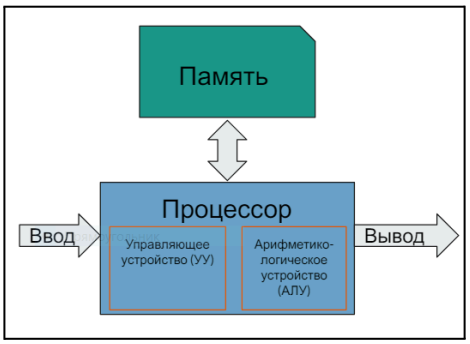
Вопросы для подготовки к экзамену по дисциплине
«Машинно-зависимые языки программирования»

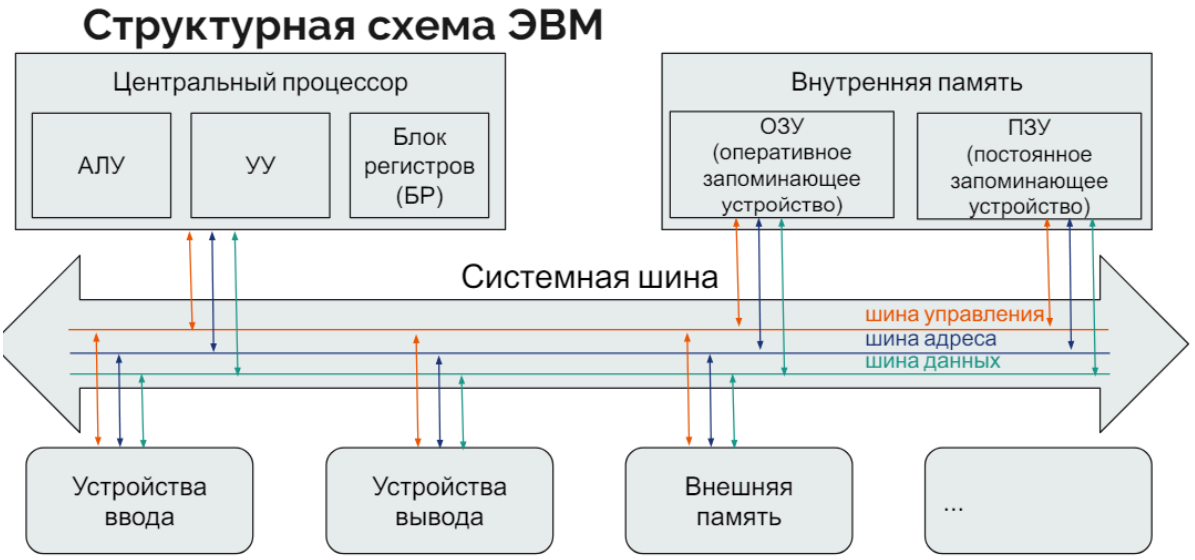
1. Архитектура фон Неймана, принципы фон Неймана.

История: В 1946 году американский учёный Герман Голдстайн опубликовал доклад математика Джона фон Неймана «Предварительное рассмотрение логической конструкции электронно-вычислительного устройства».



Принципы фон Неймана:

1. Использование двоичной системы счисления в вычислительных машинах.
2. Программное управление ЭВМ.
3. Память компьютера используется не только для хранения данных, но и программ.
4. Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.
5. Возможность условного перехода в процессе выполнения программы.



Центральный процессор (ЦП) - основной компонент, выполняющий инструкции. Он состоит из:

- **Арифметико-логическое устройство (АЛУ)** - выполняет математические и логические операции.
- **Устройство управления** - интерпретирует инструкции из памяти и управляет их выполнением.

Память - хранит данные и инструкции. Важная особенность архитектуры фон Неймана заключается в том, что и данные, и инструкции хранятся в одной и той же памяти.

Устройства ввода-вывода - средства взаимодействия компьютера с внешним миром, включая клавиатуру, мышь, дисплей, принтеры и др.

Шины - коммуникационные пути, по которым данные передаются между ЦП, памятью и устройствами ввода-вывода.

Минимальная адресуемая единица памяти - байт

Машинное слово - машинно-зависимая величина, измеряемая в битах, равная разрядности регистров и шины данных
Параграф - 16 байт

Виды современных архитектур ЭВМ

- x86-64
- ARM
- IA64
- MIPS (включая Байкал)
- VLIW (например, Эльбрус)

2. Машинные команды, машинный код. Понятие языка ассемблера.

Машинная команда - инструкция (в двоичном коде) из аппаратно определённого набора, которую способен выполнять процессор.

Машинный код - система команд конкретной вычислительной машины, которая интерпретируется непосредственно процессором.

Язык ассемблера - машинно-зависимый язык программирования низкого уровня, команды которого прямо соответствуют машинным командам.

Операционная система — программное обеспечение, управляющее компьютерами (включая микроконтроллеры) и позволяющее запускать на них прикладные программы.

DOS (disk operating system, дисковая операционная система) - семейство операционных систем для мейнфреймов, начиная с IBM System/360, и ПК.

Наиболее распространённая разновидность для ПК — MS-DOS
Основные свойства:

- отсутствие ограничений для прикладных программ на вмешательство в работу ОС и доступ к периферийному оборудованию;
- однозадачность;
- текстовый режим работы.

3. Виды памяти ЭВМ. Запуск и исполнение программы.

1. **Оперативная память** (ОЗУ, RAM - DRAM (Dynamic) - SRAM (Static))

2. **Постоянная память** (ПЗУ, ROM - PROM (Programmable), EPROM (Erasable Programmable), EEPROM (Electrically Erasable Programmable))

3. **Флеш-память** (NAND Flash, NOR Flash)

4. **Кэш-память** (Кэш L1, Кэш L2, Кэш L3)

5. **Виртуальная память** (Использует часть дискового пространства для расширения оперативной памяти, позволяет запускать большие программы и обрабатывать большие объемы данных)

6. **Регистры** (Быстрая память внутри процессора, используется для хранения промежуточных данных и инструкций во время выполнения операций.)

7. **Внешняя память** (HDD - жесткие диски, SSD - твердотельные накопители, Оптические диски, Ленты)

8. **Специализированная память** (VRAM - видеопамять, Buffer Memory - Буфер памяти)

Исполняемый файл - файл, содержащий программу в виде, в котором она может быть исполнена компьютером (то есть в машинном коде).

Получение в 2 шага: компиляция и линковка.

Компилятор - программа для преобразования исходного текста другой программы на определённом языке в объектный модуль.

Компоновщик (**линковщик**, **линкер**) - программа для связывания нескольких объектных файлов в исполняемый.

ЗАПУСК

В DOS и Windows - расширения .EXE и .COM

Последовательность запуска программы операционной системой:

1. Определение **формата** файла.
2. Чтение и разбор **заголовка**.
3. **Считывание разделов** исполняемого модуля (файла) в **ОЗУ** по необходимым адресам.
4. Подготовка к **запуску**, если требуется (загрузка библиотек).
5. **Передача** управления на точку **входа**.

Отладчик - программа для автоматизации процесса отладки. Может

выполнять трассировку,

отслеживать,

устанавливать или изменять значения переменных в процессе выполнения кода,

устанавливать или удалять контрольные точки или условия остановки

.COM (command) - простейший формат исполняемых файлов DOS и ранних версий Windows:

- не имеет заголовка;
- состоит из одной секции, не превышающей 64 Кб;
- загружается в ОЗУ без изменений;
- начинает выполняться с 1-го байта (точка входа всегда в начале).

ПОСЛЕДОВАТЕЛЬНОСТЬ запуска COM-программы:

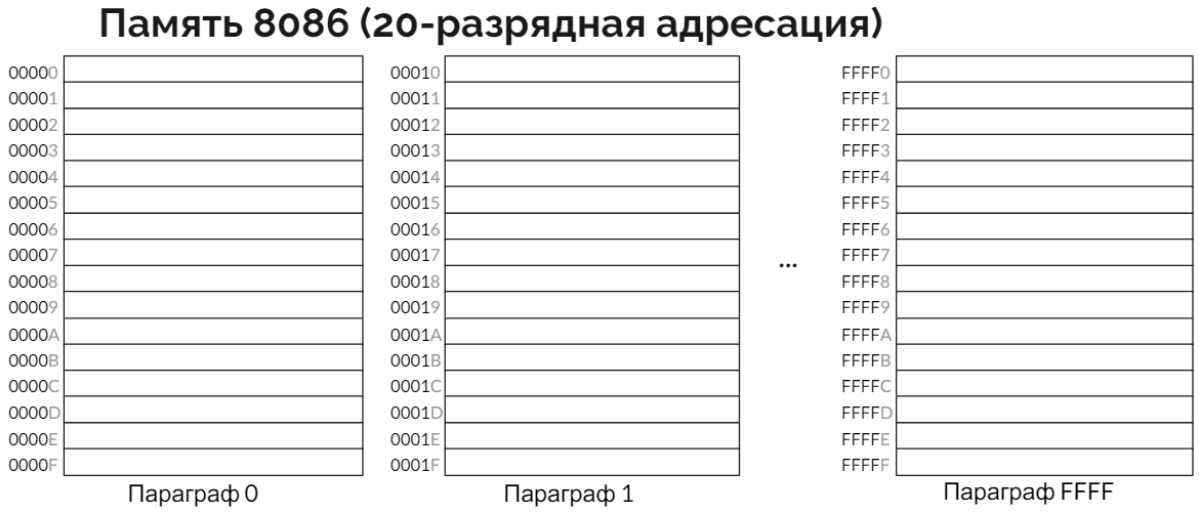
1. Система выделяет **свободный сегмент** памяти нужного размера и заносит его адрес **во все сегментные регистры** (CS, DS, ES, FS, GS, SS).
2. В **первые 256 (100h)** байт этого сегмента записывается служебная структура DOS, описывающая программу - **PSP**.
3. Непосредственно за ним загружается **содержимое COM-файла** без изменений.
4. **Указатель стека** (регистр **SP**) устанавливается на **конец сегмента**.
5. В стек записывается **0000h** (начало **PSP** - адрес возврата для возможности завершения командой ret).
6. Управление передаётся по адресу **CS:0100h**

