Вопросы

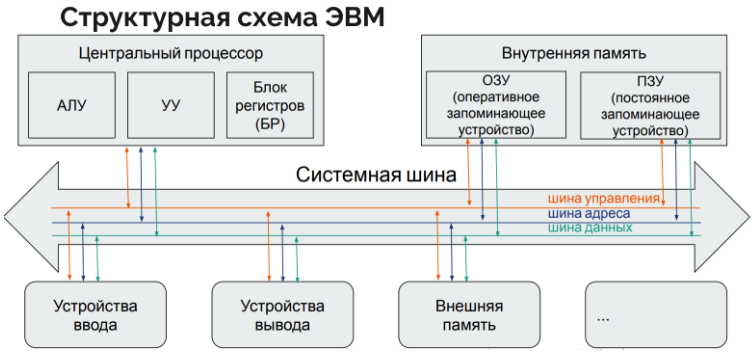
- 8086 – реальный режим. В этой архитектуре 20-разрядная адресация (разрядность шины адреса) (то есть доступно 2^20байт = 1Мб памяти, а не 2^16. Шина данных – 16 разрядная)
- Физический адрес = адрес начала сегмента + смещение
- Сегментный регистр содержит номер параграфа начала сегмента (параграф=16 байт)
- В реальном режиме работы всё адресное пространство делится на сегменты. Под сегментом понимается блок смежных ячеек с максимальным размером 64 Кбайт (2^16 байт) и начальным или базовым адресом, находящимся на 16-байтной границе (такая граница называется параграфом). Таким образом, сегменты могут частично перекрывать друг друга.
- в x86 минимальный размер страницы - 4096 байт;
- Размер всей таблицы обработчиков прерываний - 1 Кб. (256 прерываний на 4 байта в каждом) Располагается в самом начале памяти, начиная с адреса 0.

80386

32-разрядные:

- регистры, кроме сегментных
- шина данных
- шина адреса (2^32 = 4ГБ ОЗУ, до этого 2^20байт = 1Мб памяти)

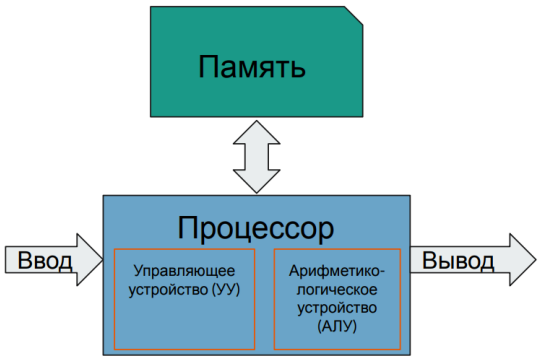
Из чего состоит системная шина



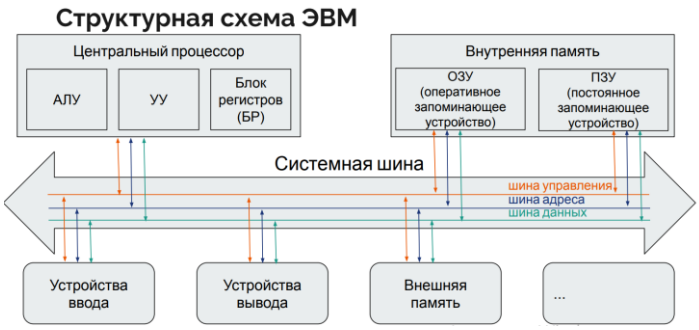
Центральный элемент – системная шина – некоторая абстракция, которая осуществляет взаимодействие между собой всех устройств. **Делится на: шина управления, шина адреса и шина данных.** К ней все подключаются – центральной процессор, внутренняя память (оперативная в первую очередь). Если процессору необходимо считать какие-то данные, то он выставляет на шину адреса номер ячейки памяти (адрес) и по шине управления отправляет сигнал в память – считай значение. Программа останавливает свою работу и ждет ответа. Память находит ячейку памяти по адресу, считывает значение и устанавливает на шину данных и отправляет по шине управления сигнал – готово, значение установлено. ЦП получает данные и программа продолжается. Если же нужно записать, то по аналогии. Адрес, данные, сигнал. Приостановление. Выполнение, сигнал.

