

**Вычислить физический адрес, соответствующий следующим сегментной части и смещению:
1A29h:37B4h**

Правильный ответ: 1DA44h

**Вычислить физический адрес, соответствующий следующим сегментной части и смещению:
5089h:37B4h**

Правильный ответ: 54044h

Какова разрядность регистра IP?

Правильный ответ: 16

Какова разрядность регистра DS?

Правильный ответ: 16

Какая формулировка соответствует характеристикам короткого перехода?

Правильный ответ: Метка занимает 1 байт, переход допустим в диапазоне от -128 до 127 байт от текущего значения IP

Какая формулировка соответствует характеристикам ближнего перехода?

Правильный ответ: Метка занимает 2 байта, переход допустим в пределах текущего сегмента

Какая формулировка соответствует характеристикам дальнего перехода?

Правильный ответ: Метка занимает 4 байта, переход допустим в пределах всей памяти

Какая команда выполняет переход, соответствующий условию "больше"?

Правильный ответ: JG

Какая команда выполняет переход, соответствующий условию "меньше или равно"?

Правильный ответ: JLE

Имеется описание следующего сегмента:

DS SEGMENT

ORG 100h

I DW 0

A DB 1

DS ENDS

Чему равно OFFSET A?

Правильный ответ: 102h

Имеется описание следующего сегмента:

DS SEGMENT

ORG 10h

I DW 0

A DB 1

B DB FFh

DS ENDS

Чему равно OFFSET A?

Правильный ответ: 12h

В каких командах языка ассемблера применяется сегментный префикс?

Правильный ответ: При работе со значениями переменных

Какая операция с сегментным регистром недопустима?

Правильный ответ: Загрузка константы

mov ax, [bx][si]+10

Правильный ответ: $bx+si+10$

mov ax, [bp][di]+4

Правильный ответ: $bp+di+4$

Что такое неупакованное двоично-десятичное число?

Правильный ответ: Десятичная цифра, хранящаяся в байте

Что такое упакованное двоично-десятичное число?

Правильный ответ: Две десятичные цифры, хранящиеся в полубайтах одного байта

Какая команда осуществляет циклический сдвиг битов вправо?

Правильный ответ: ROR

Какая команда осуществляет логический сдвиг битов влево?

Правильный ответ: SHL

Какая команда осуществляет циклический сдвиг битов влево без использования дополнительных разрядов?

Правильный ответ: ROL

Что делает команда TEST?

Правильный ответ: Побитовое И без сохранения результата

Укажите, на какую величину уменьшится указатель вершины стека после выполнения следующих команд:

PUSH AX

PUSH BX

POP CX

CALL F ; подпрограмма объявлена в том же сегменте

Правильный ответ: 4

Укажите, на какую величину изменится указатель вершины стека после выполнения следующей команды:

RETN 4

Правильный ответ: 6

Укажите, какая характеристика соответствует команде NEG

Правильный ответ: изменение знака

Какого вида прерываний не существует?

Правильный ответ: Системные

Где находится таблица векторов прерываний в реальном режиме работы процессора x86, каков её размер и каков размер одного вектора?

Правильный ответ: Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 4 байта

Значения каких регистров помещаются в стек при срабатывании прерывания?

Правильный ответ: FLAGS, CS, IP

Что программа обязательно должна сделать для корректного перехвата прерывания?

Правильный ответ: Сохранить адрес старого обработчика прерывания

Что такое порты ввода-вывода в x86?

Правильный ответ: Независимое адресное пространство для взаимодействия с устройствами

Какое число соответствует 16-разрядному дополнительному коду 111111111111110?

Решение: Переводим в прямой код: 0000 0000 0000 0001. Добавляем 1 (так как изначально число было с первым битом 1), получаем 0000 0000 0000 0010, что соответствует 2. Так как изначально число отрицательное, то правильный ответ -2.

Вопрос 1

Неверно

Баллов: 0,00 из 1,00

Отметить вопрос

Какое число соответствует 16-разрядному дополнительному коду 111111111111110?

Выберите один ответ:

☒ a. -1

☐ b. -32767

☐ c. -2

Ваш ответ неправильный.

Правильный ответ: -2

Какая команда осуществляет циклический сдвиг битов вправо без использования дополнительных разрядов?

Ответ: ROR

Вопрос 2

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Какая команда осуществляет циклический сдвиг битов вправо без использования дополнительных разрядов?

Выберите один ответ:

☐ a. SHR

☐ b. SAR

☒ c. ROR

Ваш ответ верный.

Правильный ответ: ROR

Какой регистр используется командой lodsw?

Ответ: AX

Вопрос 4
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Какой регистр используется командой lodsw?

Выберите один ответ:

- ☐ a. BP
☐ b. CX
☒ c. AX

Ваш ответ верный.

Правильный ответ: AX

Какое описание соответствует флагу IF?

Ответ: Это флаг прерываний. Его сброс приводит к прекращению обработки процессором прерываний от внешних устройств.

Вопрос 5
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Какое описание соответствует флагу IF?

Выберите один ответ:

- ☐ a. Этот флаг контролирует поведение команд обработки строк и определяет направление обработки: в сторону увеличения или уменьшения адресов
☒ b. Это флаг прерываний. Его сброс приводит к прекращению обработки процессором прерываний от внешних устройств.
☐ c. Это флаг переполнения для чисел со знаком. Он устанавливается, если результат предыдущей операции над числами со знаком выходит за допустимые для них пределы.

Ваш ответ верный.

Правильный ответ: Это флаг прерываний. Его сброс приводит к прекращению обработки процессором прерываний от внешних устройств.

Укажите, на какую величину уменьшится указатель вершины стека после выполнения следующих команд?

POP CX, PUSH AX, PUSH BX, CALL F (дальний вызов)

Решение: PUSH AX уменьшит указатель на 2, также как и BX - так как они занимают два байта. POP CX увеличит на 2 байта, а CALL F - дальний переход, поэтому метка займёт четыре байта.

Итого: $2 + 2 - 2 + 4 = 6$ - ответ.

Вопрос 9
Неверно
Баллов: 0,00 из 1,00
Отметить вопрос

Укажите, на какую величину уменьшится указатель вершины стека после выполнения следующих команд:

POP CX

PUSH AX

PUSH BX

CALL F : дальний вызов, подпрограмма объявлена в другом сегменте

Выберите один ответ:

- ☐ a. 2
☐ b. 6
☒ c. 4

Ваш ответ неправильный.

Правильный ответ: 6

Укажите, на какую величину изменится указатель вершины стека после выполнения следующей команды: RETF 4

Решение: RETF соответствует возврату после ближнего перехода, то есть указатель увеличится на 4 байта. Необязательный параметр 4 указывает на то, что нужно дополнительно очистить в стеке 4 байта. Итого: $4 + 4 = 8$ - ответ.

Вопрос 10
Верно
Баллов: 1,00 из 1,00
Отметить
прос

Укажите, на какую величину изменится указатель вершины стека после выполнения следующей команды:

RETF 4

Выберите один ответ:

☐ a. 2

☒ b. 8

☐ c. 4

Ваш ответ верный.

Правильный ответ: 8

Что делает команда LEA?

Что делает команда LEA?

Выберите один ответ:

☐ a. Выполняет произвольные вычисления

☐ b. Это не команда, а директива - аналог оператора OFFSET

☒ c. Вычисляет эффективный адрес источника и помещает его в приёмник

Ваш ответ верный.