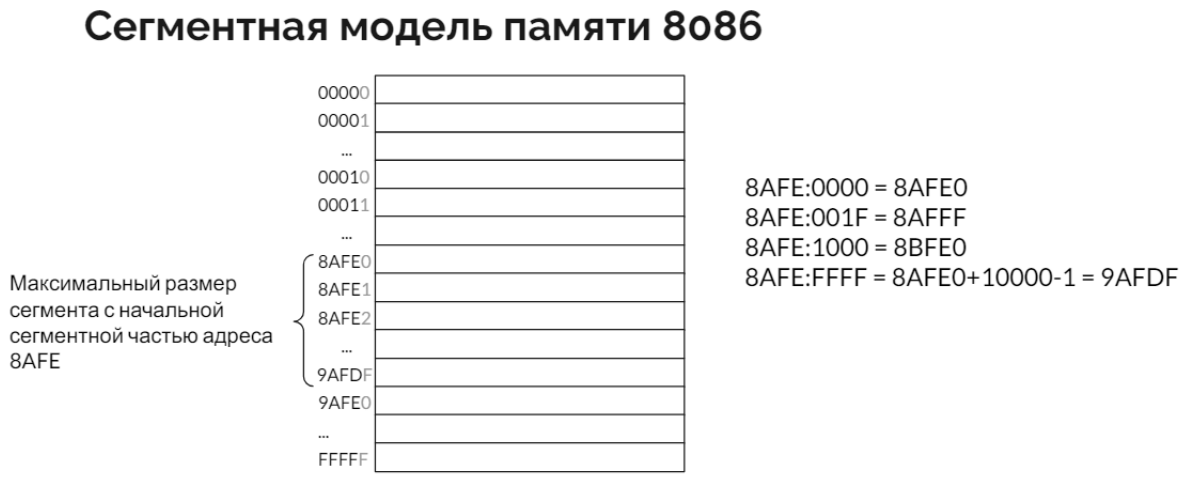
4. Сегментная модель памяти в архитектуре 8086.

Сегментация — это процесс, при котором основная память компьютера логически разделяется на различные сегменты, и каждый сегмент имеет свой собственный базовый адрес.

Сегмент — это логическая единица памяти, которая может иметь длину до 64 килобайт.

Сегментация используется для повышения скорости выполнения компьютерной системы, чтобы процессор мог легко и быстро извлекать и выполнять данные из памяти.





5. Процессор 8086. Регистры общего назначения.

КОМАНДЫ процессора 8086

- Команды *пересылки* данных
- *Арифметические* и *логические* команды
- Команды *переходов*
- Команды работы с *подпрограммами*
- Команды *управления* процессором

Восемь регистров общего назначения (каждый разрядностью 16 бит)

Каждый регистр общего назначения может использоваться для **хранения** 16-битового **значения**,
в **арифметических** и **логических** операциях, может выполняться **обмен** между **регистром** и **памятью** (запись из регистра в память и наоборот).

AX - Этот регистр всегда используется в операциях умножения или деления и является также одним из тех регистров, который можно использовать для наиболее эффективных операций (арифметических, логических или операций перемещения данных).

BX - может использоваться для ссылки на ячейку памяти (указатель), т.е. 16-битовое значение, записанное в BX, может использоваться в качестве части адреса ячейки памяти, к которой производится обращение.
(По умолчанию, когда BX используется в качестве указателя на ячейку памяти, он ссылается на нее относительно сегментного регистра DS.)

CX - Специализация регистра CX - использование в качестве счетчика при выполнении циклов.
(Уменьшение значения счетчика и цикл - это часто используемый элемент программы, поэтому в процессоре 8086 используется специальная команда для того, чтобы циклы выполнялись быстрее и были более компактными. Эта команда называется LOOP. Инструкция LOOP вычитает 1 из CX и выполняет переход, если содержимое регистра CX не равно 0.

DX - это единственный регистр, которые может использоваться в качестве указателя адреса ввода-вывода в командах IN и OUT. Фактически, кроме использования регистра DX нет другого способа адресоваться к портам ввода-вывода с 256 по 65535.

Другие уникальные качества регистра DX относятся к операциям деления и умножения. Когда вы делите 32-битовое делимое на 16-битовый делитель, старшие 16 битов делимого должны быть помещены в регистр DX (младшие 16 битов делимого

должны быть помещены в регистр AX). После выполнения деления остаток также сохраняется в регистре DX (частное от деления будет записано в AX).

6. Процессор 8086. Сегментные регистры. Адресация в реальном режиме. Понятие сегментной части адреса и смещения.

CS ⇒ Сегмент кода — область памяти, содержащая код программы. За её адресацию отвечает регистр **CS**. Командой MOV изменить невозможно, меняется автоматически по мере выполнения команд.

DS ⇒ Сегмент данных — область памяти, содержащая переменные и константы программы. Основной регистр для адресации к сегменту данных — **DS**, при необходимости дополнительных сегментов данных задействуются ES, FS, GS.

SS ⇒ Сегмент стека. Для адресации стека используется регистр **SS**.

Логический адрес состоит из двух частей: сегментного значения (16 бит) и смещения (16 бит). "сегмент:смещение"

Физический адрес: Для вычисления физического адреса процессор использует формулу - сегметное_значение*16 + смещение

[SEG]:[OFFSET] => физический адрес:

1. SEG необходимо побитово сдвинуть на 4 разряда влево (или умножить на 16, что тождественно)
2. К результату прибавить OFFSET

7. Процессор 8086. Регистр флагов.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CF	-	PF	-	AF	-	ZF	SF	TF	IF	DF	OF	IOPL		NT	-

Флаги состояния:

CF (carry flag) - флаг переноса
если при сложении - перенос старшего бита или
если при вычитании - заём

PF (parity flag) - флаг чётности
если в рез-те операции младший байт содержит чётное количество единиц.

AF (auxiliary carry flag) - вспомогательный флаг переноса
Устанавливается при переносе из младшей тетрады (4 бит) в старшую тетраду или заёме из старшей тетрады в младшую. (Используется в основном для бинарно-десятичных операций.)

ZF (zero flag) - флаг нуля
если результат операции равен нулю.

SF (sign flag) - флаг знака
если результат операции отрицателен

OF (overflow flag) - флаг переполнения
если результат арифм. операции вышел за пределы допустимого диапазона значений для знаковых чисел

Управляющий флаг:

DF (direction flag) - флаг направления
если DF = 1 - то декремент
если DF = 0 - то инкремент

Системные флаги:

IF (interrupt enable flag) - флаг разрешения прерываний
если установлен, процессор будет реагировать на аппаратные прерывания.
если сброшен, прерывания игнорируются.

TF (trap flag) - флаг трассировки
если установлен, процессор переходит в режим поэтапной трассировки, что позволяет выполнять инструкции по одной для целей отладки.

IOPL (I/O privilege flag) - уровень приоритета ввода-вывода
NT (nested task) - флаг вложенности задач

8. Команды пересылки данных.

MOV <приёмник>, <источник>

Источник: непосредственный операнд (константа, включённая в машинный код), РОН, сегментный регистр, переменная (ячейка памяти).

Приёмник: РОН, сегментный регистр, переменная (ячейка памяти).

*РОН - регистр общего назначения

- MOV AX, 5
- MOV BX, DX
- MOV [1234h], CH
- MOV DS, AX

- ~~MOV [0123h], [2345h]~~
- ~~MOV DS, 1000h~~

PUSH

Команда PUSH уменьшает значение регистра стека на размер операнда (2 или 4 байта) и копирует содержимое операнда в память по адресу SS

```
push  dx
```

POP

Команда POP копирует содержимое ячейки памяти по адресу SS
в операнд и увеличивает значение регистра стека на размер операнда (2 или 4 байта).

```
pop   [bx]
```

! С помощью команды POP нельзя загрузить из стека CS.

XCHG

Команда XCHG позволяет регистрам общего назначения и ячейкам памяти **обмениваться** своим содержимым

! Команда XCHG позволяет регистрам общего назначения и ячейкам памяти обмениваться своим содержимым

```
xchg ax,bx
xchg dx,[100h]
```

IN

Команда IN пересылает байт, слово или двойное слово из заданного порта в регистр (AL, AX или EAX).

```
mov  dx,03DFh
in   al,dx
in   ax,60h
```

OUT

Команда OUT пересылает байт, слово или двойное слово из регистра (AL, AX или EAX) в порт.

```
sub   ax,ax
mov   dx,03DFh
out   dx,ax
out   60h,al
```

LEA

Команда LEA вычисляет исполнительный адрес (смещение) второго операнда и записывает его в регистр, заданный первым операндом

```
lea ax,[bx+8]
lea bx,buf[si]
```

MOVS

Команда MOVS на самом деле не является командой процессора.

Когда в тексте программы встречается эта команда, компилятор:

1. вычисляет размерность ее операндов
2. и на основании вычислений подставляет на ее место одну из реальных команд процессора MOVSB, MOVSW или MOVSD.

Копирование **строки**

```
movs   str1, es:str2
```

Копирование строк **байтов**

```
mov    si,offset str1
mov    di,offset str2
cld
movsb
```

Копирование строк **слов**

```
mov    si,offset str1
mov    di,offset str2
cld
movsw
```

Копирование строк **двойных слов**

```
mov    si,offset str1
mov    di,offset str2
cld
movsd
```

9. Команда сравнения.

