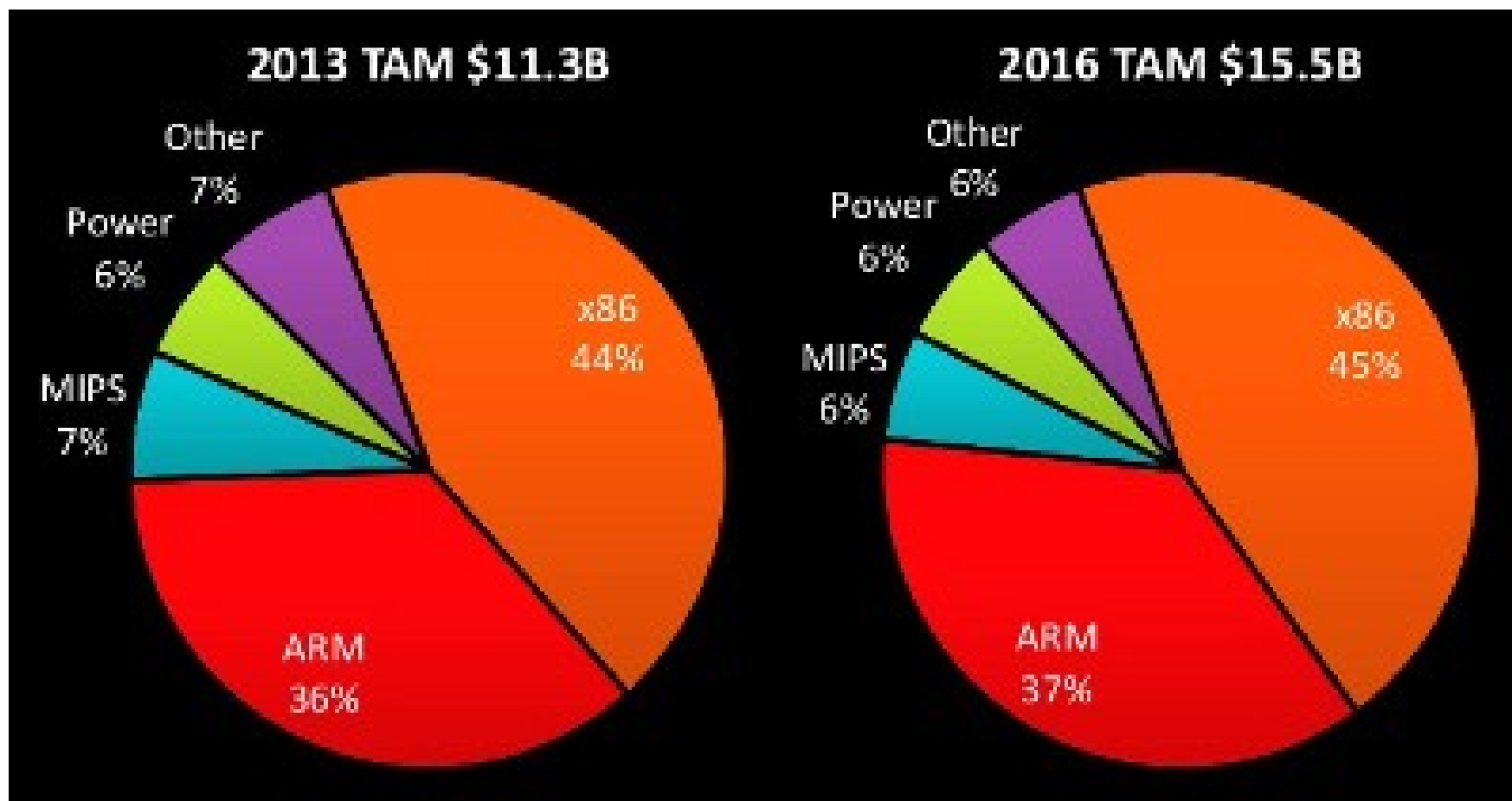


Низкоуровневое программирование

Процессорные архитектуры

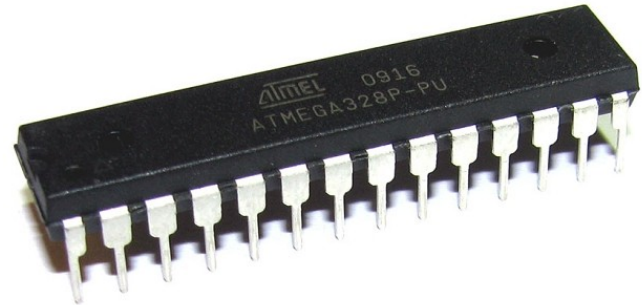
- I386, x86_64 (x86) (1978) - Intel, AMD, ...
- ARM (1985) - консорциум
- PowerPC (1992) - IBM
- MIPS (1981) - консорциум
- SPARC (1987) - Sun
- Alpha (1992) - DEC, Compaq, HP
- Itanium (2001) — Intel
- Микроконтроллеры (AVR, Z80, 8051, MaxQ, ...)

