть регистра **LDTR** содержит только селектор дескриптора локальной таблицы дескрипторов (**LDT**). Сам дескриптор **LDT** автоматически загружается в скрытую часть **LDTR** из глобальной таблицы дескрипторов.

Регистр **IDTR** содержит 32-битный (24-битный для Intel286) базовый адрес и 16-битный предел таблицы дескрипторов прерываний (**IDT**). В реальном режиме может быть использован для изменения местоположения таблицы векторов прерываний.

Видимая часть регистра **TR** содержит селектор дескриптора сегмента состояния задачи (**TSS**). Сам дескриптор **TSS** автоматически загружается в скрытую часть **TR** из глобальной таблицы дескрипторов.

Архитектура x86-64. Регистр флагов

Имя	Символьное обозначение	Бит	Описание
Carry	CF	0	В предыдущей инструкции был выполнен перенос (разрядов)
Parity	PF	2	Последний байт содержит четное число единиц
Adjust	AF	4	Операции BCD (в двоично-десятичном коде)
Zero	ZF	6	Результат предыдущей инструкции равен нулю
Sign	SF	8	В результате выполнения предыдущей инструкции самый значимый бит равен 1
Direction	DF	10	Направление операций со строками (инкремент или декремент)
Overflow	OF	11	В результате выполнения предыдущей инструкции возникло переполнение



Архитектура X86. Формат команд

Структура машинной команды для режима "16 разрядов"

Число байт	Компонент команды
0 или 1	Префикс команды
0 или 1	Префикс замены сегмента
1 или 2	Код операции
0 или 1	Байт MRM - (mod,reg,r/m)
0,1 или 2	Поле для задания адреса
0,1 или 2	Непосредственный операнд

Примечание. Есть две машинные команды (CALL, код 9A, JMP, код EA), в которых поле для задания адреса занимает 4 байта.

Архитектура X86. Формат команд

Префиксы перед кодом операции

- Если в команде более одного префикса, то они должны располагаться в порядке, указанных в форматах.
- В команде не могут стоять несколько префиксов одной группы.
- Префикс действует только в пределах той команды, перед которой он стоит.
- Код префикса не может совпадать с кодом операции какой-либо команды.

Префиксы команды

Имеют свои собственные имена на языке ассемблера. Некоторые ассемблеры показывают эти команды в отдельной строке.

- **F0** - префикс блокировки шины, команда LOCK ("lock" - запирать). Употребляется только с командами, которые поддерживают возможность блокировки шины.
- **F2, F3** - префиксы повторения. Команда REP ("repeat" - повторять) и другие команды этой группы. Употребляются только с цепочечными командами. Префиксы группы REP позволяют организовать циклическое выполнение цепочечной команды.
- Код **F1** – не используется (пока?).

Архитектура X86. Формат команд

Структура машинной команды для режима "32 разряда"

Число байт	Компонент команды
0 или 1	Префикс команды
0 или 1	Префикс изменения размера адреса
0 или 1	Префикс изменения размера операнда
0 или 1	Префикс замены сегмента
1 или 2	Код операции
0 или 1	Байт MRM - (mod,reg,r/m)
0 или 1	Байт SIB - (scale,index,base)
0,1,2 или 4	Поле для задания адреса
0,1,2 или 4	Непосредственный операнд

Примечание. Есть две машинные команды (CALL, код 9A, JMP, код EA), в которых поле для задания адреса занимает 6 байт.

Архитектура X86. Формат команд

Префиксы перед кодом операции

Префикс изменения размера адреса

Код **67**.

Действие префикса зависит от конкретной команды.

- *Если режим выполнения программы "32-разрядный", то для текущей команды устанавливается атрибут размера адреса "16" разрядов.*
- *Если общий режим выполнения программы "16-разрядный", то для команды, перед которой есть префикс 67, устанавливается атрибут размера адреса "32" бита.*

Префикс изменения размера операнда

Код **66**.

Действие префикса зависит от конкретной команды.