CMP <приёмник>, <источник>
Источник - число, регистр или переменная
Приёмник - регистр или переменная; не может быть переменной одновременно с источником
Вычитает источник из приёмника ->
результат **никуда не сохраняется**,
выставляются флаги **CF, PF, AF, ZF, SF, OF**

CF (carry flag) - флаг переноса
PF (parity flag) - флаг чётности
AF (auxiliary carry flag) - вспомогательный флаг переноса
ZF (zero flag) - флаг нуля
SF (sign flag) - флаг знака
OF (overflow flag) - флаг переполнения

! Не допускается сравнения значений двух ячеек памяти. Такое сравнение должно производиться через какой-нибудь регистр.

```
cmp    ax,0012h
cmp    cx,dx
cmp    byte ptr [bx],09h
```

10. Команды условной и безусловной передачи управления.

JMP <операнд>

- Передаёт управление в другую точку программы (на другой адрес памяти), не сохраняя какой-либо информации для возврата.
- Операнд - непосредственный адрес (вставленный в машинный код), адрес в регистре или адрес в переменной.

Виды переходов для команды JMP

- short (короткий) -128 .. +127 байт
- near (ближний) в том же сегменте (без изменения регистра CS)
- far (дальний) в другой сегмент (с изменением значения в регистре CS)
- Для короткого и ближнего переходов непосредственный операнд (константа) прибавляется к IP
- Операнды – регистры и переменные заменяют старое значение в IP (CS:IP)

Команды условных переходов Jcc

- Переход типа short или near
- Обычно используются в паре с CMP
- Термины “выше” и “ниже” - при сравнении беззнаковых чисел
- Термины “больше” и “меньше” - при сравнении чисел со знаком

Команда	Описание	Флаг условия перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнения	OF = 0
JS	Есть знак	SF = 1
JNS	Нет знака	Sf = 0
JE, JZ	Если равно/если ноль	ZF = 1
JNE, JNZ	Не равно/не ноль	ZF = 0
JP, JPE	Есть чётность / чётное	PF = 1
JNP, JPO	Нет чётности / нечётное	PF = 0
JCXZ	CX = 0	-

Команда	Описание	Флаг перехода	Тип
JB JNAE JC	Если ниже Если не выше и не равно Если перенос	CF = 1	Беззнаковый
JNB JAE JNC	Если не ниже Если выше или равно Если нет переноса	CF = 0	Беззнаковый
JBE JNA	Если ниже или равно Если не выше	CF = 1 или ZF = 1	Беззнаковый
JA JNBE	Если выше Если не ниже и не равно	CF = 0 и ZF = 0	Беззнаковый
JL JNGE	Если меньше Если не больше и не равно	SF <> OF	Знаковый

JGE JNL	Если больше или равно Если не меньше	SF = OF	Знаковый
JLE JNG	Если меньше или равно Если не больше	ZF = 1 или SF <> OF	Знаковый
JG JNLE	Если больше Если не меньше и не равно	ZF = 0 и SF = OF	Знаковый

11. Арифметические команды.

ADD **<приёмник>, <источник>** - выполняет арифметическое сложение приёмника и источника.
Сумма помещается в приёмник, источник не изменяется.
Приемник: регистр, переменная
Источник: регистр, переменная, число.
Источник приемника и и источника должны быть одинаковыми(если источник - число, оно расширяется до размера приемника).
Меняет флаги: OF, SF, ZF, AF, CF и PF

SUB **<приёмник>, <источник>** - арифметическое вычитание источника из приёмника.
Аналогично команде ADD.

ADC **<приёмник>, <источник>** - складывает применик, источник и флаг CF.

SBB **<приёмник>, <источник>** - вычитает из приемника источник и флаг CF.

MUL **<источник>** - беззнаковое умножение. Умножаются источник и AL/AX/EAX/RAX, в зависимости

от размера источника. Результат помещается в AX либо DX:AX/EDX:EAX/RDX:RAX.

```
MUL BL – AX = AL*BL
MUL BX – DS:AX=AX*BL, причем в AX – младшая часть
                      результата, в DX – старшая
```

IMUL <источник>;
IMUL <приёмник>, <источник>;
IMUL <приёмник>, <источник1>, <источник2> - знаковое умножение

DIV <источник> - целочисленное беззнаковое деление. Делится AL/AX/EAX/RAX на источник.
Результат помещается в AL/AX/EAX/RAX, остаток - в AH/DX/EDX/RDX.

IDIV <источник> - знаковое деление
INC <приёмник> - инкремент на 1
DEC <приёмник> - декремент на 1

12. Двоично-десятичная арифметика.

Двоично-десятичное число - цифры десятичного числа представляются в двоичном виде.

Неупакованное двоично-десятичное число - цифра в байте: байт от 00h до 09h.
Упакованное двоично-десятичное число - двузначное число в байте: байт от 00h до 99h

Число 26(=1Ah) представляется как 26h. Буквы (A-F) в таком формате не используются.
При сложении упакованных чисел надо корректировать результат, например:
19h + 1h = 1Ah -> 20h

Команды:
DAA (**D**ecimal **A**djust AL after **A**ddition)
Корректировка после сложения. Выполняется после команд ADD, ADC и корректирует AL под упакованное двоично-десятичное.

DAS (**D**ecimal **A**djust AL after **S**ubtraction)
Корректировка после вычитания. Выполняется после команд SUB, SBB и корректирует AL под упакованное двоично-десятичное.

DAA и DAS должны выполнятся сразу после нормальных команд, т.к. действуют в зависимости от флагов.

Пример:

```
mov al, 48h
mov bl, 15h
add al, bl  // В AL лежит 5Dh == 93
DAA         // Преобразует AL в 63h (48+15)
```

AAA, AAS - аналогично DAA и DAS, но для ASCII формата (неупакованное).
Расширяет результат до AX

Пример:

```
mov al, 9h
mov bl, 2h
add al, bl // В AL запишется 0Bh == 11
aaa        // Запишет в AX 0101 == 11
```

AAM, AAD - Корректировка для умножения и деления неупакованных.

13. Команды побитовых операций. Логические команды.

Операции побитового сдвига:

- SAR, SAL – арифметический сдвиг
- SHR, SHL – логический сдвиг
- SAL тождественна SHL
- SHR зануляет старший бит, SAR – сохраняет (знак)
- ROR, ROL - циклический сдвиг вправо/влево
- RCR, RCL - циклический сдвиг через флаг переноса CF

Команды выставляют флаг OF в зависимости от выдвигаемого бита.

Операции над битами и байтами:

- BT <база>, <смещение> - считать в CF значение бита из битовой строки
- BTS <база>, <смещение> - установить бит в 1
- BTR <база>, <смещение> - сбросить бит в 0
- BTC <база>, <смещение> - инвертировать бит
- BSF <приёмник>, <источник> - прямой поиск бита (от младшего разряда)
- BSR <приёмник>, <источник> - обратный поиск бита (от старшего разряда)
- SETcc <приёмник> - выставляет приёмник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc

14. Команды работы со строками.

Строка-источник - DS:SI, строка-приёмник - ES:DI.

За один раз обрабатывается один байт (слово).

После выполнения каждой из этих команд DI и SI:

- увеличиваются на 1, если флаг DF = 0
- уменьшаются на 1, если флаг DF = 1

MOVS:

Из DS:SI копирует в ES:DI байт/слово/д.слово

MOVSB (операндов нет) - работа с байтом.
MOVSW (операндов нет) - работа со словом.
MOVSD (операндов нет) - работа с двойным словом.
MOVS <приёмник>, <источник> - компилятор решает, что поставить в зависимости от размеров операндов.

CMPS/CMPSB/CMPSW <приёмник>, <источник> - сравнение

SCAS/SCASB/SCASW <приёмник> - сканирование (сравнение с AL/AX, устанавливает флаги как CMP)

LODS/LODSB/LODSW <источник> - чтение (в AL/AX)

STOS/STOSB/STOSW <приёмник> - запись (из AL/AX)

Префиксы повторения:

REP/REPE/REPZ - все три - одна команда

Префиксы повторяют следующую за ним команду. После каждого повторения команды регистр CX уменьшается на 1 и, если он стал равен нулю или в результате выполнения команды флаг ZF стал равен 0, происходит выход из цикла.

REPNE/REPNZ
Аналогично REP, но выходит когда ZF стал равен 1 (или CX станет 0 как в REP)

```
Запись в строку/массив 5 байт со 10h
mov cx, 5
mov al, 10h
rep stos
```

15. Команда трансляции по таблице.

XLAT [адрес]
XLATB
Помещает в AL байт из таблицы по адресу DS:BX со смещением относительно начала таблицы, равным AL.

Адрес XLAT не влияет ни на что! (в презентации написано “Если в адресе явно указан сегментный регистр, он будет использоваться вместо DS.” но я проверял, ничего не меняется)

```
table db 100h dup(?)
…
mov ax, TABLE_SEGMENT(тут таблица)
mov ds, ax
mov   bx,offset table
mov   al,1Fh
xlatb

запишет в AL 32(1Fh + 1, индекс все-таки) байт из table
```

16. Команда вычисления эффективного адреса.

LEA <приёмник>, <источник>

OFFSET - директива, LEA - команда.

Вычисляет эффективный адрес источника и помещает его в приёмник.
Позволяет вычислить адрес, описанный сложным методом адресации.