Процессорные архитектуры



Микроконтроллер

- Процессор
- ОЗУ
- ПЗУ (EEPROM, Flash)
- GPIO
- Коммуникационные интерфейсы (UART, I2C, SPI)
- Таймеры
- АЦП



System-On-Chip

- Микроконтроллер по характеристикам приближающийся к компьютерам:
 - 512 и более MiB RAM
 - Несколько ядер
 - Интегрированный GPU

Принципы фон Неймана

- Однородность памяти — команды и данные хранятся в одной памяти
- Адресность — каждая ячейка памяти имеет свой адрес
- Программное управление — инструкции процессора исполняются последовательно
- Двоичное кодирование

Гарвардская архитектура

- Для команд и данных используются различные шины
- Широко используется в микроконтроллерах

Структурная схема процессора

- Управляющее устройство — выборка и дешифрация команды, переход на следующую инструкцию
 - Регистр счетчика команд (PC, IP)
 - Регистр слова состояния (FLAGS)
 - Другие регистры...
- Арифметико-логическое устройство
- Регистры общего назначения (РОН — GPR), регистры плавающей точки

Классификации систем команд

- Битность (8, 16, 32, 64)
- Адресность (0-адресные ... 3-адресные)
- Работа с памятью
- Сложность (CISC, RISC, VLIW)

Битность процессора

- Битность процессора определяется размером регистров общего назначения: А, В, С — 8-битные РОН 8080
- АХ, ВХ, СХ, ... - 16-битные РОН 8086
- ЕАХ, ЕВХ, ЕСХ — 32-битные РОН i386
- RAX, RBX, RCX — 64-битные РОН x86_64
- Регистры могут объединяться в пары при выполнении некоторых инструкций

Инструкция процессора

- Элементарное неделимое действие, выполняемое процессором
- В программе на ассемблере записывается в виде:

`LABEL: OP CODE OPERANDS`

- Детали записи варьируются от архитектуры к архитектуре

Инструкции процессора

- PDP-11

ADD (R1)+, R0 ; R0 += *R1++

- X86 (Intel Syntax)

ADD EAX, [EBX + ECX * 4] ; EAX += EBX[ECX]

- X86 (AT&T Syntax)

ADDL (%EBX, %ECX, 4), %EAX

- ARM

LDR R0, [R1, #4]! ; R0 = *R1++

Адресность

- Адресность определяется числом операндов у типичной инструкции выполняющей бинарную операцию (ADD)
- 0-адресные: для вычислений используется стек
FADD ; x86 FPU: сложение двух чисел на стеке регистров FPU
- 1-адресные: неявный операнд и результат — аккумулятор
FADD [EBX] ; x86 FPU сложить ST(0) и [EBX]

Адресность

- Двухадресные

ADD EBX, ECX ; EBX += ECX

- Трехадресные

ADD R1, R2, R3 ; ARM: R1 = R2 + R3

Работа с памятью

- Memory-memory: оба аргумента из памяти и результат в памяти
ADD (R1)+, (R2) ; PDP-11
- Register-memory: не более одного аргумента из памяти
ADD DWORD PTR [EBX], 12
- Register-register: оба аргумента и результат должны быть в регистрах

Сложность системы команд

- CISC (Complex Instruction Set Computing)
 - «исторический набор инструкций»
 - Много форматов команд, переменная длина (от 1 до 15 байт на x86)
 - Ориентация на написание программ человеком
- RISC (Reduced Instruction Set Computing)

RISC процессоры

- Мало форматов инструкций — простой декодер инструкций
- Фиксированная длина инструкций
- Однородные РОН, «много» РОН
- Формат операций регистр-регистр
- (Как правило) одна инструкция за цикл
- Ассемблер для компилятора, а не для человека!