- *Если режим выполнения программы "32-разрядный", то для текущей команды устанавливается атрибут размера операнда "16" разрядов.*
- *Если общий режим выполнения программы "16-разрядный", то для команды, перед которой есть префикс 67, устанавливается атрибут размера операнда "32" бита.*

Архитектура X86. Формат команд

Префиксы перед кодом операции

Префиксы замены сегмента

Адрес памяти всегда относится к некоторому сегменту, заданному по умолчанию. С помощью префикса можно изменить сегмент. Замена зависит от конкретной команды.

- Код **26** - сегмент по умолчанию заменяется на сегмент *ES*.
- Код **2E** - сегмент по умолчанию заменяется на сегмент *CS*.
- Код **36** - сегмент по умолчанию заменяется на сегмент *SS*.
- Код **3E** - сегмент по умолчанию заменяется на сегмент *DS*.
- Код **64** - сегмент по умолчанию заменяется на сегмент *FS*.
- Код **65** - сегмент по умолчанию заменяется на сегмент *GS*.

Архитектура X86. Формат команд

Код операции (КОП)

- Всегда присутствует в любой машинной команде.
- Может состоять из одного байта или из двух байт.
- Если первый байт кода операции имеет значение **0F**, то есть еще и второй байт.

Примечание. При обсуждении системы команд удобно использовать понятие "основной байт" кода операции. Если КОП состоит из одного байта, то этот байт и является основным байтом. Если КОП состоит из двух байт, то основным следует считать второй байт кода операции.

Число байт	Компонент команды
0 или 1	Префикс команды
0 или 1	Префикс изменения размера адреса
0 или 1	Префикс изменения размера операнда
0 или 1	Префикс замены сегмента
1 или 2	Код операции
0 или 1	Байт MRM - (mod,reg,r/m)
0 или 1	Байт SIB - (scale,index,base)
0,1,2 или 4	Поле для задания адреса
0,1,2 или 4	Непосредственный операнд

Архитектура X86. Формат команд

Байт MRM – (mod,reg,r/m) байт режима адресации

Имеется не во всех командах.

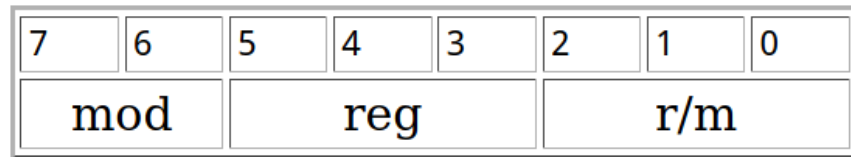
Например, ADD имеет четыре варианта с байтом MRM, еще четыре варианта с "сокращенным" байтом MRM (NNN), и два варианта без байта MRM. Итого десять разных вариантов машинной команды (кодов операции).

Делится на три битовых поля: mod, reg, r/m.

- **reg** - определяет первый операнд команды (операнд-приемник, destination). Задаёт регистр общего назначения.
- **r/m** - определяет второй операнд команды (операнд источник, source). Задаёт регистр, или память совместно с **mod** (32 варианта = 8 для задания регистров и 24 для задания формы и режима адресации памяти).

Бывает и наоборот (зависит от конкретной команды).

В "32-разрядном режиме бита" после MRM может стоять байт режима адресации SIB, увеличивая количество форм задания адреса памяти.



Обозначение NNN

В ряде команд байт MRM используется иначе. Отличие в трактовке поля **reg**. Основной КОП не всегда однозначно определяет команду. Поэтому при некоторых значениях к нему добавляются три бита из поля **reg** байта MRM.

Пример: команды с первым байтом КОП = 80, 81, 82, 83.

Для подобных команд ставится обозначение NNN в колонке "Формат" вместо обозначения MRM. Тогда в команде стоит сокращенный вариант байта MRM, который может задавать только один операнд, что определяется полями (mod) и (r/m).

В ряде команд байт MRM применяется в сокращенном виде так как не требуется задавать два операнда. Поле **reg** не используется. Оговаривается, что reg д.б. = 0.

Пример:

MOV**, коды **C6** и **C7

POP**, код **8F

*группа команд **SET**, коды от **0F 90** до **0F 9F***

Тоже трактуется как NNN.