В источнике можно использовать только BP, BX, SI, DI.
Нельзя в одном выражении использовать и BP и BX
Нельзя в одном выражении использовать и SI и SI
Иногда используется для быстрых арифметических вычислений:

```
mov bx, 6
lea ax, [bx + 12] // ax = 6 + 12 = 18

lea ax, [bx * 5] // ax = 6 * 5 = 30 (в 8086 не компилируется, увы)
lea ax, [bx + si * 5]
```

Эти вычисления занимают меньше памяти, чем соответствующие MOV и ADD, и не изменяют флаги.

17. Структура программы на языке ассемблера. Модули. Сегменты

Программа на языке ассемблера состоит из строк, имеющих следующий вид:
метка команда/директива операнды ; комментарий
Все поля необязательны.

Если после метки стоит команда, после нее надо ставить ":", чтобы сообщить ассемблеру что надо создать переменную с адресом этой команды.
Если после метки стоит директива, ":" не ставится.
Например:
метка label *тип*
Метке присваивается адрес следующей команды или данных.
метка equ *выражение*
Присваивает метке значение выражения, работает как #define в C

Директивы распределения памяти
имя_переменной D* значение

DB - определить байт;
DW - определить слово (2 байта);
DD - определить двойное слово (4 байта);
DF - определить 6 байт (адрес в формате 16-битный селектор: 32-битное смещение);
DQ - определить учетверенное слово (8 байт);
DT - определить 10 байт (80-битные типы данных, используемые FPU).

Поле значения может содержать одно или несколько чисел, строк символов (взятых в одинарные или двойные кавычки), операторов ? и DUP, разделенных запятыми. Все установленные таким образом данные окажутся в выходном файле, а имя переменной будет соответствовать адресу первого из указанных Значений.

```
text_string db 'Hello world!'
number dw 7
table db 1,2,3,4,5,6,7,8,9,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
float_number dd 3.5e7
```

А еще программа на языке ассемблера состоит из сегментов(данных, кода, стека).
Описание сегмента - директива SEGMENT:

```
имя SEGMENT [READONLY] [выравнивание] [тип] [разрядность] ['класс']
 …
имя ENDS
```

READONLY. Если этот операнд присутствует, MASM выдаст сообщение об ошибке на все команды, выполняющие запись в данный сегмент. Другие ассемблеры этот операнд игнорируют.

Выравнивание. Указывает ассемблеру и компоновщику, с какого адреса может начинаться сегмент:
BYTE - с любого адреса;
WORD - с четного адреса;
DWORD - с адреса, кратного 4;
PARA - с адреса, кратного 16 (граница параграфа) - по умолчанию;
PAGE - с адреса, кратного 256.

Тип. Выбирает один из возможных типов комбинирования сегментов:
PUBLIC (иногда используется синоним MEMORY) означает, что все такие сегменты с одинаковым именем, но разными классами будут объединены в один; **Объединяются в один друг за другом.**

STACK - то же самое, что и PUBLIC, но должен использоваться для сегментов стека, потому что при загрузке программы сегмент, полученный объединением всех сегментов типа STACK, будет использоваться как стек; **PUBLIC но для стека.**

COMMON сегменты с одинаковым именем также объединяются в один, но не последовательно, а по одному и тому же адресу, следовательно, длина суммарного сегмента будет равна не сумме длин объединяемых сегментов, как в случае PUBLIC и STACK, а длине максимального. Таким способом иногда можно формировать оверлейные программы; **Накладываются друг на друга.**

AT - выражение указывает, что сегмент должен располагаться по фиксированному абсолютному адресу в памяти. Результат выражения, использующегося в качестве операнда для AT, равен этому адресу, деленному на 16.

Например: segment at 40h - сегмент, начинающийся по абсолютному адресу 0400h. Такие сегменты обычно содержат только метки, указывающие на области памяти, которые могут потребоваться программе; **Сегмент в конкретном месте.**

PRIVATE (значение по умолчанию) - сегмент такого типа не объединяется с другими сегментами. **Не объединяются.**

Разрядность. Этот операнд может принимать значения USE16 и USE32.
USE16:
- Размер сегмента не больше 64 Кб.
- Все команды считаются 16-битными.

USE32:
- Размер сегмента не больше 4 Гб.
- Все команды считаются 32-битными.

USE16 по умолчанию, при условии что перед .MODEL не применялась директива задания допустимого набора команд .386 или старше.

Класс сегмента - это любая метка, взятая в одинарные кавычки. Все сегменты с одинаковым классом, даже сегменты типа PRIVATE, будут расположены в исполняемом файле непосредственно друг за другом.

Директива ASSUME.

Директива ASSUME `регистр:имя сегмента` устанавливает значение сегментного регистра по умолчанию. Эта директива не изменяет значений сегментных регистров, а только позволяет ассемблеру проверять допустимость ссылок и самостоятельно вставлять при необходимости префиксы переопределения сегментов.

Модули.

Модули - файлы исходного кода.

Директивой END завершается любая программа. В роли необязательного операнда выступает метка определяющая адрес, с которого начинается выполнение программы.

Если в программе несколько модулей, только один может содержать точку входа.

Директива PUBLIC делает метку/переменную доступной для других модулей.

Директива EXTRN описывает метку, определенную в другом модуле (с помощью PUBLIC). Тип (BYTE, WORD, DWORD, FWORD, QWORD, TBYTE, имя структуры, FAR, NEAR, ABS) должен соответствовать типу метки в том модуле, где она была установлена.

18. Виды ассемблеров. Intel-синтаксис, AT&T-синтаксис.

Виды ассемблеров:

- MASM
- TASM
- NASM
- FASM
- YASM
- as

Intel-синтаксис:

1. Приёмник находится слева от источника.
2. Название регистров зарезервировано (нельзя использовать метки с именами eax, ebx и т. д.)

Основные отличия AT&T от Intel-синтаксиса:

1. Имена регистров предваряются префиксом %.
2. Обратный порядок операндов: вначале источник, затем приёмник.
3. Размер операнда задается суффиксом, замыкающим инструкцию.
4. Числовые константы записываются в Си-соглашении.
5. Для получения смещения метки используется префикс $.
6. Комментарии с “#”, а не “;”. “;” позволяет писать несколько команд в одной строке.

19. Подпрограммы. Объявление, вызов.

Объявление.

```
метка proc язык тип USES регистры ; TASM
ИЛИ
метка proc тип язык USES регистры ; MASM/WASM
ret
метка endp
```

Метка - имя.
Тип может принимать значения NEAR и FAR, и если он указан, все команды RET в теле процедуры будут заменены соответственно на RETN и RETF.

USES - список регистров, значения которых изменяет процедура. Ассемблер помещает в начало процедуры набор команд PUSH, а перед командой RET - набор команд POP, так что значения перечисленных регистров будут восстановлены.

CALL - вызов процедуры.

CALL <операнд>

Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента(т.е если процедура ближняя или дальняя)

Передает управление на значение аргумента.

20. Подпрограммы. Возврат управления.

RET/RETN/RETF <число>

Загружает из стека адрес возврата, увеличивает SP
Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров.
RETN - ближний возврат, достанет из стека только IP
RETF - дальний возврат, достанет из стека IP и CS

21. Макроопределения.

Макроопределение (макрос) - именованный участок программы, который ассемблируется каждый раз, когда его имя встречается в тексте программы.

● Определение:

```
имя MACRO параметры …... ENDM
```

● Пример:

```
load_reg MACRO register1, register2
push register1
pop register2
ENDM
```

Директива присваивания

Директива присваивания служит для создания целочисленной макропеременной или изменения её значения и имеет формат: `Макроимя = Макровыражение`

- Макровыражение (или Константное выражение) - выражение, вычисляемое препроцессором, которое может включать целочисленные константы, макроимена, вызовы макрофункций, знаки операций и круглые скобки, результатом вычисления которого является целое число
- Операции: арифметические (+, -, *, /. MOD), логические, сдвигов, отношения

Преимущества макросов: (Зубков)

Вместо кода с использованием макроопределения можно оформить этот же участок кода в виде процедуры и вызывать его командой CALL - если процедура вызывается больше одного раза, этот вариант программы займет меньше места, но вариант с макроопределением станет выполняться быстрее, так как в нем не будет лишних команд CALL и RET. Однако скорость выполнения – не главное преимущество макросов. В отличие от процедур макроопределения могут вызываться с параметрами

Директивы отождествления EQU, TEXTEQU

Директива для представления текста и чисел:

```
Макроимя EQU нечисловой текст и не макроимя ЛИБО число
Макроимя EQU <Операнд>
Макроимя TEXTEQU Операнд
```

Пример:

```
X EQU [EBP+8]
MOV ESI,X
```

Макрооперации

- % - вычисление выражение перед представлением числа в символьной форме
- <> - подстановка текста без изменений
- & - склейка текст
- ! - считать следующий символ текстом, а не знаком операции
- ;; - исключение строки из макроса

Блоки повторения

- REPT число … ENDM - повтор фиксированное число раз
- IRP или FOR: IRP form, … ENDM
- Подстановка фактических параметров по списку на место формального
- IRPC или FORC: IRPC form,fact … ENDM
- Подстановка символов строки на место формального параметра
- WHILE: WHILE cond … ENDM

Директивы условного ассемблирования

- IF

```
IF c1 …
ELSEIF c2
…
ELSE
…
ENDIF
```

- IFB - истинно, если параметр не определён
- IFNB - истинно, если параметр определён
- IFIDN , - истинно, если строки совпадают
- IFDIF , - истинно, если строки разные
- IFDEF/IFNDEF - истинно, если имя объявлено/не объявлено

Директивы управления листингом

- Листинг - файл, формируемый компилятором и содержащий текст ассемблерной программы, список определённых меток, перекрёстных ссылок и сегментов.
- TITLE, SUBTTL - заголовок, подзаголовок на каждой странице
- PAGE высота, ширина
- NAME - имя программы
- .LALL - включение полных макрорасширений, кроме ;;
- .XALL - по умолчанию
- .SALL - не выводить тексты макрорасширений
- .NOLIST - прекратить вывод листинга

Комментарии

```
comment @
… многострочный текст…
@
```

22. Стек. Аппаратная поддержка вызова подпрограмм.