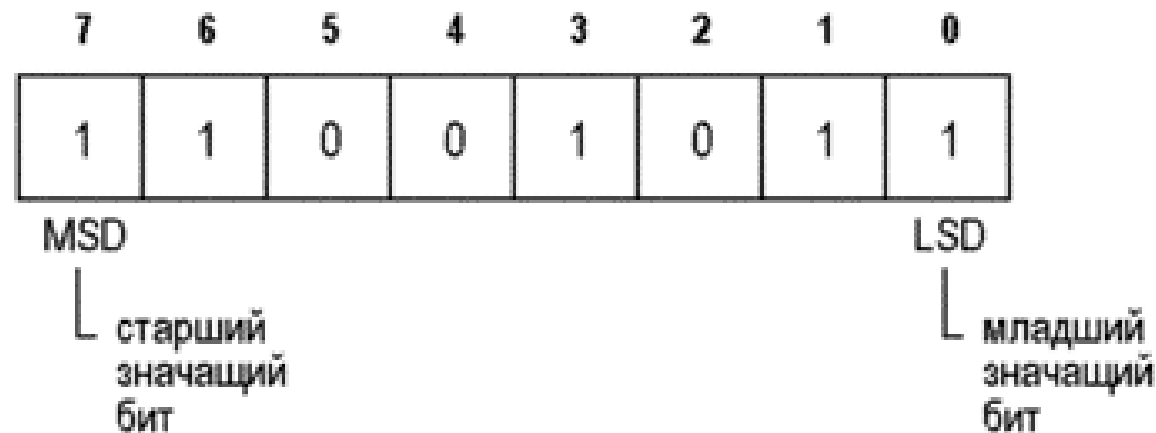
Типы данных

- Беззнаковые целые
- Знаковые целые
- Адреса — беззнаковые целые
- Смещения — знаковые целые
- Вещественные

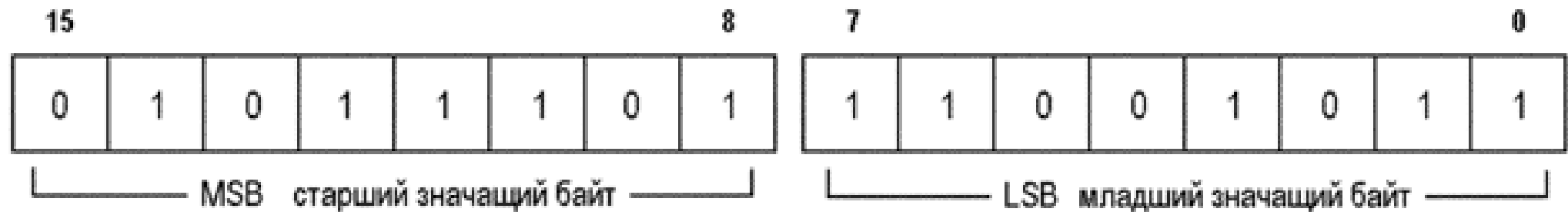
Беззнаковые типы

- Биты нумеруются от младшего к старшему

Байт (8 бит)



Слово (16 бит)

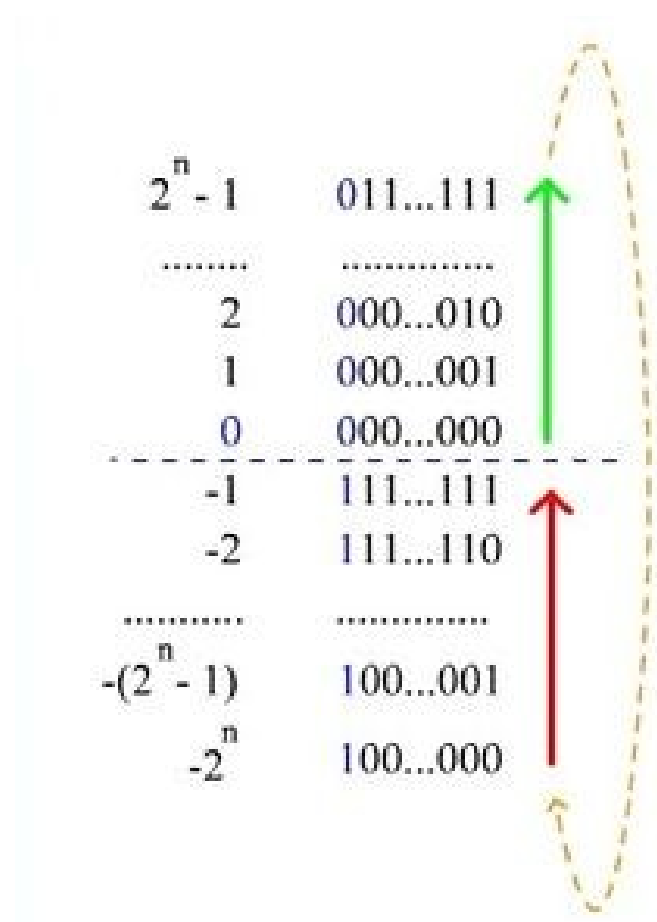


Беззнаковые типы

- Byte (0 .. 255) — unsigned char
- Word (0 .. 65536) — unsigned short
- Dword (0 .. 4294967296) — unsigned int
- Qword (0 .. 18446744073709551616) — unsigned long long