Архитектура X86. Формат команд

Байт SIB - (scale,index,base)

Второй байт режима адресации.

Позволяет применять еще более сложные формы задания адреса в памяти.

Возможен в команде только в 32-разрядном режиме в том случае, когда есть байт MRM (в полной форме или в форме NNN).

Включается в состав команды только при определенных значениях в полях **mod** и **r/m**.

Поля:

- scale** - масштабный множитель (00 – 1, 01 – 2, 10 – 4, 11 – 8);
- index** - номер индексного регистра (000 – EAX, 001 – ECX, 010 – EDX, 011 – EBX, 100 – *reg2* не используется, 101 – EBP, 110 – ESI, 111 – EDI);
- base** - номер базового регистра (000 – EAX, 001 – ECX, 010 – EDX, 011 – EBX, 100 – ESP, 101 – 32-offset при *mod* = 00 иначе EBP, 110 – ESI, 111 – EDI).

7	6	5	4	3	2	1	0
scale		index			base		

Архитектура X86. Формат команд

Поле для задания адреса

Задается полный адрес, или смещение относительно некоторого адреса (адреса сегмента).
Может по-разному использоваться в разных командах:

- Если есть байт MRM, то поле адреса является приложением к байту MRM и может потребоваться в разных формах задания адреса. От MRM зависит, будет ли в данной команде поле для задания адреса или нет.
- Если нет байта MRM, то поле используется в той конкретной форме, которая предписана для данной команды.

Непосредственный операнд

Поле для непосредственного операнда есть не во всех машинных командах.
Если в формате команды есть поле для задания непосредственного операнда, то это поле в команде всегда будет последним.

Архитектура X86, X86-64. Формат команд

Команды архитектуры x86

- Команды общего назначения
- Системные команды
- Команды сопроцессора (x87 FPU)
- Команды управления состоянием сопроцессора и SIMD
- Команды технологии MMX
- Команды расширения SSE
- Команды расширения SSE2
- Команды расширения SSE3
- Команды расширения 3DNow!

В описаниях команд могут встречаться следующие обозначения:

Byte	- 8-битное целое (байт)
Word	- 16-битное целое (слово)
DWord	- 32-битное целое (двойное слово)
QWord	- 64-битное целое (учетверенное слово)
Float	- вещественное число одинарной точности (32 бита)
Double	- вещественное число двойной точности (64 бита)

Команды общего назначения

Команды передачи данных

MOV	Присваивание
CMOVxx	Условное присваивание
XCHG	Обмен значений
BSWAP	Перестановка байтов
XADD	Обмен и сложение
CMPXCHG	Сравнение и обмен
CMPXCHGB	Сравнение и обмен 8 байтов
PUSH	Поместить значение в стек
POP	Взять значение из стека
PUSHA/PUSHAD	Поместить значения регистров общего назначения в стек
POPA/POPAD	Взять значения регистров общего назначения из стека
IN	Прочитать значение из порта ввода/вывода
OUT	Записать значение в порт ввода/вывода
CWD	Преобразовать Word в DWord
CDQ	Преобразовать DWord в QWord
CBW	Преобразовать Byte в Word
CWDE	Преобразовать Word в DWord в регистре eax
MOVSX	Присвоить и расширить с учетом знака
MOVZX	Присвоить и расширить нулевым значением

Двоичные арифметические команды

ADD	Сложение
-----	----------

Архитектура x86-64. Регистры XMM, YMM

SSE (Streaming SIMD Extensions, потоковое SIMD-расширение процессора) — (Single Instruction, Multiple Data, Одна инструкция — множество данных) набор инструкций, разработанный Intel и впервые представленный в процессорах серии Pentium III как ответ на аналогичный набор инструкций 3DNow! от AMD, который был представлен годом раньше.

В SSE добавлены шестнадцать (для x64) 128-битных регистров, которые называются xmm0 — xmm15. Каждый регистр может содержать четыре 32-битных значения с плавающей запятой одинарной точности.

<https://ru.wikipedia.org/wiki/SSE>

Режим Advanced Vector Extensions (AVX) — расширение системы команд x86 для микропроцессоров Intel и AMD (марте 2008 г.)