Стек - организованный специальным образом участок памяти, который используется для временного хранения переменных, передачи параметров вызываемым подпрограммам и сохранения адреса возврата при вызове процедур и прерываний. (по Зубкову)

Стек

- LIFO/FILO (last in, first out) - последним пришёл, первым ушёл
- Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний
- SP - указатель на вершину стека
- В x86 стек "растёт вниз", в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента

Команды непосредственной работы со стеком

- PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- POPA - загрузить регистры из стека (SP игнорируется)
- PUSHF - поместить в стек содержимое регистра флагов
- POPF - загрузить регистр флагов из стека

CALL - вызов процедуры

CALL <операнд>

- Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передаёт управление на значение аргумента.

RET - возврат из процедуры

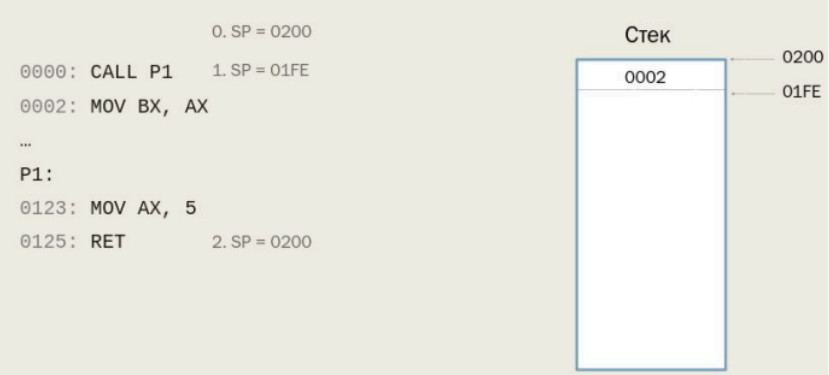
RET/RETN/RETF <число>

- Загружает из стека адрес возврата, увеличивает SP
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

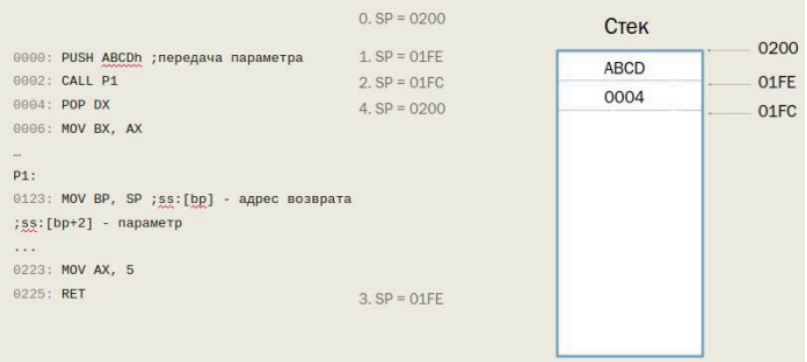
BP – base pointer

- Используется в подпрограмме для сохранения "начального" значения SP
- Адресация параметров
- Адресация локальных переменных

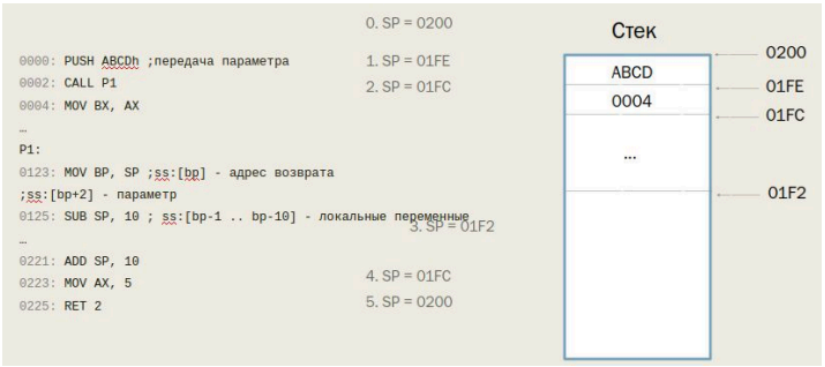
Пример вызова подпрограммы №1



Пример вызова подпрограммы №2



Пример вызова подпрограммы №3



Использование стека подпрограммами

Стековый кадр (фрейм) — механизм передачи аргументов и выделения временной памяти с использованием аппаратного стека. Содержит информацию о состоянии подпрограммы.

Включает в себя:

- параметры
- адрес возврата (обязательно)
- локальные переменные

23. Соглашения о вызовах. Понятие, основные виды соглашений.

Соглашение о вызовах определяют как функция вызывается, как функция управляет стеком и стековым кадром, как аргументы передаются в функцию, как функция возвращает значения.

Описания технических особенностей вызова подпрограмм, определяющие:

- способ передачи параметров подпрограммам;
- способ передачи управления подпрограммам;
- способ передачи результатов выполнения из подпрограмм в точку вызова;
- способ возврата управления из подпрограмм в точку вызова.

Распространённые соглашения

- cdecl32 — соглашение о вызовах, используемое компиляторами для языка Си на 32-разрядных системах.
    1. Аргументы функций передаются через стек, справа налево.
    2. Аргументы, размер которых меньше 4-х байт, расширяются до 4-х байт.
    3. Очистку стека производит вызывающая программа.
    4. Возврат параметров 1, 2, 4 байта (целые числа, указатели) - через EAX.
    5. Возврат больших структур, массивов, строк - указателем через EAX.

    Перед вызовом функции вставляется код, выполняющий следующие действия:

    - сохранение значений регистров, используемых внутри функции;
    - запись в стек аргументов функции.

    После вызова функции вставляется код, выполняющий следующие действия:

    - очистка стека;
    - восстановление значений регистров
- pascal

    Параметры помещаются в стек в порядке слева направо (в противоположность cdecl), и вызываемый объект несет ответственность за их удаление из стека.

    Возврат результата работает следующим образом:

    - Порядковые значения возвращаются в форматах AL (8-битные значения), AX (16-битные значения), EAX (32-битные значения) или DX:AX (32-битные значения в 16-битных системах).
    - Реальные значения возвращаются в DX:BX:AX.
    - Значения с плавающей запятой (8087) возвращаются в ST0.

    - Указатели возвращаются в EAX в 32-битных системах и в AX в 16-битных системах.
    - Строки возвращаются во временном месте, указанном символом @Result.
- stdcall (Win32 API)

  Вызываемый объект отвечает за очистку стека, но параметры помещаются в стек в порядке справа налево, как в соглашении о вызовах _cdecl. Регистры EAX, ECX и EDX предназначены для использования внутри функции. Возвращаемые значения сохраняются в регистре EAX.
- fastcall

  Передает первые два аргумента (оцениваются слева направо) в ECX и EDX. Остальные аргументы помещаются в стек справа налево. Главным отличием от двух соглашениях выше является то, что аргументы кладутся в регистры, если это возможно, что позволяет увеличить скорость вызова функции, потому что обратиться к регистру быстрее, чем к стеку.
- thiscall - вызов нестатических методов C++ (this через ECX)

  Это соглашение о вызовах используется для вызова нестатических функций-членов C++. В зависимости от компилятора и от того, использует ли функция переменное количество аргументов, используются две основные версии этого вызова.

  Для компилятора GCC этот вызов почти идентичен вызову cdecl: вызывающая программа очищает стек, а параметры передаются в порядке справа налево. Разница заключается в добавлении указателя this, который помещается в стек последним, как если бы он был первым параметром в прототипе функции.

  В компиляторе Microsoft Visual C++ этот указатель передается в ECX, и именно вызываемый объект очищает стек, отражая соглашение stdcall, используемое в C для этого компилятора и в функциях Windows API. Когда функции используют переменное количество аргументов, именно вызывающая сторона очищает стек (см. cdecl).
- Microsoft x64

  По умолчанию соглашение о вызовах x64 передает первые четыре аргумента функции в регистрах. Регистры, используемые для этих аргументов, зависят от расположения и типа аргумента. Оставшиеся аргументы передаются в стек в порядке справа налево.

  Целочисленные аргументы в первых четырех позициях передаются в порядке слева направо в RCX, RDX, R8 и R9, соответственно. Пятый и следующие аргументы передаются в стек, как описано выше. Все целочисленные аргументы в регистрах выравниваются по правому краю, поэтому вызываемый объект может игнорировать верхние биты регистра и получить доступ только к той части регистра, которая необходима.

24. Прерывания. Обработка прерываний в реальном режиме работы процессора

Прерывание – особая ситуация, когда выполнение текущей программы приостанавливается и управление передаётся программе-обработчику возникшего прерывания.

Виды прерываний:

- аппаратные (асинхронные) – события от внешних устройств;
- внутренние (синхронные) – события в самом процессоре, например, деление на ноль;
- программные – вызванные командой int

Маскирование прерываний

Внешние прерывания, в зависимости от возможности запрета, делятся на:

- маскируемые — прерывания, которые можно запрещать установкой соответствующего флага;
- немаскируемые (англ. Non-maskable interrupt, NMI) — обрабатываются всегда, независимо от запретов на другие прерывания

Таблица векторов прерываний в реальном режиме работы процессора

- Вектор прерывания — номер, который идентифицирует соответствующий обработчик прерываний. Векторы прерываний объединяются в таблицу векторов прерываний, содержащую адреса обработчиков прерываний.
- Располагается в самом начале памяти, начиная с адреса 0.
- Доступно 256 прерываний.
- Каждый вектор занимает 4 байта - полный адрес.
- Размер всей таблицы - 1 Кб.