Знаковые целые числа

- Могут иметь длину 8, 16, 32, 64 бита
- Представляются в дополнительном коде
- $-x == \sim x + 1$
- Сложение, вычитание, сдвиг влево выполняются одинаково для знаковых и беззнаковых целых



$2^n - 1$	011...111
.....
2	000...010
1	000...001
0	000...000
-----	-----
-1	111...111
-2	111...110
.....
$-(2^n - 1)$	100...001
-2^n	100...000

Byte order

- Память адресуется побайтно
- Целые числа большей длины могут размещаться в памяти по-разному
- Преобразование прозначно для программиста
- Little-endian: x86
- Big-endian: SPARC
- Переключаемые: ARM, PPC (Android — LE, iOS - LE)

Byte order

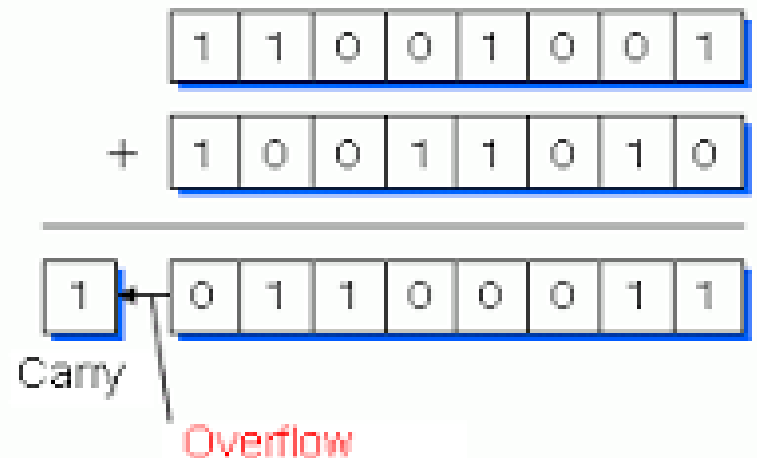
	Low address				High address			
Address	0	1	2	3	4	5	6	7
Little-endian	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Big-endian	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Memory content	0x11	0x22	0x33	0x44	0x55	0x66	0x77	0x88
64 bit value on Little-endian				64 bit value on Big-endian				
0x8877665544332211				0x1122334455667788				

Арифметические флаги

- В процессоре есть специальный регистр FLAGS (EFLAGS, RFLAGS), в котором находятся как флаги, управляющие работой, так и флаги результата операций
- Флаги Z, S (N), C, O (V) модифицируются в зависимости от значения результата операции
- Для каждой инструкции специфицировано какие флаги и как модифицируются

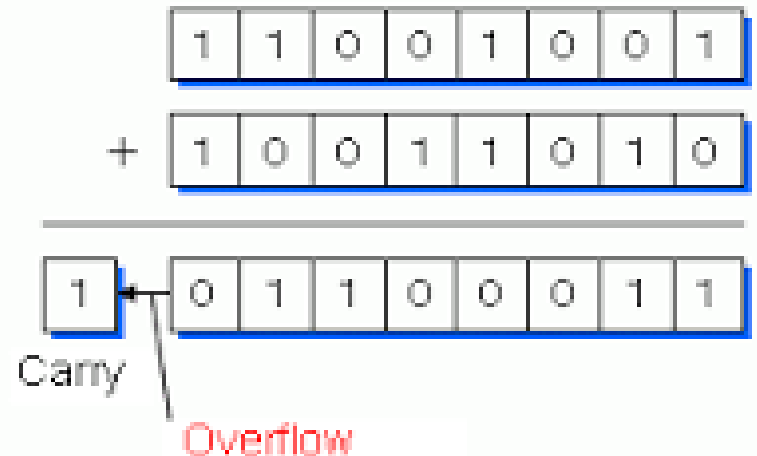
Арифметические флаги

- Z — устанавливается в 1, если результат операции равен 0
- S (N) — устанавливается в 1, если результат операции отрицателен (т. е. старший бит результата)
- C - «перенос» или «заем»
результат операции над
двумя беззнаковыми
числами не представим
как беззнаковое число того
же размера