Правильный ответ: Вычисляет эффективный адрес источника и помещает его в приёмник

Что делает команда XLAT?

Что делает команда XLAT?

Выберите один ответ:

☐ a. Выполняет умножение операндов

☐ b. Меняет операнды местами

☒ c. Выполняет трансляцию значения AL по таблице в памяти

Ваш ответ верный.

Правильный ответ: Выполняет трансляцию значения AL по таблице в памяти

При выполнении каких команд меняется значение регистра IP?

При выполнении каких команд меняется значение регистра IP?

Выберите один ответ:

- ☐ a. Любых
- ☒ b. Команд вычисления адресов
- ☐ c. Команд передачи управления

Ваш ответ неправильный.

Правильный ответ: Любых

Какая команда не делает ничего?

Какая команда не делает ничего?

Выберите один ответ:

- ☒ a. NOP
- ☐ b. NOG
- ☐ c. NEG

Ваш ответ верный.

Правильный ответ: NOP

`mov ax, [bx][di]+6`

`mov ax, [bx][di]+6`

Выберите один ответ:

- ☐ a. $bx \cdot di + 6$
- ☒ b. $bx + di + 6$
- ☐ c. $bx \cdot 16 + di + 6$

Ваш ответ верный.

Правильный ответ: $bx + di + 6$

Какие прерывания могут быть замаскированы?

Какие прерывания могут быть замаскированы?

- ☐ а. программные
- ☐ б. аппаратные (асинхронные)
- ☒ в. внутренние (синхронные)

Ваш ответ неправильный.

Правильный ответ:
аппаратные (асинхронные)

Чем команда IRET отличается от RETF в архитектуре 8086?

Чем команда IRET отличается от RETF в архитектуре 8086?

- ☐ а. RETF восстанавливает из стека значение регистра флагов, IRET - нет
- ☐ б. IRET увеличивает SP на 6 байт, RETF - на 2
- ☒ в. IRET восстанавливает из стека значение регистра флагов, RETF - нет

Ваш ответ верный.

Правильный ответ:
IRET восстанавливает из стека значение регистра флагов, RETF - нет

Что необходимо для перехвата прерывания?

Что необходимо для перехвата прерывания?

- ☐ а. Обратиться к соответствующему порту ввода-вывода
- ☒ б. Указать адрес нового обработчика в таблице векторов прерываний
- ☐ в. Завершить программу резидентной

Ваш ответ верный.

Правильный ответ:
Указать адрес нового обработчика в таблице векторов прерываний

Где находится таблица векторов прерываний в реальном режиме работы процессора x86, каков ее размер и каков размер одного вектора?

Где находится таблица векторов прерываний в реальном режиме работы процессора x86, каков ее размер и каков размер одного вектора?

- ☐ a. Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 2 байта
- ☒ b. Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 4 байта
- ☐ c. Находится в начале памяти начиная с адреса 0, занимает 256 байт, размер вектора - 2 байта

Ваш ответ верный.

Правильный ответ:

Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 4 байта

Для взаимодействия с какими устройствами предназначен механизм ввода-вывода через порты?

Для взаимодействия с какими устройствами предназначен механизм ввода-вывода через порты?

- ☐ a. С устройствами внешней памяти
- ☐ b. С любыми внешними устройствами
- ☐ c. С устройствами ввода-вывода

Ваш ответ неправильный.

Правильный ответ:

С любыми внешними устройствами

Какие прерывания могут быть замаскированы?

Какие прерывания могут быть замаскированы?

- ☐ a. программные
- ☐ b. аппаратные (асинхронные)
- ☒ c. внутренние (синхронные)

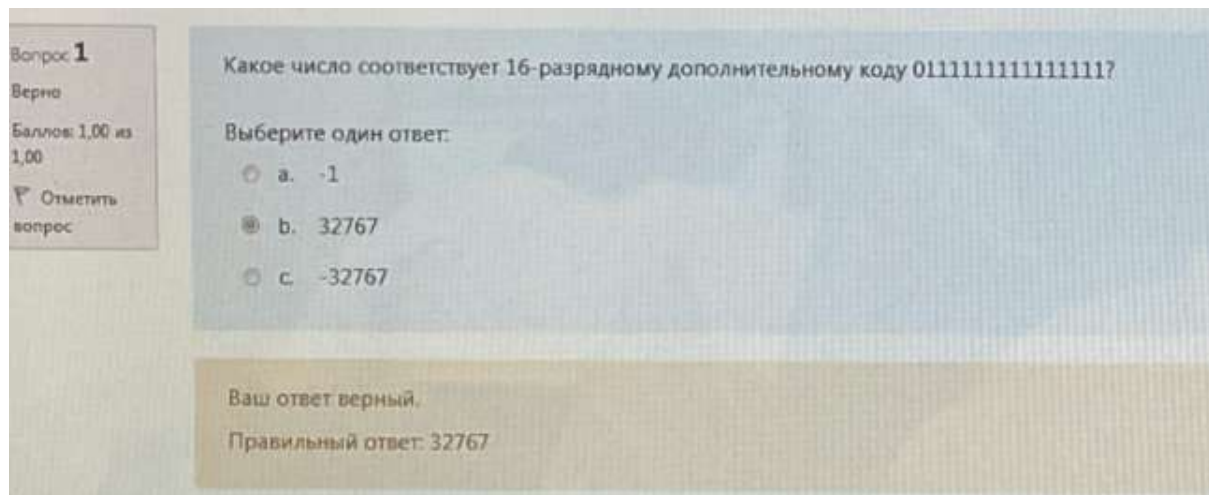
Ваш ответ неправильный.

Правильный ответ:

аппаратные (асинхронные)

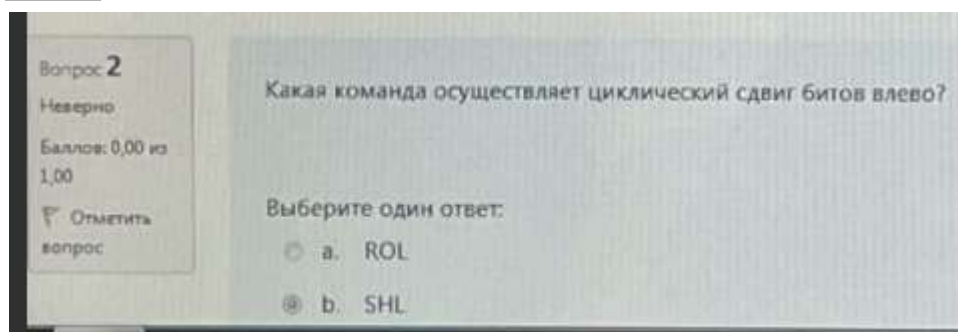
Какое число соответствует 16-разрядному дополнительному коду 0111111111111111?

Решение: Переводим в прямой код: 1000 0000 0000 0000. Отнимаем 1 (так как изначально число было с первым битом 0), получаем 111 1111 1111 1111, что соответствует 32767.