Срабатывание прерывания

- Сохранение в текущий стек регистра флагов и полного адреса возврата (адреса следующей команды) - 6 байт
- Передача управления по адресу обработчика из таблицы векторов
- Настройка стека?
- Повторная входимость (реентерабельность), необходимость запрета прерываний ?

IRET - возврат из прерывания

- Используется для выхода из обработчика прерывания
- Восстанавливает FLAGS, CS:IP
- При необходимости выставить значение флага обработчик меняет его значение непосредственно в стеке

Перехват прерывания

- Сохранение адреса старого обработчика
- Изменение вектора на "свой" адрес
- Вызов старого обработчика до/после отработки своего кода
- При деактивации - восстановление адреса старого обработчика

Установка обработчика прерывания в DOS

int 21h

- AH=35h, AL = номер прерывания – возвращает в ES:BX адрес обработчика (в BX 0000:[AL*4], а в ES - 0000:[AL*4+2].
- AH=25h, AL = номер прерывания, DS:DX – адрес обработчика

Некоторые прерывания

- 0 - деление на 0
- 1 - прерывание отладчика, вызывается после каждой команды при флаге TF
- 3 - "отладочное", int 3 занимает 1 байт
- 4 - переполнение при команде INTO (команда проверки переполнения)
- 5 - при невыполнении условия в команде BOUND (команда контроля индексов массива)
- 6 - недопустимая (несуществующая) инструкция
- 7 - отсутствует FPU
- 8 - таймер
- 9 - клавиатура
- 10h - прерывание BIOS

Прерывание BIOS 10h	
AH = 00h	установка видеорежима, код в AL
AH = 02h	установить позицию курсора
AH = 08h	считать символ и атрибуты из позиции курсора
AH = 09h	записать символ и атрибуты в позицию курсора
AH = 0Ch	задать пиксель
AH = 0Dh	прочитать цвет пикселя

Резидентные программы

Резидентная программа - та, которая остаётся в памяти после возврата управления DOS

Завершение через функцию 31h прерывания 21h / прерывание 27h

DOS не является многозадачной операционной системой

Резиденты - частичная реализация многозадачности

Резидентная программа должна быть составлена так, чтобы минимизировать используемую память

Способы вызова старого обработчика прерывания

1. Вызов в начале командой CALL
2. Безусловная передача управления в конце командой JMP на переменную

3. Безусловная передача управления в конце командой JMP с сохранением адреса в машинном коде команды

Завершение с сохранением в памяти

- int 27h
    - DX = адрес первого байта за резидентным участком программы (смещение от PSP)
- int 21h, ah=31h
    - AL - код завершения
    - DX - объём памяти, оставляемой резидентной, в параграфах

Порты ввода-вывода

- Порты ввода-вывода - отдельное адресное пространство для взаимодействия программы, выполняемой процессором, с устройствами компьютера.
- IN - команда чтения данных из порта ввода
- OUT - команда записи в порт вывода
- Пример:

```
IN al, 61h
OR al, 3
OUT 61h, al
```

25. Процессор 80386. Режимы работы. Регистры.

32-разрядные процессоры (386+)

Производство x86: 1985 - ~2010 32-разрядные:

- Регистры, кроме сегментных
- Шина данных
- Шина адреса (232 = 4Гб ОЗУ)

Режимы работы

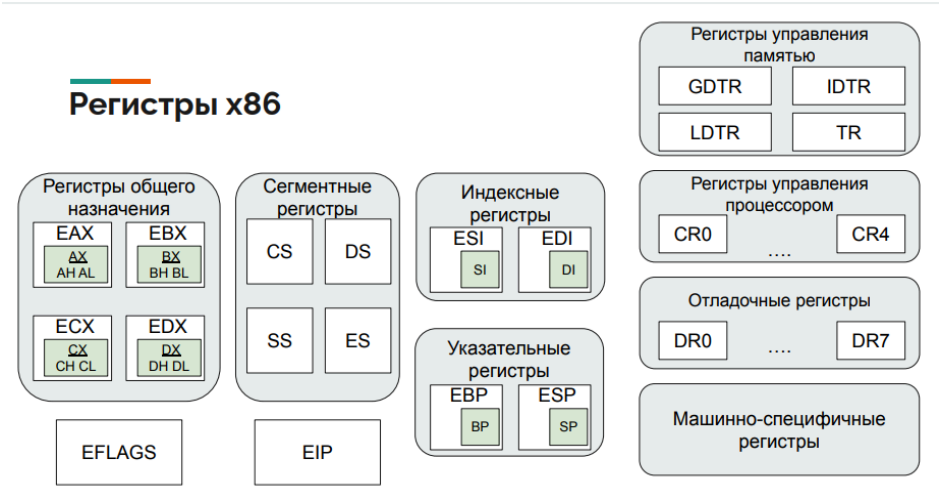
"Реальный" режим (режим совместимости с 8086)

- обращение к оперативной памяти происходит по реальным (действительным) адресам, трансляция адресов не используется;
- набор доступных операций не ограничен;
- защита памяти не используется.

"Защищённый" режим

- обращение к памяти происходит по виртуальным адресам с использованием механизмов защиты памяти;
- набор доступных операций определяется уровнем привилегий (кольца защиты): системный и пользовательский уровни

Режим V86



Система команд

- Аналогична системе команд 16-разрядных процессоров
- Доступны как прежние команды обработки 8- и 16-разрядных аргументов, так и 32-разрядных регистров и переменных
- Пример: mov eax, 12345678h xor ebx, ebx mov bx, 1 add eax, ebx ; eax=12345679h

26. Страничная модель памяти. Виртуальная память.

Модели памяти

- Плоская – код и данные используют одно и то же пространство
- Сегментная – сложение сегмента и смещения
- Страничная – виртуальные адреса отображаются на физические постранично
    - Виртуальная память — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (файл, или раздел подкачки)
    - Основной режим для большинства современных ОС
    - В x86 минимальный размер страницы - 4096 байт основывается на таблице страниц - структуре данных, используемой системой виртуальной памяти в операционной системе компьютера для хранения сопоставления между виртуальным адресом и физическим адресом. Виртуальные адреса используются выполняющимся процессом, в то время как физические адреса используются аппаратным обеспечением. Таблица страниц является ключевым компонентом преобразования виртуальных адресов, который необходим для доступа к данным в памяти.

Управление памятью в x86

- В сегментных регистрах - селекторы ○ 13-разрядный номер дескриптора ○ какую таблицу использовать - глобальную или локальную ○ уровень привилегий запроса 0-3
- По селектору определяется запись в одной из таблиц дескрипторов сегментов
- При включённом страничном режиме - по таблице страниц определяется физический адрес страницы либо выявляется, что она выгружена из памяти, срабатывает исключение и операционная система подгружает затребованную страницу из "подкачки" (swap)

27. Математический сопроцессор. Типы данных.

Сопроцессор (FPU – Floating Point Unit)

Изначально - отдельное опциональное устройство на материнской плате, с 80486DX встроен в процессор Операции над 7-ю типами данных :

- целое слово (16 бит)
- короткое целое (32 бита)
- длинное слово (64 бита)
- упакованное десятичное (80 бит)
- короткое вещественное (32 бита)
- длинное вещественное (64 бита)
- расширенное вещественное (80 бит)

Форма представления числа с плавающей запятой в FPU

Нормализованная форма представления числа (1,...*2exp)

Экспонента увеличена на константу для хранения в положительном виде Пример представления 0,625 в коротком вещественном типе:

- 1/2+1/8 = 0,101b
- 1,01b*2-1
- Бит 31 - знак мантиссы, 30-23 - экспонента, увеличенная на 127, 22-0 - мантисса без первой цифры
- 00111111001000000000000000000000

Все вычисления FPU - в расширенном 80-битном формате

Особые числа FPU

Положительная бесконечность: знаковый - 0, мантисса - нули, экспонента - единицы

Отрицательная бесконечность: знаковый - 1, мантисса - нули, экспонента - единицы NaN (Not a Number):

- qNAN (quiet) - при приведении типов/отдельных сравнениях
- sNAN (signal) - переполнение в большую/меньшую сторону, прочие ошибочные ситуации

Денормализованные числа (экспонента = 0): находятся ближе к нулю, чем наименьшее представимое нормальное число

28. Математический сопроцессор. Регистры.

Регистры FPU

- R0..R7, адресуются не по именам, а рассматриваются в качестве стека ST. ST соответствует регистру - текущей вершине стека, ST(1)..ST(7) - прочие регистры
- SR - регистр состояний, содержит слово состояния FPU. Сигнализирует о различных ошибках, переполнениях
- CR - регистр управления. Контроль округления, точности
- TW – 8 пар битов, описывающих состояния регистров: число, ноль, не-число, пусто
- FIP, FDP - адрес последней выполненной команды и её операнда для обработки исключений

  Для сопоставления флагов сопроцессора с флагами процессора нужно скопировать значения битов регистра SR в регистр FLAGS.

```
fstsw ax
sahf
```

Исключения FPU

- Неточный результат - произошло округление по правилам, заданным в CR. Бит в SR хранит направление округления
- Антипереполнение - переход в денормализованное число
- Переполнение - переход в "бесконечность" соответствующего знака
- Деление на ноль - переход в "бесконечность" соответствующего знака
- Денормализованный операнд
- Недействительная операция

29. Математический сопроцессор. Классификация команд.