Арифметические флаги

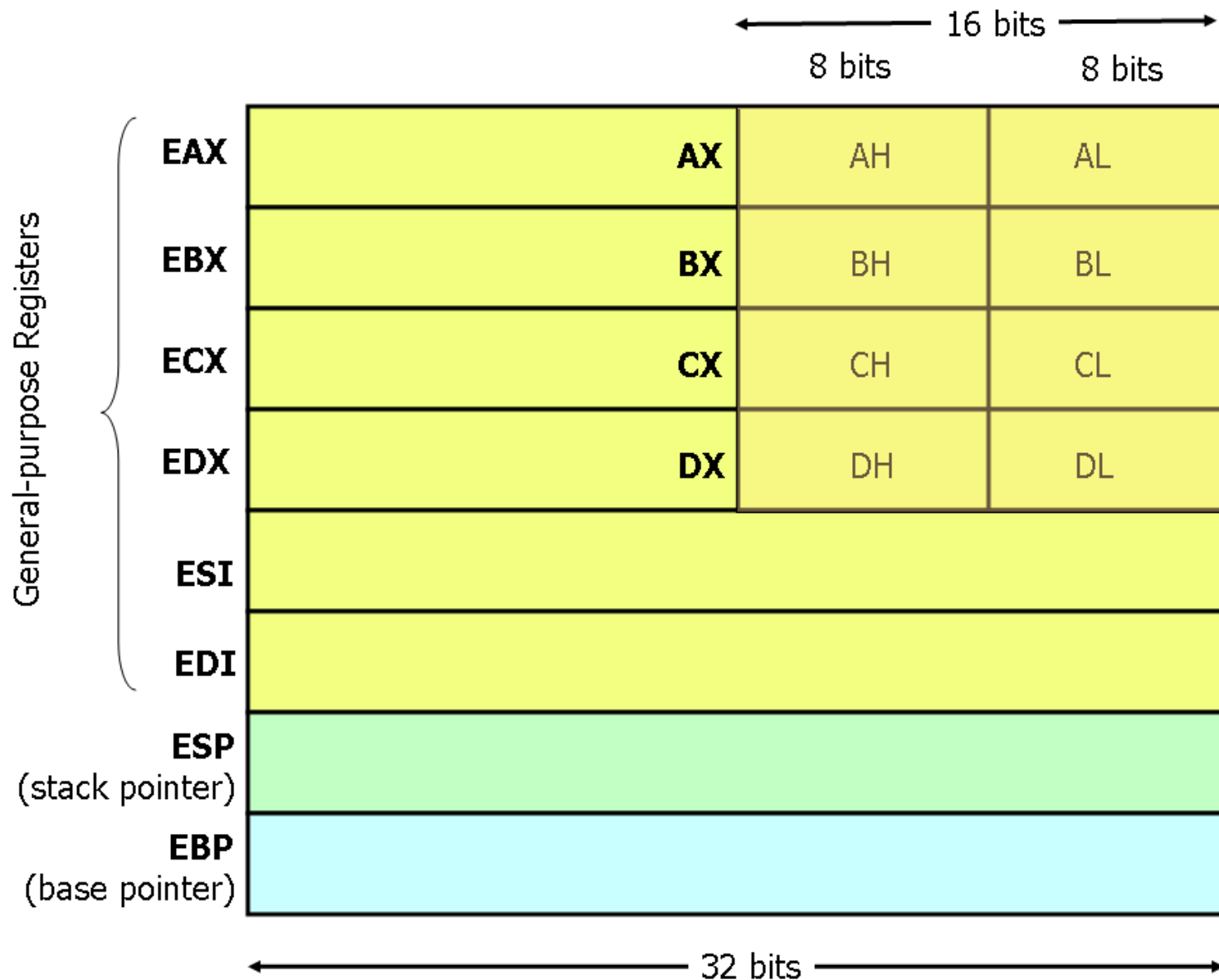
- $O(V)$ — флаг «переполнения» - результат операции над двумя знаковыми числами не представим как знаковое число того же размера
- Пусть $R = A + B$, числа — 8-битные
- $O = (A \wedge R) \& (B \wedge R) \& 0x80$
- $O = C6 \wedge C7$



Условные переходы

- Инструкции условных переходов проверяют значения флагов, установленных ранее
- Условия описываются в предположении, что предыдущая инструкция была CMP R1, R2, то есть R1 — R2 без сохранения результата
- ja — переход, если беззнаковое больше, т. е. !z && !c
- jae — переход, если больше или равно (!c)
- jg — переход, если знаковое больше (!z && s == 0)
- je (jz) — переход, если нуль (z)