На практике системная шина – материнская плата. + обычно несколько шин – для оперативки, для другой памяти.

насколько современный комп соответствует архитектуре фон Неймана?



Тут все абстрактно, а вот структурная схема – ближе к реальности



Добавляется системная шина, разделяется память, добавляется блок регистров. Еще не хватает кэш-памяти

Ближе к реальности, отражает схему современного компьютера.

Центральный элемент **– системная шина** – некоторая абстракция, которая осуществляет взаимодействие между собой всех устройств. Делится на: шина управления, шина адреса и шина данных**.** К ней все подключаются – центральной процессор, внутренняя память (оперативная в первую очередь). Если процессору необходимо считать какие-то данные, то он выставляет на шину адреса номер ячейки памяти (адрес) и по шине управления отправляет сигнал в память – считай значение. Программа останавливает свою работу и ждет ответа. Память находит ячейку памяти по адресу, считывает значение и устанавливает на шину данных и отправляет по шине управления сигнал – готово, значение установлено. ЦП получает данные и программа продолжается. Если же нужно записать, то по аналогии. Адрес, данные, сигнал. Приостановление. Выполнение, сигнал.

В ЦП добавляется Блок регистров – внутренние ячейки памяти процессор, их всего несколько десятков.

Также процессор взаимодействует с оперативкой – единственный участок памяти, к которым процессор имеет доступ, туда загружается код программы. Доступ к дисками, флешкам – это как внешняя память.

На практике системная шина – материнская плата. + обычно несколько шин – для оперативки, для другой памяти.

Ответ, который был: почти да, но в современной оперативной памяти есть некоторые поля только для данных, и только для команд

Почему нельзя использовать в командах пересылки данных 2 операнда типа память

(ответ: системная шина состоит из 3 шин: шина управления, шина данных, шина адреса, на шине управления выставляется команда получения/записи данных, на шину адреса - адрес и по шине данных они пересылаются, а одновременно послать два сигнала по ним невозможно)

У меня спросил, чтобы подробно рассказать про ОЗУ и ПЗУ. что такое ПЗУ, и в чём вообще смысл,

a. ОЗУ (оперативное запоминающее устройство) – энергозависимая, быстрая, небольшая по объему память для чтения и записи информации. Процессор имеет к нему прямой доступ. В неё загружают данные и программы, с её помощью работает компьютер в целом. Очищается при отключении питания.
(Random Access Memory, RAM — память с произвольным доступом) — *энергозависимая часть системы компьютерной памяти, в которой во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные и промежуточные данные, обрабатываемые процессором. ОЗУ может изготавливаться как отдельный внешний модуль или располагаться на одном кристалле с процессором*

   ПЗУ (постоянное запоминающее устройство) – энергонезависимая, только для чтения. В ПЗУ хранится информация, которая записывается туда при изготовлении ЭВМ. Важнейшая микросхема ПЗУ - BIOS.
   ROM (read-only memory, постоянное запоминающее устройство)
   Нужна для запуска компьютера, так как оперативная память очищается. В ПЗУ хранится стартовая программа загрузка компьютера.

ответ - там хранятся системные штуки (BIOS) которые используются для загрузки операционной системы

Про регистры флагов, какими командами их можно установить потом по классификации спрашивал, зачем нужны флаги состояний и управления

Флаги делятся на группы:

1. Флаги состояний (CF, AF, PF, ZF, SF, OF)
2. Системные флаги (TF, IF, IOPL, NT)
3. Флаг направления (DF).

CF, TF, IF, DF - можно изменять командами. Остальные флаги напрямую нельзя изменить

- STC/CLC/CMC - установить/сбросить/инвертировать CF
- STD/CLD - установить/сбросить DF
- AHF - загрузка флагов состояния в AH
- SAHF - установка флагов состояния из AH
- CLI/STI - запрет/разрешение прерываний (сброс/установка IF)
- Загрузка и выгрузка в стеке

Флаги состояния – отслеживать корректность (арифметических операций) выставляются при cmp, test и далее используются (командами условного перехода (Jcc), условной пересылки данных (CMOVcc) или условной установки байтов (SETcc),), направления – обработка строк

Системные флаги и поле IOPL управляют операционной средой и не предназначены для использования в прикладных программах.