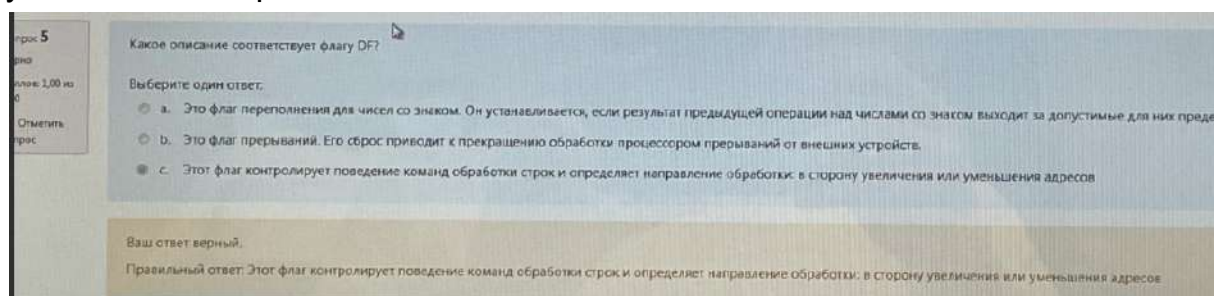
Какая команда осуществляет циклический сдвиг битов влево?

Ответ: SHL



Какое описание соответствует флагу DF?

Ответ: Этот флаг контролирует поведение команд обработки строк и определяет направление обработки: в сторону увеличения или уменьшения адресов.



Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 1A29h:37B4h?

Решение: $1A29h * 10h + 37B4h = 1DA44h$.

Вопрос 1

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 1A29h:37B4h

Выберите один ответ:

☐ a. 51DD0h

☒ b. 1DA44h ✓

☐ c. 1A2937B4h

Какова разрядность регистра DS?

Ответ: 16

Вопрос 2

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Какова разрядность регистра DS?

Выберите один ответ:

☒ a. 16 ✓

☐ b. 32

☐ c. 8

Какая формулировка соответствует характеристикам ближнего перехода?

Ответ: Метка занимает два байта, переход допустим в пределах текущего сегмента.

Вопрос 3

Неверно

Баллов: 0,00 из 1,00

Отметить вопрос

Какая формулировка соответствует характеристикам ближнего перехода?

Выберите один ответ:

☐ a. Метка занимает 2 байта, переход допустим в пределах всей памяти

☐ b. Метка занимает 2 байта, переход допустим в пределах текущего сегмента

☒ c. Метка занимает 1 байт, переход допустим в диапазоне от -128 до 127 байт от текущего значения IP ✗

Какая команда выполняет переход, соответствующий условию "больше"?

Ответ: JG

Вопрос 4

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Какая команда выполняет переход, соответствующий условию "больше"?

Выберите один ответ:

☐ a. JB

☐ b. JBE

☒ c. JG ✓

Имеется описание сегмента: ORG 10h, I DW 0, A DB 1, B DB FFh.

Чему равно OFFSET A?

Решение: ORG переносит внутреннюю переменную на 10h, DW занимает 2 байта, то есть 2h. Тогда OFFSET A = 10h + 2h = 12h.

Вопрос **5**

Верно

Баллов: 1,00
из 1,00

🚩 Отметить
вопрос

Имеется описание следующего сегмента:

DS SEGMENT

ORG 10h

I DW 0

A DB 1

B DB FFh

DS ENDS

Чему равно OFFSET A?

Выберите один ответ:

- ☒ a. 12h ✓
- ☐ b. 13h
- ☐ c. 10h

В каких командах языка ассемблера применяется сегментный префикс?

Ответ: При работе со значениями переменных

Вопрос **6**

Неверно

Баллов: 0,00
из 1,00

🚩 Отметить
вопрос

В каких командах языка ассемблера применяется сегментный префикс?

Выберите один ответ:

- ☐ a. При загрузке сегментных регистров
- ☐ b. При работе со значениями переменных
- ☒ c. При работе со смещениями ✗

Какая операция с сегментным регистром недопустима?

Ответ: Загрузка константы

Вопрос 7

Верно

Баллов: 1,00
из 1,00

🚩 Отметить
вопрос

Какая операция с сегментным регистром недопустима?

Выберите один ответ:

- ☒ а. Загрузка константы ✓
- ☐ б. Выгрузка в регистр общего назначения
- ☐ в. Загрузка из регистра общего назначения

`mov ax, [bp][di] + 4`

Ответ: `bp + di + 4`

Вопрос 8

Верно

Баллов: 1,00
из 1,00

🚩 Отметить
вопрос

`mov ax, [bp][di]+4`

Выберите один ответ:

- ☐ а. `bp*16+di+4`
- ☒ б. `bp+di+4` ✓
- ☐ в. `bp*di+4`

Что такое неупакованное двоично-десятичное число?

Ответ: Десятичная цифра, хранящаяся в байте.

Вопрос 9

Верно

Баллов: 1,00
из 1,00

🚩 Отметить
вопрос

Что такое неупакованное двоично-десятичное число?

Выберите один ответ:

- ☒ а. Десятичная цифра, хранящаяся в байте ✓
- ☐ б. Десятичное число от 0 до 99, хранящееся в байте
- ☐ в. Две десятичные цифры, хранящиеся в полубайтах одного байта

Что такое упакованное двоично-десятичное число?

Ответ: Две десятичные цифры, хранящиеся в полубайтах одного байта

Вопрос **10**

Верно

Баллов: 1,00
из 1,00

🚩 Отметить
вопрос

Что такое упакованное двоично-десятичное число?

Выберите один ответ:

- ☐ a. Десятичная цифра, хранящаяся в байте
- ☒ b. Две десятичные цифры, хранящиеся в полубайтах одного байта ✓
- ☐ c. Десятичное число от 0 до 99, хранящееся в байте

Какая команда осуществляет логический сдвиг битов влево?

Ответ: SHL

Вопрос **11**

Верно

Баллов: 1,00
из 1,00

🚩 Отметить
вопрос

Какая команда осуществляет логический сдвиг битов влево?

Выберите один ответ:

- ☐ a. SAL
- ☐ b. ROL
- ☒ c. SHL ✓

Что делает команда TEST?

Ответ: Побитовое И без сохранения результата

Вопрос **12**

Верно

Баллов: 1,00
из 1,00

🚩 Отметить
вопрос

Что делает команда TEST?

Выберите один ответ:

- ☒ a. Побитовое И без сохранения результата ✓
- ☐ b. Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ без сохранения результата
- ☐ c. Побитовое ИЛИ без сохранения результата

Укажите, на какую величину уменьшится указатель вершины стека после выполнения следующих команд?

PUSH AX, PUSH BX, POP CX, CALL F (подпрограмма в том же сегменте)

Решение: PUSH AX уменьшит указатель на 2, также как и BX - так как они занимают два байта. POP CX увеличит на 2 байта, а CALL F - ближний переход, поэтому метка тоже займёт два байта.

Итого: $2 + 2 - 2 + 2 = 4$ - ответ.

Вопрос **13**
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Укажите, на какую величину уменьшится указатель вершины стека после выполнения следующих команд:

PUSH AX
PUSH BX
POP CX
CALL F ; подпрограмма объявлена в том же сегменте

Выберите один ответ:

☒ a. 4 ✓
☐ b. 2
☐ c. 6

Укажите, на какую величину изменится указатель вершины стека после выполнения следующей команды: RETN 4

Решение: RETN соответствует возврату после ближнего перехода, то есть указатель увеличится на 2 байта. Необязательный параметр 4 указывает на то, что нужно дополнительно очистить в стеке 4 байта.

Итого: $2 + 4 = 6$ - ответ.

Вопрос **14**
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Укажите, на какую величину изменится указатель вершины стека после выполнения следующей команды:

RETN 4

Выберите один ответ:

☐ a. 4
☒ b. 6 ✓
☐ c. 2

Укажите, какая характеристика соответствует команде NEG?