Регистры x86



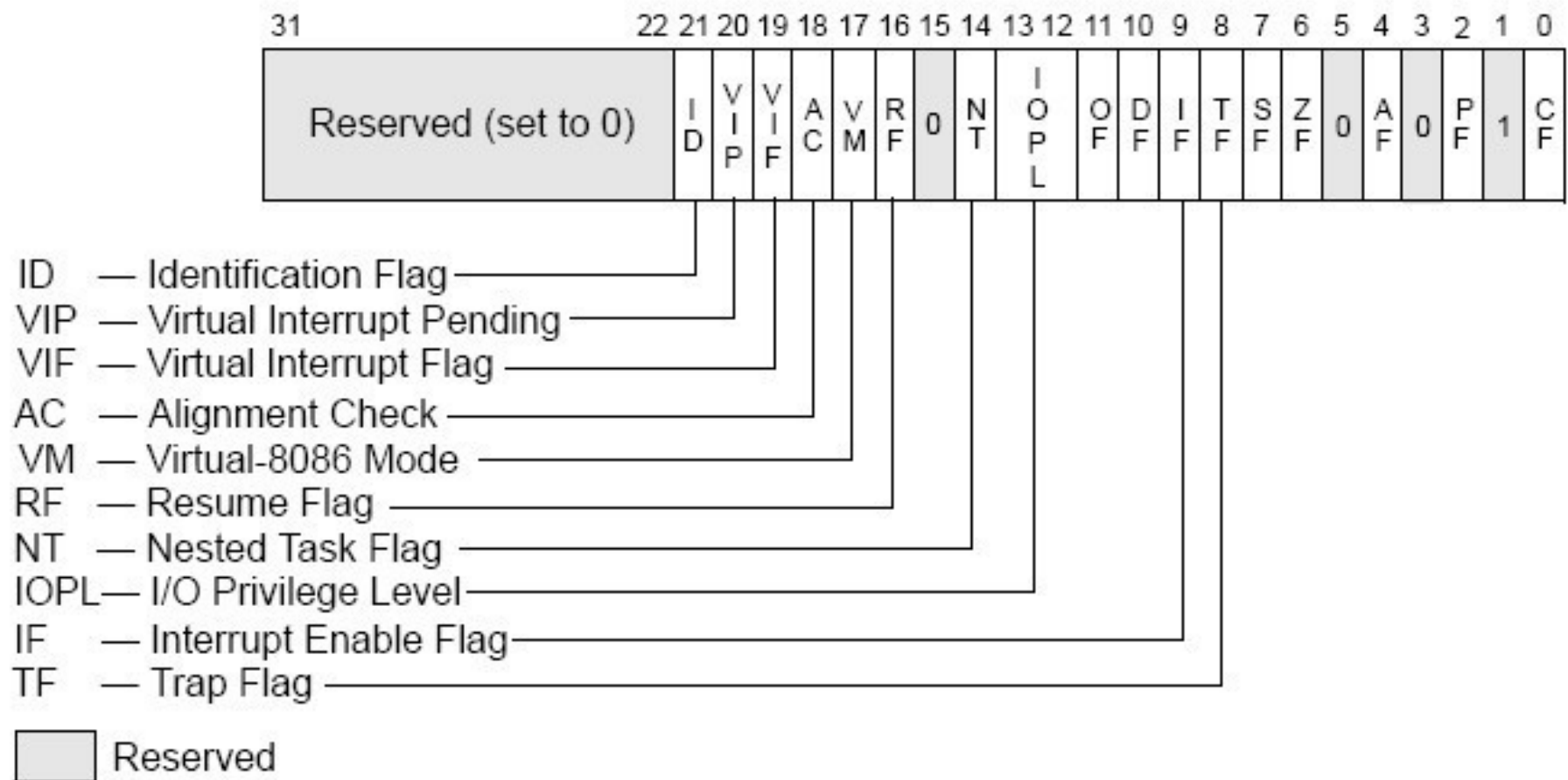
Специфика регистров

- ESP — stack pointer, указатель на вершину стека
- EBP — указатель на текущий стековый фрейм, может использоваться как регистр для вычислений
- ESI, EDI — неявно используются в строковых инструкциях
- EAX, EDX — используются при умножении и делении
- ECX — счетчик повторов или сдвигов (CL)

Управляющие регистры

- EIP (Instruction Pointer) — указывает на следующую инструкцию, которая будет выполняться
- EFLAGS
- CR0, CR1, ...

EFLAGS



Режимы адресации

- Immediate (непосредственный)

MOV EAX, 42

- Absolute (прямой)

VAR1: DD 0 ; глобальная переменная
MOV EAX, VAR1

- Indirect

MOV [ESI], EAX ; разыменование указателя

Режимы адресации

- Memory (base + offset)

MOV EAX, [EBP + 4]

MOV [GLOBARR + EBX * 4], EAX

MOV BX, [EBP + EDX * 2 + 4]

Стек вызовов

- ESP указывает на первую занятую ячейку стека
- Стек растет в сторону уменьшения адресов
- PUSH EAX ; ESP -= 4, [ESP] = EAX
- POP EAX ; EAX = [ESP], ESP += 4
- CALL ADDR ; PUSH EIP, EIP = ADDR
- RET ; POP EIP

Передача параметров

- Способ передачи параметров и возврата результата определяется «соглашениям о передаче параметров» (calling convention)
- Например, параметры передаются на стеке, результат возвращается на регистрах