еще про флаг DF и IF, и еще зачем нам делать IF = 0, то есть зачем выключать обработку прерываний

зеленый DF (direction flag) - флаг направления
Он контролирует поведение команд обработки строк: когда он установлен в 1, строки обрабатываются в сторону уменьшения адресов (справа-налево), когда DF = 0 – наоборот (слева-направо).
IF (interrupt enable flag) - флаг разрешения прерываний
если 0 процессор перестает обрабатывать прерывания от внешних устройств. Когда **выполняются критические блоки кода**, но надолго нельзя снимать

Помню, когда обработчик прерываний изменяли, нужно было выключить обработку, чтобы случайно не выполнилось прерывание, пока ты его перенастраиваешь, если можн так сказать

cmp, как называются все эти флаги вместе

состояния

надо бы ответить что знак/беззнак одинаково сравниваются, а условия уже программист выбирает в зависимости от того с чем работает знак/беззнак

и спросит про другие команды, изменяющие флаги и используемые для сравнения - test, dec и как они это делают

TEST <приемник>, <источник> - логическое сравнение.

- *Аналог AND, но результат не сохраняется*
- *Выставляются флаги SF, ZF, PF. OF и CF обнуляются, значение AF не определено*

INC <приемник> DEC <приемник>

Увеличивает/уменьшает приёмник (регистр/переменная) на 1.

В отличие от ADD, не изменяет CF.

Арифметические OF, SF, ZF, AF, PF устанавливаются в соответствии с результатом.

CMPS<приемник>, <источник>/CMPSB/CMPSW –

сравнение строк и установка флагов аналогично CMP

xlat где можно использовать и что там за доп параметр

- Ответ, который был: адрес, и он сказал ок, только там сегментный регистр указывается

- В качестве аргумента для XLAT в ассемблере можно указать имя таблицы, но эта информация никак не используется процессором и служит только в качестве комментария.
- Если в адресе явно указан сегментный регистр, он будет использоваться вместо DS.

  Например, можно написать следующий вариант преобразования шестнадцатеричного числа в ASCII-код соответствующего ему символа:

  ```
  mov al, 0Ch
  mov bx, offset htable
  xlatb
  ```

  если в сегменте данных, на который указывает регистр ES, было записано

  ```
  htable db "0123456789ABCDEF"
  ```

  то теперь AL содержит не число 0Ch, а ASCII-код буквы C.
-
- *Применение: XLAT применяется для перекодировки значений. Команду XLAT хорошо использовать при кодировании и декодировании текстовых данных. С помощью этой команды программа может организовать простую замену кодов символов.*

Почему команды обработки строк так называются, если они работают с байтами/словами?

(ответ: потому что, используя префиксы rep/repne и тд, с их помощью можно проанализировать/обработать строку длины cx)

Что, помимо действия над байтами/словами делают команды?

(ответ: изменяют di и si. Надо, наверное, было про df добавить, но он итак 5 поставил) еще флаги выставляются

lea, какие математические операции можно ей заменять

LEA <приемник>, <источник> - вычисление эффективного адреса (значение смещения) Вычисляет эффективный адрес источника (переменная) и помещает его в приёмник (регистр).

- Позволяет вычислить адрес, описанный сложным методом адресации, например, по базе с индексированием
- Иногда используется для быстрых арифметических вычислений: lea bx, [bx+bx*4] lea bx, [ax+12]
- Эти вычисления занимают меньше памяти, чем соответствующие MOV и ADD, и не изменяют флаги.

По сути, является аналогом операции mov ПРИЕМНИК, offset ИСТОЧНИК

Если адрес - 32-битный, а регистр-приемник - 16-битный, старшая половина вычисленного адреса теряется, если наоборот, приемник - 32-битный, а адресация - 16-битная, то вычисленное смещение дополняется нулями.