Ответ: изменение знака

Вопрос **15**
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Укажите, какая характеристика соответствует команде NEG

Выберите один ответ:

☐ a. инверсия
☒ b. изменение знака ✓
☐ c. отрицание

Какая команда выполняет переход, соответствующий условию “меньше или равно”?

Ответ: JLE

Вопрос 4

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Какая команда выполняет переход, соответствующий условию “меньше или равно”?

Выберите один ответ:

- ☐ a. JAE
- ☐ b. JA
- ☒ c. JLE ✓

Имеется описание сегмента: ORG 100h, I DW 0, A DB 1.

Чему равно OFFSET A?

Решение: ORG переносит внутреннюю переменную на 100h, DW занимает 2 байта, то есть 2h. Тогда $\text{OFFSET A} = 100h + 2h = 102h$.

Вопрос 5

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Имеется описание следующего сегмента:

```
DS SEGMENT  
    ORG 100h  
    I DW 0  
    A DB 1  
DS ENDS
```

Чему равно OFFSET A?

Выберите один ответ:

- ☒ a. 102h ✓
- ☐ b. 2h
- ☐ c. 100h

Какова разрядность регистра IP? (без картинки)

Ответ: 16

Какая формулировка соответствует характеристикам короткого перехода? (без картинки)

Ответ: Метка занимает 1 байт, переход допустим в диапазоне от -128 до 127 байт от текущего значения IP.

Какая команда осуществляет циклический сдвиг битов вправо? (без картинки)

Ответ: ROR

Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 5089h:37B4h?

Решение: $5089h * 10h + 37B4h = 54044h$.

Вопрос 1

Неверно

Баллов: 0,00 из 1,00

Отметить вопрос

Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 5089h:37B4h

Выберите один ответ:

☐ a. 54044h

☒ b. 88CD0h

☐ c. 508937B4h

Ваш ответ неправильный.

Правильный ответ: 54044h

Имеется описание сегмента: ORG 20h, I DB 0, A DB 1.

Чему равно OFFSET A?

Решение: ORG переносит внутреннюю переменную на 20h, DB занимает 1 байта, то есть 1h. Тогда OFFSET A = 20h + 1h = 21h.

Вопрос 5

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Имеется описание следующего сегмента:

```
DS SEGMENT
    ORG 20h
    I DB 0
    A DB 1
DS ENDS
```

Чему равно OFFSET A?

Выберите один ответ:

☐ a. 22h

☒ b. 21h

☐ c. 20h

Ваш ответ верный.

Правильный ответ: 21h

Какого вида прерываний не существует?

Ответ: Системные

Вопрос **16**
Неверно
Баллов: 0,00 из 1,00
Отметить вопрос

Какого вида прерываний не существует?

- ☐ а. Системные
- ☐ б. Программные
- ☒ в. Синхронные

Ваш ответ неправильный.
Правильный ответ:
Системные

Где находится таблица векторов прерываний в реальном режиме работы процессора x86, каков её размер и каков размер одного вектора?

Ответ: Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 4 байта

Вопрос **17**
Неверно
Баллов: 0,00 из 1,00
Отметить вопрос

Где находится таблица векторов прерываний в реальном режиме работы процессора x86, каков её размер и каков размер одного вектора?

- ☒ а. Находится в начале памяти начиная с адреса 0, занимает 256 байт, размер вектора - 2 байта
- ☐ б. Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 4 байта
- ☐ в. Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 2 байта

Ваш ответ неправильный.
Правильный ответ:
Находится в начале памяти начиная с адреса 0, занимает 1024 байта, размер вектора - 4 байта

Значение каких регистров помещаются в стек при срабатывании прерывания?

Ответ: FLAGS, CS, IP

Вопрос **18**
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Значения каких регистров помещаются в стек при срабатывании прерывания?

- ☐ а. IP
- ☒ б. FLAGS, CS, IP
- ☐ в. CS, IP

Ваш ответ верный.
Правильный ответ:
FLAGS, CS, IP

Что программа обязательно должна сделать для корректного перехвата прерывания?

Ответ: Сохранить адрес старого обработчика прерывания

Вопрос **19**
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Что программа обязательно должна сделать для корректного перехвата прерывания?

- ☒ a. Сохранить адрес старого обработчика прерывания
- ☐ b. Обратиться к порту ввода-вывода
- ☐ c. Настроить свой стек для обработчика прерывания

Ваш ответ верный.
Правильный ответ:
Сохранить адрес старого обработчика прерывания

Что такое порты ввода-вывода в x86?

Ответ: Независимое адресное пространство для взаимодействия с устройствами.

Вопрос **20**
Верно
Баллов: 1,00 из 1,00
Отметить вопрос

Что такое порты ввода-вывода в x86?

- ☒ a. Независимое адресное пространство для взаимодействия с устройствами
- ☐ b. Специально выделенные ячейки ОЗУ
- ☐ c. Разъёмы для подключения устройств на материнской плате

Ваш ответ верный.
Правильный ответ: Независимое адресное пространство для взаимодействия с устройствами

Какая формулировка соответствует характеристикам дальнего перехода? (без картинки)

Ответ: Метка занимает 4 байта, переход допустим в пределах всей памяти.

Какая команда осуществляет циклический сдвиг битов влево без использования дополнительных разрядов? (без картинки)

Ответ: ROL

Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 72A0h:842Fh?

Решение: $72A0h * 10h + 842Fh = 7AE2F$.

Вопрос 1

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 72A0h:842Fh

Выберите один ответ:

☒ a. 7AE2Fh

☐ b. F6CF0h

☐ c. 72A0842Fh

Ваш ответ верный.

Правильный ответ: 7AE2Fh

Какова разрядность регистра DH?

Ответ: 8

Вопрос 2

Верно

Баллов: 1,00 из 1,00

Отметить вопрос

Какова разрядность регистра DH?

Выберите один ответ:


☒ a. 8

☐ b. 16

☐ c. 32

Ваш ответ верный.

Правильный ответ: 8



Машинно-зависимые языки программирования, лекция 4

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2021 г.