Команды пересылки данных FPU

- FLD - загрузить вещественное число из источника (переменная или ST(n)) в стек. Номер вершины в SR увеличивается
- FST/FSTP - скопировать/считать число с вершины стека в приёмник
- FILD - преобразовать целое число из источника в вещественное и загрузить в стек
- FIST/FISTP - преобразовать вершину в целое и скопировать/считать в приёмник
- FBLD, FBSTP - загрузить/считать десятичное BCD-число
- FXCH - обменять местами два регистра (вершину и источник) стека

Базовая арифметика FPU

- FADD, FADDP, FIADD - сложение, сложение с выталкиванием из стека, сложение целых. Один из операндов - вершина стека
- FSUB, FSUBP, FISUB - вычитание
- FSUBR, FSUBRP, FISUBR - обратное вычитание (приёмника из источника)
- FMUL, FMULP, FIMUL - умножение
- FDIV, FDIVP, FIDIV - деление
- FDIVR, FDIVRP, FIDIVR - обратное деление (источника на приёмник)
- FPREM - найти частичный остаток от деления (делится ST(0) на ST(1)). Остаток ищется цепочкой вычитаний, до 64 раз
- FABS - взять модуль числа
- FCHS - изменить знак
- FRNDINT - округлить до целого
- FSCALE - масштабировать по степеням двойки (ST(0) умножается на 2ST(1))
- FXTRACT - извлечь мантиссу и экспоненту. ST(0) разделяется на мантиссу и экспоненту, мантисса дописывается на вершину стека
- FSQRT - вычисляет квадратный корень ST(0)

Команды сравнения FPU

- FCOM, FCOMP, FCOMPP - сравнить и вытолкнуть из стека
- FUCOM, FUCOMP, FUCOMPP - сравнить без учёта порядков и вытолкнуть
- FICOM, FICOMP, FICOMP - сравнить целые
- FCOMI, FCOMIP, FUCOMI, FUCOMIP (P6)
- FTST - сравнивает с нулём
- FXAM - выставляет флаги в соответствии с типом числа

Трансцендентные операции FPU

- FSIN
- FCOS
- FSINCOS
- FPTAN
- FPATAN
- F2XM1 – 2x -1
- FYL2X, FYL2XP1 – y*log2 x, y*log2 (x+1)

Константы FPU

- FLD1 – 1,0
- FLDZ - +0,0
- FLDPI - число Пи
- FLDL2E - log2 e
- FLDL2T - log2 10
- FLDLN2 – ln(2)
- FLDLG2 – lg(2)

Команды управления FPU

- FINCSTP, FDECSTP - увеличить/уменьшить указатель вершины стека
- FFREE - освободить регистр
- FINIT, FNINIT - инициализировать сопроцессор / инициализировать без ожидания (очистка данных, инициализация CR и SR по умолчанию)
- FCLEX, FNCLEX - обнулить флаги исключений / обнулить без ожидания
- FSTCW, FNSTCW - сохранить CR в переменную / сохранить без ожидания
- FLDCW - загрузить CR
- FSTENV, FNSTENV – сохранить вспомогательные регистры (14/28 байт) / сохранить без ожидания
- FLDENV - загрузить вспомогательные регистры
- FSAVE, FNSAVE, FXSAVE - сохранить состояние (94/108 байт) и инициализировать, аналогично FINIT
- FRSTOR, FXRSTOR - восстановить состояние FPU
- FSTSW, FNSTSW - сохранение CR
- WAIT, FWAIT - обработка исключений
- FNOP - отсутствие операции

Команда CPUID (с 80486)

Идентификация возможностей процессора

- Если EAX = 0, то в EAX - максимальное допустимое значение (1 или 2), а EBX:ECX:EDX – 12- байтный идентификатор производителя (ASCII-строка).
- Если EAX = 1, то в EAX - версия, в EDX - информация о расширениях
    - ○ EAX - модификация, модель, семейство

    - EDX: наличие FPU, поддержка V86, поддержка точек останова, CR4, PAE, APIC, быстрые системные вызовы, PGE, машинно-специфичный регистр, CMOVcc, MMX, FXSR (MMX2), SSE
- Если EAX = 2, то в EAX, EBX, ECX, EDX возвращается информация о кэшах и TLB

30. Расширения процессора. MMX. Регистры, поддерживаемые типы данных.

В основу положен принцип SIMD (Single Instruction Multiple Data) – одна инструкция - множество данных.

Увеличение эффективности обработки больших потоков данных (изображения, звук, видео) – выполнение простых операций над массивами однотипных чисел.

Расширение MMX включает в себя восемь 64-битных регистров общего пользования MM0—MM7

Регистры объединены с мантиссами регистров FPU. При записи в MMX экспонента и знаковый бит заполняются единицами.

> Пользоваться одновременно и FPU, и MMX не получится, требуется *FSAVE* и *FRSTOR*

Типы данных MMX:

> Команды MMX обрабатывают целочисленные данные, упакованные в группы (векторы) общей длиной 64 бита, либо одиночные 64-битные слова. Такие данные могут находиться в памяти или в восьми MMX-регистрах.
>
> - учетверённое слово – quadword (64 бита);
> - упакованные двойные слова – packed doubleword (2);
> - упакованные слова – packed word (4);
> - упакованные байты – packed byte(8).

Почему регистры *FPU* и *MMX* объединены? Дело в том, что в многозадачной среде программы должны быть независимы друг от друга. При переключении с одной задачи на другую значения всех регистров *«старой»* задачи должны быть сохранены. В настоящее время о сохранении регистров микропроцессора заботится сам микропроцессор, а о сохранении регистров сопроцессора заботится операционная система. Так как все регистры *MMX* одновременно регистры *FPU*, то операционная система, сохраняя регистры *FPU*, сохраняет и регистры MMX.

31. Расширения процессора. MMX. Классификация команд.

Команды *MMX* перемещают упакованные данные в память или обычные регистры целиком, но арифметические и логические операции выполняют поэлементно.

Большинство команд имеет суффикс, который определяет тип данных и используемую арифметику:

- *US* (unsigned saturation) – арифметика с насыщением, данные без знака.
- *S* или *SS* (signed saturation) – арифметика с насыщением, данные со знаком. Если в суффиксе нет ни S, ни SS, используется циклическая арифметика (wraparound).

B, W, D, Q указывают тип данных. Если в суффиксе есть две из этих букв, первая соответствует входному операнду, а вторая — выходному.

- Команды пересылки
  - *MOVD*, *MOVQ* – пересылка двойных/учетверённых слов
  - *PACKSSWB*, *PACKSSDW* – упаковка со знаковым *насыщением* слов в байты/двойных слов в слова. Приёмник – младшая половина приёмника, источник – старшая половина приёмника
  - *PACKUSWB* – упаковка слов в байты с беззнаковым насыщением
  - *PUNPCKH**BW***, *PUNPCKH**WD***, *PUNPCKH**DQ*** – распаковка и объединение старших элементов источника и приёмника через одного

  Насыщение – замена переполнения/антипереполнения превращением в максимальное/минимальное значение.
- Арифметические операции
  - *PADDB*, *PADDW*, *PADDD, PADDQ* – поэлементное сложение, перенос игнорируется
  - *PADDSB*, *PADDSW* – сложение с насыщением
  - *PADDUSB*, *PADDUSW* – беззнаковое сложение с насыщением
  - *PSUBB*, *PSUBW*, *PDUBD* – вычитание, заём игнорируется
  - *PSUBSB*, *PSUBSW* – вычитание с насыщением
  - *PSUBUSB*, *PSUBUSW* – беззнаковое вычитание с насыщением
  - *PMILHW*, *PMULLW* – старшее/младшее умножение (сохраняет старшую или младшую части результата в приёмник)
  - *PMADDWD* – умножение и сложение. Перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших
- Команды сравнения
  - *PCMPEQB*, *PCMPEQW*, *PCMPEQD* - проверка на равенство. Если пара равна – соответствующий элемент приёмника заполняется единицами, иначе - нулями

    - *PCMPGTB*, *PCMPGTW*, *PCMPGTD* – сравнение. Если элемент приёмника больше, то заполняется единицами, иначе - нулями
- Сдвиговые операции
    - *PSLLW, PSLLD, PSLLQ* – логический влево
    - *PSRLW, PSRLD, PSRLQ* – логический вправо
    - *PSRAW, PSRAD* – арифметический вправо
- Логические операции
    - *PAND* – логическое И
    - *PANDN* – логическое НЕ-И
    - *POR* – логическое ИЛИ
    - *PXOR* – исключающее ИЛИ

32. Расширения процессора. SSE. Регистры, поддерживаемые типы данных.

Технология SSE позволяла преодолеть две основные проблемы MMX: при использовании MMX невозможно было одновременно использовать инструкции сопроцессора, так как его регистры были общими с регистрами MMX, и возможность MMX работать только с целыми числами.

SSE включает восемь 128-битных регистров xmm0 — xmm7, свой регистр флагов и набор инструкций, работающих со скалярными и упакованными типами данных.

SSE позволял работать только с одним типом данных:

- Четыре 32-битные значения с плавающей запятой одинарной точности.
- Целочисленные команды работают с регистрами MMX

SSE2 позволил окончательно заменить MMX:

- Четыре 64-битных значения с плавающей запятой двойной точности.
- Два 64-битные целые числа
- Четыре 32-битные целые числа
- Восемь 16-битные целые числа
- Шестнадцать 8-битные целые числа

33. Расширения процессора. SSE. Классификация команд.