Команду LEA часто используют для быстрых арифметических вычислений, например умножения:

```
lea     bx,[ebx+ebx*4] ; BX = EBX x 5
```

или сложения:

```
lea     ebx,[eax+12]    ;EBX = EAX + 12
```

(эти команды меньше, чем соответствующие MOV и ADD, и не изменяют флаги)

Вместо команд инкремента и декремента можно использовать команду LEA. Сложение и вычитание с константой можно заменить командой LEA. Умножение и деление на константу можно заменить командой LEA или сочетанием команд сдвига и команд сложения и вычитания.

Про FPU типы данных спрашивал, каким образом хранятся вещественные числа

Типы данных.

Операции над 7-ю типами данных

1. целое слово (16 бит)
2. короткое целое (32 бита)
3. длинное слово (64 бита)
4. упакованное десятичное (80 бит)
   Packed BCD (Binary-coded decimal). Каждые 4 бита такого числа могут принимать значения от 0 до 9, бит #79 - знак, 78-72 не имеют значения. При выгрузке NaN'ов, получается число, 18 старших битов которого установлены, а остальные - сброшены.
   Пример: десятичное число -12345 в формате Packed BCD равно 80 00 00 00 00 00 00 01 23 45h. В памяти байты расположены наоборот.

5. короткое вещественное (32 бита)
6. длинное вещественное (64 бита)
7. 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
- Бит 31 - знак мантиссы, 30-23 - экспонента, увеличенная на 127, 22-0 - мантисса без первой цифры
- Пример представления 0,625 в коротком вещественном типе:
  1/2+1/8 = 0,101b
  1,01b*2-1
  001111110**01000000000000000000000**
- Все вычисления FPU - в расширенном 80-битном формат

по сегментам и модулям спросил какого размера максимум может быть программа.

- 8086 – реальный режим. В этой архитектуре 20-разрядная адресация (разрядность шины адреса) (то есть доступно 2^20байт = 1Мб памяти, а не 2^16. Шина данных – 16 разрядная) Физический адрес = адрес начала сегмента + смещение
- 80386 шина адреса (2^32 = 4ГБ ОЗУ, до этого 2^20байт = 1Мб памяти

сколько максимум и минимум сегментов кода может быть в проге. какой максимальный размер сегмента

- ассемблер позволяет изменять устройство программы как угодно - помещать данные в сегмент кода, разносить код на множество сегментов, помещать стек в один сегмент с данными или вообще использовать один сегмент для всего.

если мы в неинциализированный сегмент пишем что будет?

какие регистры общего назначения в 32, 64 - разрядных процессорах. В чем их отличия от архитектуры 8086

- 8086 – 16 бит DX, DL

- С 386+ процессоры 32 разрядные, 32-разрядные регистры, кроме сегментных (EDX = Extended DX (обращение к частям остается (DX, DL))
- 64-разрядные процессоры (x86-64)
  Режимы работы:
  - Legacy mode - совместимость с 32-разрядными процессорами
  - Long mode – 64-разрядный режим с частичной поддержкой 32-разрядных программ. Рудименты V86 и сегментной модели памяти упразднены

  Регистры:

  - целочисленные 64-битных регистры общего назначения - RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP;
  - новые целочисленные 64-битных регистры общего назначения R8 — R15

По стеку я ему начала рассказывать, он спросил, а как у нас подпрограммы, сами по себе включаются?

Нет, команда "call". Она сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Это про какой переход вы говорите?
- Про ближний
- А при дальнем? Какое там отличие?
- процессор помещает в стек значения регистров CS и EIP и осуществляет дальний переход аналогично команде JMP.

CALL <имя> - вызов процедуры,

1. Сохраняет адрес следующей (текущей по учебнику) команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
2. Передаёт управление на значение аргумента.
   Операндом может быть непосредственное значение адреса (метка в ассемблерных программах), регистр или переменная, содержащие адрес перехода.
   Если операнд CALL - регистр или переменная, то его значение рассматривается как абсолютное смещение, если операнд - ближняя метка в программе, то ассемблер указывает ее относительное смещение.

   При вызове подпрограммы параметры в большинстве случаев помещают в стек, а в EBP записывают текущее значение ESP. Если подпрограмма использует стек для хранения локальных переменных, ESP изменится, но EBP можно будет использовать для того, чтобы считывать Значения параметров напрямую из стека