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



Прерывания

- 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



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



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 - механизм 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
```



Вопросы к рубежному контролю

1. Регистры общего назначения.
2. Сегментные регистры. Адресация в реальном режиме. Понятие сегментной части адреса и смещения.
3. Регистры работы со стеком.
4. Структура программы. Сегменты.
5. Прерывание 21h. Примеры ввода-вывода.
6. стек. Назначение, примеры использования.
7. Регистр флагов.
8. Команды условной и безусловной передачи управления.
9. Организация многомодульных программ.
10. Подпрограммы. Объявление, вызов.
11. Арифметические команды.
12. Команды побитовых операций.
13. Команды работы со строками.
14. Прерывания. Обработка прерываний.
15. Работа с портами ввода-вывода.

Машинно-зависимые языки программирования

Лекция 1

Краткая история компьютеров

Изначально слово компьютер означало не устройство, а профессию. Первоначально это были люди, которые решали какие-то сложные задачи (типа для карт для ориентации по звездам). При усложнении задач постепенно начали появляться вычислительные машины (арифмометры и тд).

Эти первые машины были механическими. К началу 20 века научились делать электрические (выполняли действия быстрее). Но и у тех и у других была проблема с тем, что они решали только одну конкретную задачу. Поэтому в середине 20 века назрел вопрос о создании универсальной машины, которая могла бы выполнять произвольные алгоритмы без перестраивания машины.

Американский ученый фон Нейман работал над этой проблемой. Ему удалось сформулировать некие основные принципы.

Принципы фон Неймана:

1. Использование двоичной системы счисления
2. Программное управление ЭВМ
3. Принцип однородности памяти (память используется не только для хранения данных, но и для хранения программ)
4. Принцип адресности (все ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы)
5. Возможность условного перехода в процессе выполнения программы (чтобы можно было создавать не только линейные алгоритмы)

Схема архитектуры машины фон Неймана:

- Абстрактный процессор – вычислитель
 - Управляющее устройство (интерпретирует команды и их выполняет)
 - Арифметико-логическое устройство (выполняет арифметические и некоторые логические бинарные операции над данными)
- Абстрактная память (из памяти процессор берет значения, код, и что-то туда пишет)
- Также обозначены абстрактный ввод и вывод



Структурная схема ЭВМ:



- Центральным устройством является системная **шина** (это некоторая абстракция, которая обеспечивает взаимодействие всех устройств)
 - шина управления
 - шина адреса
 - шина данных

Все три составляющие используются для любого обмена. Например, если процессору необходимо считать какое-то значение из памяти, то он выставляет на шину адреса номер ячейки памяти и по шине управления отправляет сигнал. Память считывает значение с шины адреса, ищет нужную ячейку, выставляет ее значение на шину данных. Данные получены.

В компьютере функции системной шины по сути выполняет материнская плата. Также на практике шин обычно несколько (быстрая между процессором и оперативкой, шины для подключения внешних устройств [pci, usb])

- **Центральный процессор**
 - Блок регистров
(регистр – внутренняя ячейка памяти процессора. Обычно таких ячеек не очень много, десятков-полтора ячеек из нескольких байтов)
 - АЛУ
 - УУ

Процессор много взаимодействует с оперативкой. В частности, во время работы программа должна быть загружена в оперативную память. Это единственная память, к которой процессор имеет прямой доступ.

- **Внутренняя память**
 - ОЗУ (RAM) // очищается при выключении компьютера
 - ПЗУ // нужна для хранения данных

В ПЗУ также хранится стартовая программа для загрузки компьютера. (как это работает: при включении процессор всегда начинает работать с фиксированного адреса. Производители материнских плат этим пользуются и настраивают ПЗУ так, чтобы данные из ПЗУ копировались в ОЗУ по нужным адресам, таким образом добиваются того, что при включении процессор начинает читать именно стартовую программу (bios))

BIOS:

1. Выполняет первичную диагностику устройства
 2. Определяет какие устройства подключены
 3. Находит внешнюю память, находит загрузочный диск
 4. Загружает операционную систему
- Устройства ввода (клавиатура, мышка и тд)
 - Устройства вывода (монитор, принтер и тд)
 - Внешняя память (все возможные виды накопителей – дискеты, магнитные ленты, жесткие диски, оптические диски, флешки, SSD-диски)

Память

Почему так много разных видов памяти?

Чем более быстродействующая память, тем она дороже стоит => тем меньше ее можно вставить.

Пирамидка быстродействия:

- регистры процессора
- кэш-память процессора (несколько уровней)
- оперативная память
- внешние

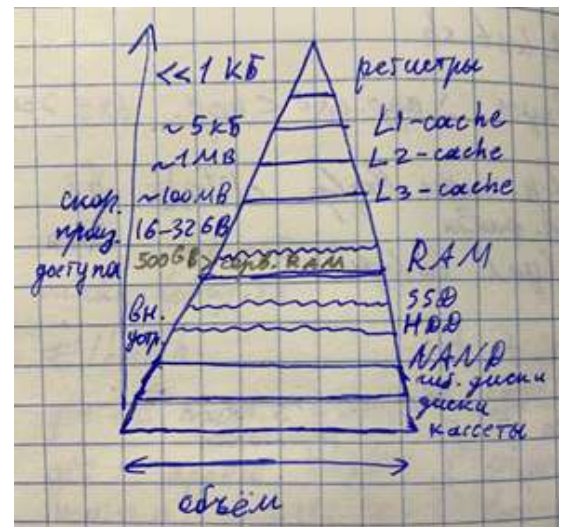


Минимальная адресуемая единица памяти¹ – байт

Побитово работа с памятью никогда не осуществляется (это нужно для сокращения адресных линий)

(доп: байт очень удобно представлять в 16-ричной системе. В этом случае он будет выглядеть как ровно две 16-ричные цифры)

Машинное слово² – машинно-зависимая величина, измеряемая в битах, равна разрядности регистров процессора и шины данных.



16-ти разрядный процессор => размер его регистров 16 бит = 2 байта => шина данных такая же

Параграф – 16 байт

ASCII – однобайтовая символьная таблица (7-битная кодировка)

Первые 32 байта такой таблицы занимают различные непечатные символы.

¹ Минимальная адресуемая единица памяти по костру – минимальная разница между двумя адресами в памяти

² Машинное слово по костру – сообщение фиксированной длины, с помощью которых компьютер общается со своими составными частями

Системы счисления

Двоичная (binary)

- 0, 1, 10, 11, 100, 101...
- $2^8 = 256$
- $2^{10} = 1024$
- $2^{16} = 65536$
- Суффикс - b. Пример: 1101b

Шестнадцатеричная (hexadecimal)

- 0, 1, ..., 8, 9, A, B, C, D, E, F, 10, 11, 12, ..., 19, 1A, 1B, ...
- $2^4 = 10_{16}$
- $2^8 = 100_{16}$
- $2^{16} = 10000_{16}$
- Суффикс - h (10h - 16). Некоторые компиляторы требуют префикса 0x (0x10)

$$101101101111000_2 = B6F8_{16}$$

Так как первое время мы будем изучать 16-ти разрядный процессор, то нам важно понимать, что $2^{16} = 64$ Кбайт.

Представление отрицательных чисел

Знак - в старшем разряде (0 - "+", 1 - "-").

Возможные способы:

- прямой код
- обратный код (инверсия)
- дополнительный код (инверсия и прибавление единицы)

Примеры доп. кода на 8-разрядной сетке

-1:

1. 00000001
2. 11111110
3. 11111111

Смысл: $-1 + 1 = 0$ (хоть и с переполнением):

$$11111111 + 1 = (1)00000000$$

-101101:

1. 00101101
2. 11010010
3. 11010011

Плюс хранения в дополнительном коде – в плане арифметических выражений ничего не меняется.

Виды современных архитектур ЭВМ

- x86-64 8086 (16-разр.) \rightarrow x86 (32-разр.) \rightarrow x64 (64-разр.)
- ARM
- IA64
- MIPS (включая Байкал)
- Эльбрус

Семейство процессоров x86 и x86-64

- Микропроцессор 8086: 16-разрядный, 1978 г., 5-10 МГц, 3000 нм
- Предшественники: 4004 - 4-битный, 1971 г.; 8008 - 8-битный, 1972 г.; 8080 - 1974 г.
- Требуется микросхем поддержки
- 80186 - 1982 г., добавлено несколько команд, интегрированы микросхемы поддержки
- 80286 - 1982 г., 16-разрядный, добавлен защищенный режим
- 80386, 80486, Pentium, Celeron, AMD, ... - 32-разрядные, повышение быстродействия и расширение системы команд
- x86-64 (x64) - семейство с 64-разрядной архитектурой
- Отечественный аналог - K1810BM86, 1985 г.



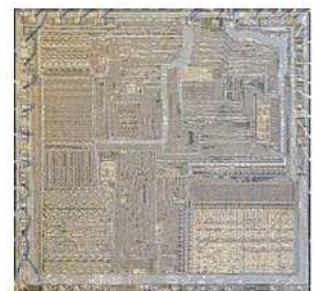
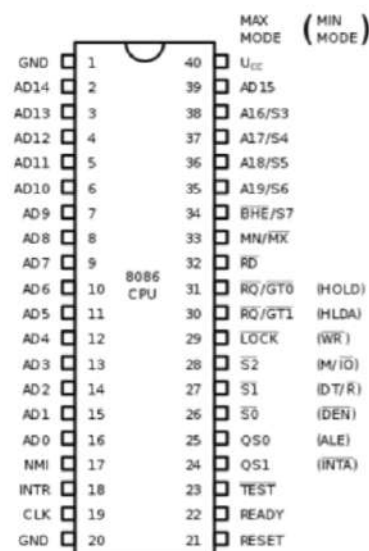
Устройство 8086

Если с процессора 8086 снять корпус, то внутри окажется небольшой квадратик.

// сложная система, переслушайте сами

A – шина адреса (здесь 20 \Rightarrow нам доступно 2^{20} адресных ячеек)

D – шина данных (их количество определяет разрядность, здесь 16)



Структура блока регистров (архитектура 8086 с точки зрения программиста)

1. IP – указатель команд (хранит смещение команды, которая будет выполнена следующей)
2. Регистры общего назначения – используются в арифметических, логических и т.д. операциях (для большинства операций являются взаимозаменяемыми). Базово AX – умножение и деление, BX – вычисление адресов, CX – циклы, DX – обмен с внешними устройствами)
3. Регистры в центре близки к регистрам общего назначения
4. Сегментные регистры (отвечают за сегменты памяти [вторая составл адреса])



Язык ассемблера

Машинная команда – инструкция (в двоичном коде) из аппаратного определенного 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.

Чтобы получить программу в машинном коде необходимо ее из исходного кода скомпилировать и получить исполняемый файл.

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

В dos и windows расширения у исполняемых файлов `exe` и `com`

Компилятор – программа для преобразования исходного текста программы в объектный модуль.

Компоновщик – программа для связывания нескольких объектных файлов в исполняемый.

Последовательность запуска программы операционной системой:

- Определение формата файла
- Чтение и разбор заголовка
- Считывание разделов исполняемого модуля в ОЗУ по необходимым адресам
- Подготовка к запуску (загрузка библиотек)
- 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 останова.

.COM – простейший формат исполняемого файла в dos и ранней винде.

Особенности:

- не имеет заголовка
- состоит из одной секции, не превышающей 64Кб
- 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 записывает служебная структура DOS, описывающая программу – PSP (префикс программного обеспечения)
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 передается по адресу CS:0100h.

Классификация команд процессора 8086:

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

Команды пересылки данных
(типа оператор присваивания)

MOV < приемник >, < источник >

Источник: непосредственный операнд (константа, включенная в машинный код) / PОН (?) / регистр общего назначения / сегментный регистр / переменная

Приемник: все то же самое, кроме константы

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

Ограничения:

- Нельзя скопировать переменную в переменную
- В сегментные регистры напрямую нельзя записывать константы (записать можно только из регистра общего назначения)