Streaming SIMD Extensions

SSE включает в себя

- Пересылки и упаковки
  - *MOVSS*, *MOVSD* – пересылка значения с плавающей запятой одинарной/двойной точности.
  - MOVNTPD, MOVNTPQ, MOVNTPS – пересылка значения минуя кэш
  - Команды упаковки аналогичные MMX
- Арифметические
- Сравнения
- Преобразования типов
- Логические
- Целочисленные
- Управления состоянием
- Управления кэшированием
- Горизонтальная работа с регистрами (сложение и вычитание значений в одном регистре) (SSE3)
- ускорение видеокодеков (SSE4)
- вычисление CRC32 (SSE4)
- обработка строк (SSE4)

34. Расширения процессора. AVX. Регистры, поддерживаемые типы данных

Advanced Vector Extensions

Как и SSE позволяет работать со скалярными значения (отдельные числа) и векторами (наборы чисел). Скалярные значения представляют отдельные целые числа или числа с плавающей точкой одинарной или двойной точности. Векторы содержат несколько значений с плавающей точкой или целых чисел.

Позволяет обрабатывать по восемь 32-битных чисел с плавающей точкой одинарной точности или по четыре 64-битных числа с плавающей точкой двойной точности.

AVX/AVX2:

- Ширина векторных регистров SIMD увеличивается с 128 (XMM) до 256 бит (регистры YMM0 — YMM15). Существующие 128-битные SSE-инструкции будут использовать младшую половину новых YMM-регистров, не изменяя старшую часть. Для работы с YMM-регистрами добавлены новые 256-битные AVX-инструкции.
- SSE-инструкции используют младшую половину YMM-регистров, не меняя старшую часть;

AVX3:

- AVX3 (2013 г.) - 512-битное расширение (регистры ZMM0 – ZMM31).

35. Расширения процессора. AVX. Классификация команд.

В AVX введены «неразрушающие» (трёхоперандные) инструкции: $c = a + b$ вместо $a = a + b$, при этом регистр a остаётся неизменённым.

В AVX2 ввели FMA – Fused Multiply-Add (совмещённое умножение-сложение $a \leftarrow a + (b \cdot c)$ для вещественных чисел). Данная инструкция позволяет более эффективно реализовать операции деления и извлечения квадратного корня (при отсутствии аппаратной реализации), умножение векторов и матриц, вычисление полиномов по схеме Горнера.

Набор инструкций AVX содержит в себе аналоги 128-битных SSE-инструкций для вещественных чисел. При этом, в отличие от оригиналов, сохранение 128-битного результата будет обнулять старшую половину YMM-регистра

SSE-инструкции используют младшую половину YMM-регистров, не меняя старшую часть.

36. Процессоры семейства x86-64. Регистры, режимы работы.

64-битная версия (изначально — расширение) архитектуры x86, позволяющая выполнять программы в 64-разрядном режиме.

Регистры:

- целочисленные 64-битных регистры общего назначения - RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP;
- новые целочисленные 64-битных регистры общего назначения R8 — R15
- 64-битный указатель RIP и 64-битный регистр флагов RFLAGS.

Режимы работы:

- Legacy mode - совместимость с 32-разрядными процессорами: «Унаследованный» режим позволяет процессору x86-64 выполнять команды, предназначенные для процессоров x86, обеспечивая таким образом полную совместимость с 32-битным кодом и операционными системами для x86. В этом режиме процессор ведёт себя точно так же, как x86-процессор. Функции и возможности, предоставляемые архитектурой x86-64 (например, 64-битные регистры), в этом режиме, естественно, недоступны. В этом режиме 64-битные программы и операционные системы работать не будут.
- Long mode – 64-разрядный режим с частичной поддержкой 32-разрядных программ. Рудименты V86 и сегментной модели памяти упразднены.

37. Расширения процессора. AES. Назначение, классификация команд.

Intel Advanced Encryption Standard New Instructions

Intel добавили поддержку на аппаратном уровне симметричного алгоритма шифрование AES — одного из самых популярных алгоритмов блочного шифрования.

Цель – ускорение приложений, использующих шифрование по алгоритму AES

Набор инструкций AES включает инструкции для расширения ключа, шифрования и дешифрования с использованием ключей разного размера (128 бит, 192 бит и 256 бит).

Так как AES реализуется как набор инструкций, а не как программное обеспечение, что повышает безопасность. Само использовании инструкций AES дает прирост к скорости шифрования по сравнению с использованием регистров SSE, AVX и пр.

Команды:

- раунда шифрования и расшифровывания;
    - Выполнить один раунд
    - Выполнить последний раунд
- способствования генерации раундового ключа

Примеры программного обеспечения, поддерживающих расширение команд AES:

- OpenSSL
- The Bat! 4.3
- TrueCrypt 7.0
- 7-ZIP для поддержки шифрования алгоритмом AES-256 для форматов 7z и ZIP.

38. Архитектура RISC. Семейство процессоров ARM. Версии архитектуры, профили.

Ранние архитектуры процессоров (комплексные, CISC (Complex instruction set computer)):

- большее количество команд, длина которых произвольна
- разные способы адресации для упрощения написания программ на ассемблере;
- небольшое число регистров, каждый из которых выполняет строго определённую функцию.
- арифметические действия кодируются в одной команде;
- поддержка конструкций языков высокого уровня.

Недостатки: на практике многие возможности CISC используются компиляторами ЯВУ ограничено, а их поддержка затратна. Основной недостаток CISC — более сложный подход к распараллеливанию вычислений.

RISC (reduced instruction set computer):

- сведение набора команд к простым типовым;
- большее количество регистров;
- стандартизация формата команд, упрощение конвейеризации.

Поскольку многие реальные программы тратят большую часть своего времени на выполнение простых операций, идея RISC сделать эти операции максимально быстрыми. Быстродействие увеличивается за счёт такого кодирования инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим. Такой подход облегчает повышение тактовой частоты и делает более эффективным распараллеливание.

Процессоры ARM занимают свыше 90% рынка процессоров для мобильных устройств. Первая версия – ARMv1 появилась в 1985.

Современные версии архитектуры - ARMv7 (32-разрядная), ARMv8 (64-разрядная), ARMv9 (поддержка SVE2), ARMv11 и Cortex

Разделяют три профиля процессоров ARM (не считая Classic, не реализуется, был основным профилем до архитектуры v6):

- A (application) – для устройств, требующих высокой производительности (смартфоны, планшеты), поддерживает виртуальную память с помощью блоков управления памятью;
- R (real time) – для приложений, работающих в реальном времени;
- M (microcontroller) – для микроконтроллеров и недорогих встраиваемых устройств. Отсутствуют блоки управления памятью;

Профили могут поддерживать меньшее количество команд (команды определенного типа).

Процессоры ARM имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и преобладают на рынке мобильных устройств, для которых данный фактор критически важен.

39. Процессоры ARM. Наборы команд. Основные регистры.

Наборы команд ARM

- A32 (32-разрядные)
- Thumb (16-разрядные, более компактные)
- Thumb2 (16- и 32-разрядные)
- A64 (32-разрядные)

Регистры общего назначения (в ARMv8):

- R0-R29 (Xnn – 64-разрядный алиас, Wnn - 32-разрядный алиас младшей половины)
- SP
- LR (R30) регистр связи – используется в командах прыжка(ветвления) и при выполнении прерываний.
- PC (счётчик команд) – используется для хранения адресов выполняющихся команд.
- CPSR – регистр флагов

Процессор может находиться в одном из следующих операционных режимов:

- User mode — обычный режим выполнения программ. В этом режиме выполняется большинство программ.
- Fast Interrupt (FIQ) — режим быстрого прерывания (меньшее время срабатывания).
- Interrupt (IRQ) — основной режим прерывания.
- System mode — защищённый режим для использования операционной системой.
- Abort mode — режим, в который процессор переходит при возникновении ошибки доступа к памяти (доступ к данным или к инструкции на этапе prefetch конвейера).
- Supervisor mode — привилегированный пользовательский режим.
- Undefined mode — режим, в который процессор входит при попытке выполнить неизвестную ему инструкцию.

Переключение режима процессора происходит при возникновении соответствующего исключения или же модификацией регистра статуса.

Расширения:

- VFP v1-v5
- SIMD, Advanced SIMD (NEON), SVE, SVE2
- AES, SHA

40. Процессоры ARM. Основные команды.

Базовые команды ARM:

- Команды пересылки данных: LDR, STR, MOV
- Арифметические команды: ADD, SUB, MUL (Команды деления отсутствуют. Замена для деления на константу - умножение на заранее вычисленную степень 2, затем сдвиг)
- Побитовые операции: ANR, ORR, XOR, LSLS, LSRS, ASR, ROR, RRX…
- Команда сравнения CMP

Команды ветвления B, BL, BLX, Bnn:

- B (Branch) - переход
- BL (Branch with link) - переход с сохранением адреса возврата в LR
- BLX - переход с переключением системы команд
- BEQ, BNE, BLT, BLE, BGT, BGE…

Вызов программного прерывания:

- SWI immed_8 (0..255)
  Переводит процессор в Supervisor mode, CPSR сохраняется в Supervisor Mode SPSR, управление передаётся обработчику прерывания по вектору.
  Новое название – SVC.

Условное исполнение:

- Возможно добавлять к командам суффикс условия
  SUBGT
- Возможно соединять сдвиги и вращения в инструкции
  a += (j << 2) может быть преобразовано в команду из одного слова и одного цикла в ARM:
  ADD Ra, Ra, Rj, LSL #2

Допускаются команды push lr, pop pc.

Не мешает рассмотреть (в вопросах нет)

1. **EXE-файлы в Windows**

2. ELF (Executable and Linking format)

3. Дизассемблирование. Реверс-инжиниринг

4. Архитектура VLIW. Эльбрус-8С

5. Java. Java virtual machine (JVM)

6. Платформа .NET. CLR, CIL

7. WebAssembly (wasm)