Предоставляет различные улучшения, новые инструкции и новую схему кодирования машинных кодов.

Поддержка новых регистров YMM (256 разрядов)

<https://ru.wikipedia.org/wiki/AVX>

Использование Intel AVX: пишем программы завтрашнего дня

<https://habr.com/ru/post/99367/>

Пример простой программы на ассемблере X86-64

```
9      .intel_syntax noprefix
10     .section .rodata
11 msg:
12     #.string      "hello, world!\n"
13     .ascii        "hello, world!\n"
14     #.byte        10
15     .equ          msgLen, .-msg
16
17     .text          # Code
18     .global _start
19 _start:
20     mov     rax, 1      # 1 = write
21     mov     rdi, 1      # 1 = to stdout
22     #lea     rsi, msg[rip] # string to display in rsi
23     lea     rsi, msg     # if _start: without rip
24     mov     rdx, msgLen  # length of the string, without 0
25     syscall            # display the string
26
27     mov     rax, 60     # 60 = exit
28     mov     rdi, 0      # 0 = success exit code
29     syscall            # quit
30
```

Используемые источники

Википедия: Список микропроцессоров Intel

https://ru.wikipedia.org/wiki/Список_микропроцессоров_Intel

Математических сопроцессор Intel 8087 https://ru.wikipedia.org/wiki/Intel_8087

Справочник по командам процессоров X86 (32-х разрядная архитектура):

<http://looch-disasm.narod.ru/refe01.htm>

Структура машинной команды: <http://looch-disasm.narod.ru/refe09.htm>

Команды x86, x86-64: <http://ccfit.nsu.ru/~kireev/lab2/lab2com.htm>

Регистры x86, x86-64: <http://ccfit.nsu.ru/~kireev/lab2/lab2reg.htm>

Архитектура x86-64 (AMD64)



AMD64 Technology

Developer Guides, Manuals & ISA Documents

<https://developer.amd.com/resources/developer-guides-manuals/>

AMD64 Architecture

AMD64 Architecture Programmer's Manual Volumes 1-5

<https://www.amd.com/system/files/TechDocs/40332.pdf>

AMD64 Architecture Programmer's Manual: Volumes 1-5

Рэндал Э. Брайант
Дэвид Р. О'Халларон

Компьютерные системы

Архитектура и программирование

Третье издание



Рэндал Э. Брайант, Дэвид Р. О'Халларон

Компьютерные системы: архитектура и
программирование. 3-е изд. /
– М.: ДМК Пресс, 2022. – 994 с.: ил.



Программирование на ассемблере x64

От начального уровня
до профессионального
использования AVX

Йо Ван Гуй



Йо Ван Гуй. Программирование на
ассемблере x64: от начального
уровня до профессионального
использования AVX. - М.: ДМК
Пресс, 2021. – 332 с.

Исходные тексты примеров к книге:

<https://github.com/Apress/beginning-x64-assembly-programming>

INTRODUCTION TO COMPUTER ORGANIZATION

AN UNDER THE HOOD LOOK AT
HARDWARE AND X86-64 ASSEMBLY

ROBERT G. PLANTZ



Robert G. Plantz.

INTRODUCTION TO COMPUTER
ORGANIZATION. Copyright © 2022 by



**no starch
press**

San Francisco

DIVE INTO SYSTEMS

A Gentle Introduction to Computer Systems

SUZANNE J. MATTHEWS, TIA NEWHALL,
and KEVIN C. WEBB



by Suzanne J. Matthews, Tia Newhall, and
Kevin C. Webb.