```
MOV [0123h], [2345h]
MOV DS, 1000h
```

Целочисленная арифметика

- ADD <приемник>, <источник> – арифметическое сложение приемника и источника. Сумма помещается в приемник, источник не меняется
- SUB <приемник>, <источник> – арифметическое вычитание источника из приемника
- MUL <источник> – беззнаковое умножение. Умножаются источник и AL/AX, в зависимости от размера источника. Результат помещается в AX либо DX:AX.
- DIV <источник> – целочисленное беззнаковое деление. Делится AL/AX на источник. Результат помещается в AL/AX, остаток – в AH/DX.
- INC <приемник> – инкремент на 1
- DEC <приемник> – декремент на 1

Побитовая арифметика

- AND <приемник>, <источник> – побитовое И (AND al, 00001111b)
- OR <приемник>, <источник> – побитовое ИЛИ
- XOR <приемник>, <источник> – побитовое исключающее ИЛИ (XOR AX, AX)
- NOT <приемник> – инверсия

Команда безусловной передачи управления JMP *JMP <операнд>*

- Передает 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.

Команда, которая ничего не делает NOP

- Занимает место и время
- Размер – 1 байт, код – 90h
- Назначение – задержка выполнения либо заполнения памяти, например, для выравнивания

Пример

...

XOR AX, AX

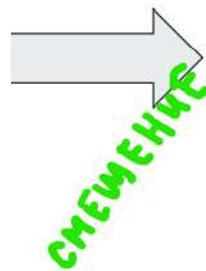
MOV BX, 5

label1:

INC AX

ADD BX, AX

JMP label 1



AX 0000	SI 0000	CS 19F5	IP 0100
BX 0000	DI 0000	DS 19F5	
CX 0024	BP 0000	ES 19F5	HS 19F5
DX 0000	SP FFFE	SS 19F5	FS 19F5
CMD >			
НАШ КОД ИЛИ КОМАНДА			
0100	33C0	XOR	AX, AX
0102	BB0500	MOV	BX, 0005
0105	40	INC	AX
0106	03D8	ADD	BX, AX
0108	EBFB	JMP	0105
010A	BA1401	MOV	DX, 0114
010D	CD21	INT	21
010F	B44C	MOV	AH, 4C

братный подсказ

Взаимодействие программы с внешней средой

Прерывания – аппаратный механизм для приостановки выполнения текущей программы и передачи управления специальной программе – обработчику прерывания.

Основные виды:

- аппаратные
- программные

Вызов – `int <номер>`

21h – прерывание DOS. Передается через регистр AH. Параметры функций передаются собственным способом, он описан в документации.

Память в реальном режиме работы процессора

Реальный режим работы - режим совместимости современных процессоров с 8086.

Доступен 1 Мб памяти (2^{20} байт), то есть разрядность шины адреса - **20 разрядов**.

Физический адрес получается сложением адреса **начала сегмента** (на основе сегментного регистра) и **смещения**.

Сегментный регистр хранит в себе **старшие 16 разрядов** (из 20) адреса начала сегмента. 4 младших разряда в адресе начала сегмента всегда нулевые. Говорят, что сегментный регистр содержит в себе **номер параграфа начала сегмента**.

Память в реальном режиме работы процессора - пример

Номер параграфа начала сегмента

19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

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

5678h:1234h =>

56780

+1234

579B4

Вычисление физического адреса выполняется процессором аппаратно, без участия программиста.

Распространённые пары регистров: CS:IP, DS:BX, SS:SP

Структура памяти программы. Виды сегментов. Назначение отдельных сегментных регистров

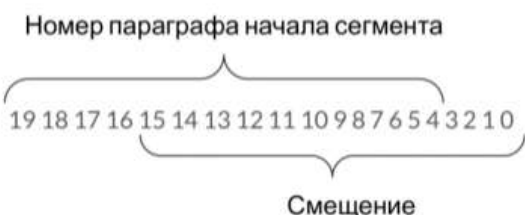
- Сегмент кода - регистр CS. Командой MOV изменить невозможно, меняется автоматически по мере выполнения команд.
- Сегмент данных. Основной регистр - DS, при необходимости дополнительных сегментов данных задействуются ES, FS, GS.
- Сегмент стека - регистр SS

Машинно-зависимые языки программирования

Лекция 2.

Принцип формирования 20-разрядного физического адреса из 16-разрядного сегментного адреса (старшие разряды) и 16-разрядного смещения (младшие разряды):

Память в реальном режиме работы процессора - пример



[SEG]:[OFFSET] => физический адрес:

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

$$\begin{array}{r} 5678\text{h}:1234\text{h} \Rightarrow \\ 56780 \\ + 1234 \\ \hline 579\text{B}4 \end{array}$$

Вычисление физического адреса выполняется процессором аппаратно, без участия программиста.

Распространённые пары регистров: CS:IP, DS:BX, SS:SP

То есть берется адрес, сдвигается на 4 разряда, прибавляется смещение.

Пример:

Допустим, что у нас в компьютере ровно 1 Мб памяти. Тогда ячейки будут пронумерованы с 0 до FFFFF. Блоки памяти по 16 байт будут называться **параграфами**. Номер параграфа – старшие четыре цифры из 5 (или старшие 16 из 20 двоичных разрядов).

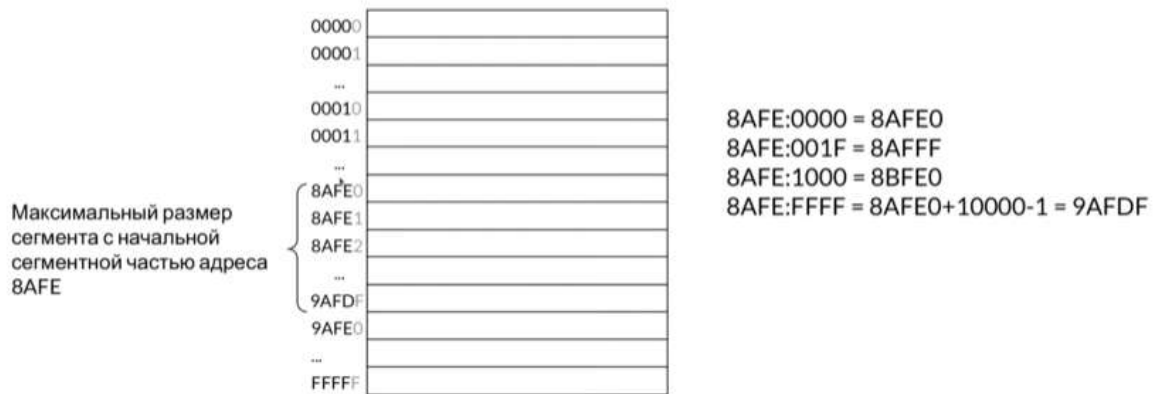
Память 8086 (20-разрядная адресация)



Сегмент может начинаться только с адреса, кратного параграфу, где последний 16-ричный разряд равен 0 (то есть с начала какого-то параграфа, например, 00000 или 00010 и тд). ?

К сегментной части адреса далее добавляется какое-то смещение, которое позволяет уже адресовать некоторый конкретный байт памяти и получить полный физический адрес ячейки в оперативной памяти.

Предположим, некоторый сегмент начинается с адреса 8AFE0 (соответственно, сегментная часть адреса здесь это 8AFE).



Максимальное смещение, которое нам доступно – это 2^{16} . Поэтому последней доступной ячейкой этого сегмента будет $8AFE0 + 2^{16} - 1 = 9AFDF$.

На практике размер сегмента зависит от данных, которые мы в него записываем. Может выделяться ровно 64 Кб, может меньше.

Сегменты

Программа может состоять из сегментов трех типов:

- Сегмент кода (регистр CS) // обязательный
- Сегмент данных (основной регистр – DS, для дополнительных сегментов – ES, FS, GS)
- Сегмент стека (регистр SS)

Команда организации цикла LOOP

LOOP метка – уменьшается регистр CX на 1 и выполняет переход на метку, если CX не равен 0. Метка не может быть дальше -128...127 байт от команды.

Структура программы на ассемблере

Основное, из чего состоит программа – это **файлы с исходным кодом**. Такие файлы в ассемблере называются модулями.

Основное, что находится в каждом модуле – это **описание сегментов** (то есть описание блоков памяти). Внутри сегментов находятся:

- Команды процессора
- Инструкции описания структур данных, выделения памяти для переменных и констант

- макроопределения

Полный формат строки (все части необязательны):

метка команда/директива операнды ; комментарий

Метки

Записываются по-разному, в зависимости от того, где он применяются.

В коде

- Имя метки обязательно отделяется двоеточием
- Имя может состоять из латинских букв, цифр (не могут идти в начале), знаков подчеркивания и еще нескольких спец символов
- Обычно используются в командах передачи управления
- Пример:

```
mov cx, 5
label1:
    add ax, bx
    loop label1 ; цикл выполняется 5 раз
```

В данных

- Используются для объявления переменных или констант
- label – ключевое слово, для определения переменной
- Возможные типы:
 - BYTE – 1 байт
 - WORD – 2 байта
 - DWORD – 4 байта
 - FWORD – 6 байт
 - QWORD – 8 байт
 - TBYTE – 10 байт
- // И два доп типа для хранения меток в коде, адресов команд
 - NEAR – метка ближнего перехода
 - FAR – метка дальнего перехода
- Пример:

```
метка label тип
```
- Использование в макросах
- Пример:

```
метка EQU выражение (EQU можно заменить на =)
```

Директивы

Директива – инструкция ассемблеру, влияющая на процесс компиляции и не являющаяся командой процессора. Обычно не оставляет следов в формируемом машинном коде.

Псевдокоманда – директива ассемблера, которая приводит к включению данных или кода в программу, но не соответствующая никакой команде процессора.

Псевдокоманды определения данных указывают, что в соответствующем месте располагается переменная, резервируют под нее место заданного типа, заполняют значением и ставят в соответствие метку.

Виды: DB(1), DW(2), DD(4), DF(6), DQ(8), DT(10)

Примеры:

- а DB 1 – выделить ячейку размером 1 байт, присвоить ей имя а и положить туда значение 1
- float_number DD 3.5e7 – объявляем вещественную переменную размером 4 байта и помещаем в нее вещественное число 3.5e7
- text_string DB 'Hello world!' – если строка задается вот так в кавычках (или несколько чисел перечисляются через запятую), то будет выделен не один байт, а столько сколько символов в строке (перечисленных значений) по одному байту.

DUP – заполнение повторяющимися данными

метка DB 100 DUP какое-то значение – будет выделено 100 байт с указанным значением

? – неинициализированное значение

uninit DW 512 DUP (?)

Объявление сегмента программы

имя SEGMENT [READONLY] [выравнивание] [тип] [разрядность] ['класс']

...

имя ENDS

READONLY – если во время выполнения компилятор увидит, что идет попытка записи в этот сегмент, компиляция будет остановлена с ошибкой.

Выравнивание – обозначает с каких адресов будет начинаться сегмент:

- BYTE – сегмент может начинаться с произвольного адреса
- WORD – сегмент всегда начинается с адреса, кратного 2
- DWORD – сегмент всегда начинается с адреса, кратного 4
- PARA – сегмент всегда начинается с начала параграфа (адреса, кратного 16) !
это значение по умолчанию
- PAGE – сегмент начинается с адреса, кратного 256

Тип:

- PUBLIC – сегменты с одним именем будут располагаться в памяти непосредственно друг за другом, независимо от того, в каком порядке они были объявлены в исходном коде

- STACK – сегмент будет использоваться под стек. Все сегменты в исходном коде с таким типом будут объединяться в один для увеличения размера стека.
- COMMON – сегменты также будут объединяться, но не друг за другом, а как бы накладываться (те начинаться с одного и того же адреса)
- AT – должен иметь определенный аргумент (номер параграфа начала сегмента), обозначает, что сегмент будет загружаться в память по некоторому фиксированному постоянному адресу (вне зависимости от других сегментов)
- PRIVATE – сегмент не объединяется с другими, существует сам по себе **!это значение по умолчанию**

Класс – любая метка, взятая в одинарные кавычки. Сегменты одного класса будут расположены в памяти друг за другом.

Директива .model (модель памяти)

.model модель, язык, модификатор

// Нужны для сокращения записи программ определенных типов

Модели:

- TINY – один сегмент на все (пример – COM программа)
- SMALL – код в одном сегменте, данные и стек – в другом
- COMPACT – допустимо несколько сегментов данных
- MEDIUM – код в нескольких сегментах, данные в одном
- LARGE, HUGE

Язык: C, PASCAL, BASIC, SYSCALL, STDCALL.

(этот параметр нужен для связывания с языками высокого уровня и вызова подпрограмм, то есть на случай, если мы код на ассемблере захотим подключить куда-нибудь)

Модификатор – способ подключения стека:

- NEARSTACK – ближний
- FARSTACK – дальний

Директива END

...

END [точка_входа]

Этой директивой должно заканчиваться описание любого модуля.

точка_входа – имя метки в сегменте кода, указывающей на команду, с которой начнется исполнение программы

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

звучит как бред

Сегментный префикс. Директива ASSUME.

Если мы собираемся работать с переменными, то зная, что переменная находится в каком-либо из сегментов данных (с этим сегментом может быть связано целых 4 сегментных регистра – DS, ES, FS, GS) мы понимаем, что процессору надо явно давать понять с каким именно из сегментов мы собираемся работать.

Можно писать полную запись DS:Var1

Можно пользоваться упрощением через директиву ASSUME регистр : имя сегмента. Эта директива устанавливает значение сегментного регистра по умолчанию.

В программе слева нас интересует строчка *mov ax, [Var2]*. Когда компилятор до нее дойдет, он посмотрит из какого сегмента переменная Var2, найдет последний ASSUME, который связан с Data2 и неявно сам подставит правильный сегментный префикс.