Пример

- long long addll(long long v1, long long v2)

```
addll:  MOV EAX, [ESP + 4]
        MOV EDX, [ESP + 8]
        ADD EAX, [ESP + 12]
        ADC EDX, [ESP + 16]
        RET
```

Пример

```
VAR1  DQ  0x1234567887654321  
VAR2  DQ  0x1212AABBCCDDEEFF
```

```
    PUSH  DWORD PTR [VAR2 + 4]  
    PUSH  DWORD PTR [VAR2]  
    PUSH  DWORD PTR [VAR1 + 4]  
    PUSH  DWORD PTR [VAR1]  
    CALL  addll  
    ADD   ESP, 16
```

Карта стека

ESP + 16

V2 (high 32 bits)

ESP + 12

V2 (low 32 bits)

ESP + 8

V1 (high 32 bits)

ESP + 4

V1 (low 32 bits)

ESP

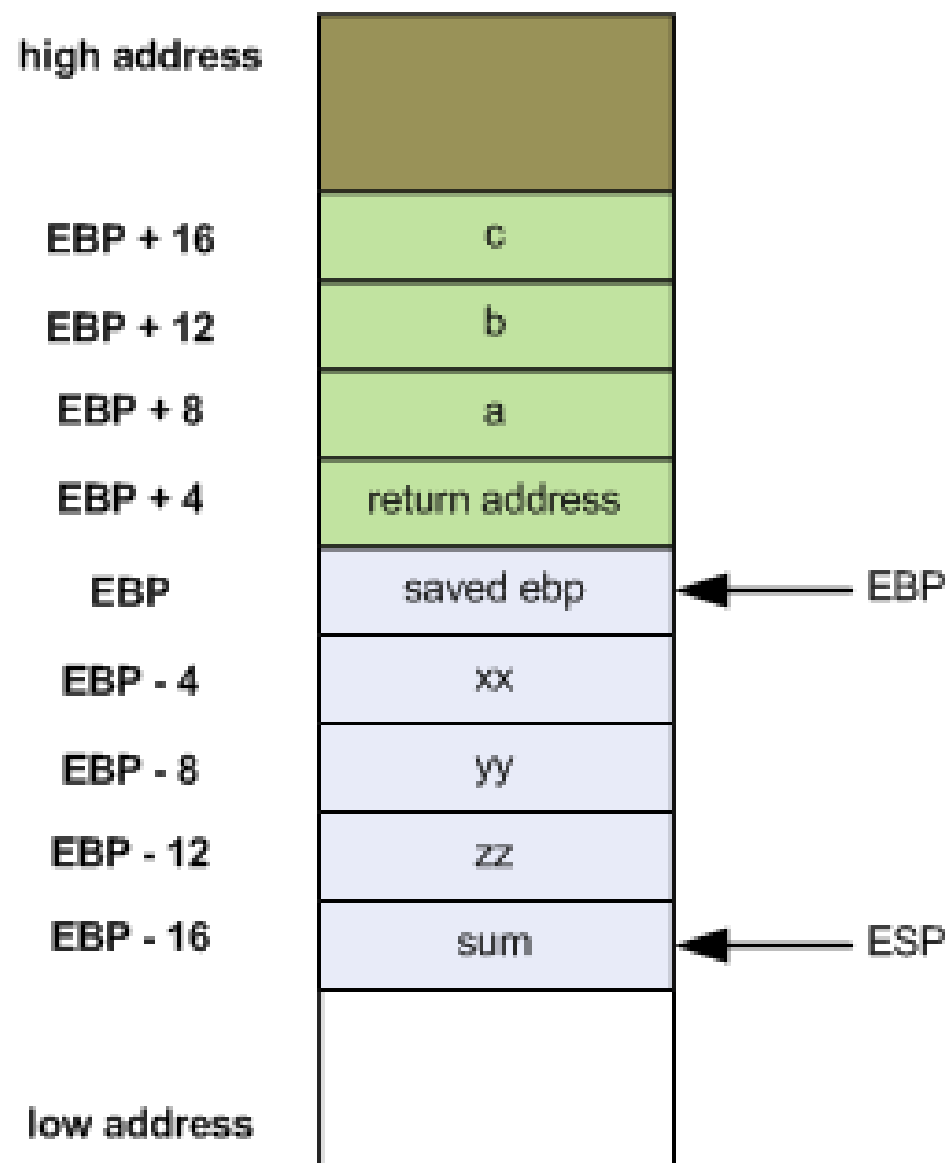
Return address

Цепочки вызовов (stack frames)