DIVE INTO SYSTEMS. Copyright © 2022



**no starch
press**

San Francisco

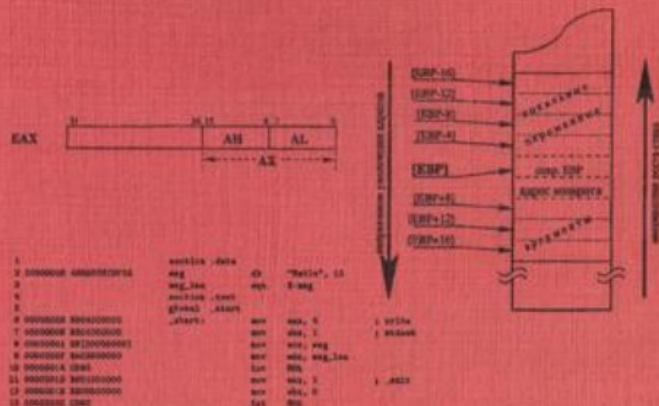
А. В. СТОЛЯРОВ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА NASM ДЛЯ ОС UNIX

Столяров А.В. Программирование на языке ассемблера NASM для ОС Unix: Уч. пособие. - 2-е изд. - М.: МАКС Пресс, 2011. - 188 с.

Сайт книги с электронной версией и примерами:

http://www.stolyarov.info/books/asm_unix





А.В.СТОЛЯРОВ

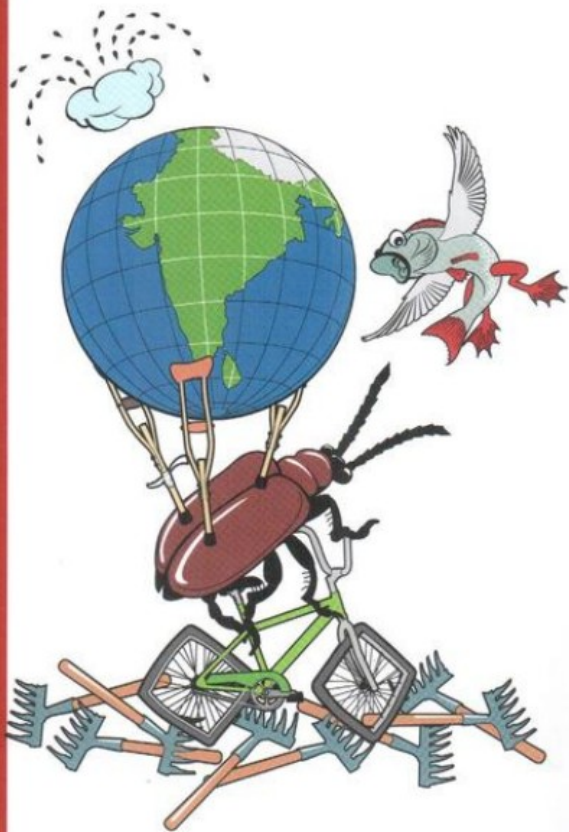
ассемблер nasm • конвенции linux и freebsd
язык си • сборка и контроль версий

**НИЗКОУРОВНЕВОЕ
ПРОГРАММИРОВАНИЕ**

Столяров А.В. Программирование:
введение в профессию. II:
Низкоуровневое программирование.
- М.: МАКС Пресс, 2016. - 496 с.

Сайт книги с электронной версией и примерами:

http://www.stolyarov.info/books/programming_intro/vol2



Столяров А.В. Программирование: введение в профессию. Том 1: Азы программирования. - 2-е изд. - М.: МАКС Пресс, 2021. - 704 с.

Сайт книги с электронной версией и примерами:

http://www.stolyarov.info/books/programming_intro/e2

Гагарина Л. Г., Кононова А. И.



АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И АССЕМБЛЕР

с приложением методических указаний
к лабораторным работам

Учебное пособие

- базовые определения и понятия архитектуры вычислительных систем;
- тридцатидвухбитный и шестидесятичетырёхбитный режимы x86;
- кроссплатформенный синтаксис ассемблера AT&T;
- базовый набор команд, адресация и структура x86;
- процесс сборки программы «под микроскопом»;
- приемы прикладного программирования на ассемблере и C++;
- стыковка кода на ассемблере и C++.

Оценка «Отлично»!

Библиотека студента

Гагарина Л. Г., Кононова А. И.

Архитектура вычислительных систем и Ассемблер с приложением методических указаний к лабораторным работам. Учебное пособие.

— М.: СОЛОН-Пресс, 2019. - 368 с.