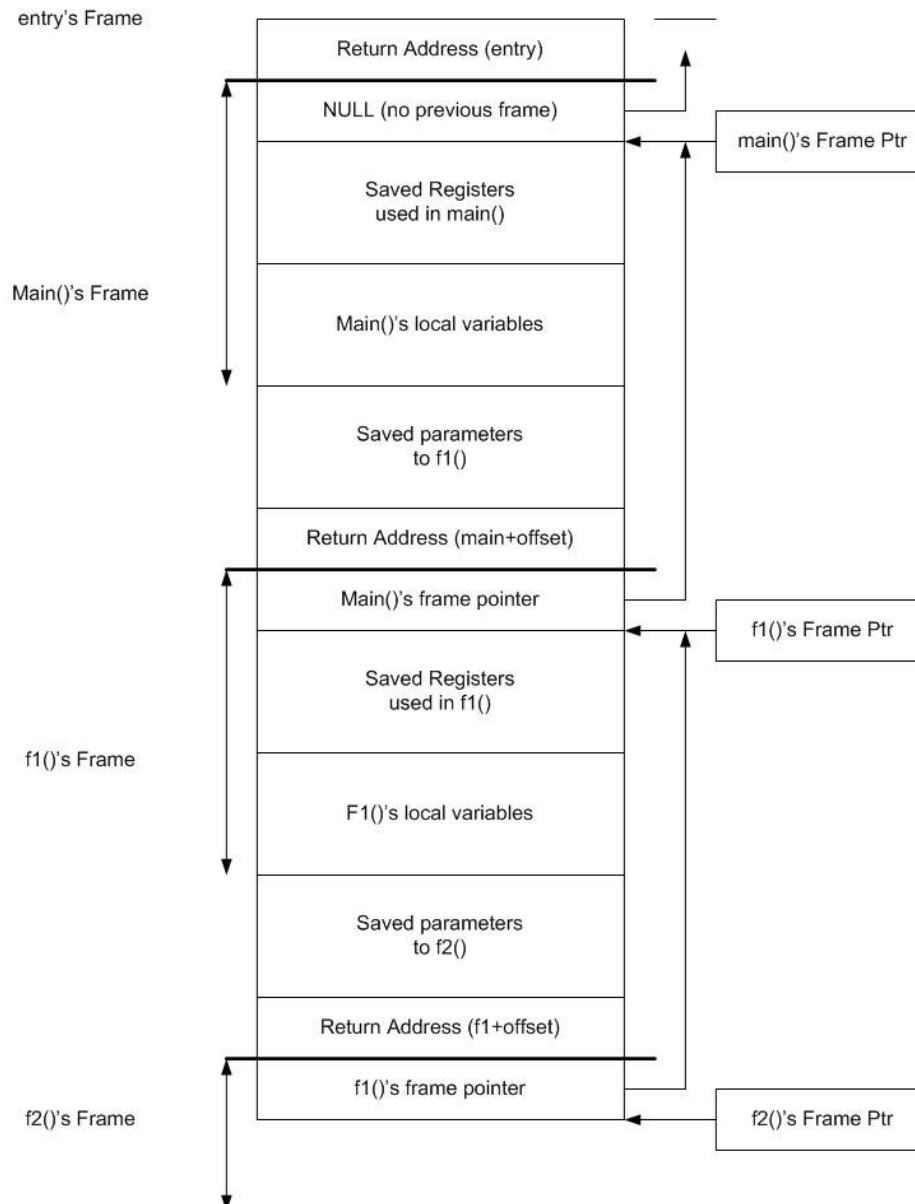
- Локальные переменные размещаются «ниже» адреса возврата

```
PUSH EBP
MOV EBP, ESP
```

```
MOV ESP, EBP
POP EBP
RET
```



Цепочки вызовов



Соглашение о вызовах cdecl

- Аргументы в стек заносятся в обратном порядке (т.е. лежат в прямом)
- Стек очищается вызывающей функцией
- Результат возвращается на регистрах EAX, EAX/EDX или ST(0)
- EAX, EDX, ECX могут изменяться вызванной функцией
- Остальные регистры должны быть сохранены
- (Стек выравнивается по границе 16 байт)

Соглашение о вызовах stdcall (Win32)

- Аргументы в стек заносятся в обратном порядке (т.е. лежат в прямом)
- Стек очищается вызываемой функцией
- Результат возвращается на регистрах EAX, EAX/EDX или ST(0)
- EAX, EDX, ECX могут изменяться вызванной функцией
- Остальные регистры должны быть сохранены

Пример

- long long addll(long long v1, long long v2)

```
addll:  MOV EAX, [ESP + 4]
        MOV EDX, [ESP + 8]
        ADD EAX, [ESP + 12]
        ADC EDX, [ESP + 16]
        RET 16
```