```
Data1 SEGMENT WORD 'DATA'
Var1 DW 0
Data1 ENDS

Data2 SEGMENT WORD 'DATA'
Var2 DW 0
Data2 ENDS

Code SEGMENT WORD 'CODE'
ASSUME CS:Code
ProgramStart:
    mov ax,Data1
    mov ds,ax
    ASSUME DS:Data1
    mov ax,Data2
    mov es,ax
    ASSUME ES:Data2
    mov ax,[Var2]

Code ENDS
END ProgramStart
```

Прочие директивы

- Задание набора допустимых команд: .8086, .186, .286,586, .686
- Управление программным счетчиком:
 - ORG значение – начиная с этой директивы отступ в сегменте будет идти с переданного значения (org 100h – сразу пропустить 256 байт)
 - EVEN – автоматически выровнять по четному адресу то, что идет после нее
 - ALIGN значение – явно задать какую-то кратность выравнивания

// Выравнивание имеет смысл для ускорения работы программы в будущем, потому что процессор работает с памятью не по байтам, а по ячейкам, размер которых равен машинному слову.

- Глобальные объявления
 - public – объявление метки доступной из других модулей
 - comm
 - extrn – подключить метку из другого модуля
 - global

- Условное ассемблирование

IF выражение

...

ELSE

...

ENDIF

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)

Почему существует три вида переходов? – для экономии места.

Индексные регистры SI и DI

SI (source index) – индекс источника

DI (destination index) – индекс приемника

Могут использоваться в большинстве команд, как регистры общего назначения.

Применяются в специфических командах поточной обработки данных.

Способы адресации

Для процессора 8086 выделяют 6 возможных способов адресации ячеек:

- Регистровая адресация (mov ax, bx)
- Непосредственная адресация (mov ax, 2)
- Прямая адресация (mov ax, ds:0032)
- Косвенная адресация (mov ax, [bx])
В 8086 допустимы BX, BP, SI, DI
- Адресация по базе со сдвигом (mov ax, [bx]+2; mov ax, 2[bx])
- Адресация по базе с индексированием (допустимы BX + SI, BX + DI, BP + SI, BP + DI)
 - mov ax, [bx + si + 2]
 - mov ax, [bx + 2][si]
 - mov ax, 2[bx][si]
 - mov ax, [bx][si] + 2
 - mov ax, [bx][si + 2]

Регистр FLAGS

– специальный регистр, к которому напрямую обращаться нельзя. Используется ни как единое число, а как отдельные биты (то есть все его 16 битов независимы друг от друга, каждый имеет свое собственное значение)

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	-
• 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		• TF (trap flag) - флаг трассировки		• IF (interrupt enable flag) - флаг разрешения прерываний		• DF (direction flag) - флаг направления	
												• OF (overflow flag) - флаг переполнения		• IOPL (I/O privilege flag) - уровень приоритета ввода-вывода	
														• NT (nested task) - флаг вложенности задач	

Черные – биты состояния

Синие – системные флаги

Зеленый – флаг управления DF

Жирные – флаги, которые можно непосредственно изменять

Состояния:

CF – флаг переноса. Возникает в случае перехода через разрядную сетку при работе с беззнаковыми числами.

PF – флаг четности (0 или 1). Указывает, какое количество бит получилось в младшем байте результата. Нужен для взаимодействия с какими-то устройствами, передачи данных, контроля корректности передачи данных.

AF – вспомогательный флаг переноса. Выставляется, если произошел перенос из младшего полубайта в старший полубайт (если биты нумеровать от 1 до 8, то из 4 в 5 бит). Нужен при работе с упакованными двоично-десятичными числами, когда у нас в байте не 256 различных значений, а всего 100.

ZF – флаг нуля. Принимает единичное значение, когда в качестве результата арифметической операции получился 0.

SF – флаг знака. Этот флаг всегда равен старшему биту результата.

OF – флаг переполнения. Предназначается для работы со знаковыми числами, когда старший бит предполагается знаковым и происходит его переполнение.

Системные флаги:

TF – флаг трассировки. Служит для отладки программы

IF – флаг разрешения прерываний. Если этот флаг выставлен в 1, то прерывания будут срабатывать.

IOPB (появился в 256 процессоре) – уровень приоритета ввода-вывода.

NT (появился в 256 процессоре) – флаг вложенности задач.

Флаг управления

DF – используется в командах поточной обработки данных, для которых нужны регистры SI и DI.

Команда сравнения **CMP <приемник>, <источник>**

Источник – число, регистр или переменная

Приемник – регистр или переменная (не может быть переменной одновременно с источником)

Вычитает источник из приемника, результат никуда не сохраняется, выставляются флаги CF, PF, AF, ZF, SF, OF

Команды условных переходов **J..**

// работают как jmp, но переход выполняют не всегда, а только при наличии определенной комбинации флагов

- Переход только 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	Состояние флагов для выполнения перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнение	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	Состояние флагов для выполнения перехода	Знаковый
JB JNAE JC	Если ниже Если не выше и не равно Если перенос	CF = 1	нет
JNB JAE JNC	Если не ниже Если выше или равно Если нет переноса	CF = 0	нет
JBE JNA	Если ниже или равно Если не выше	CF = 1 или ZF = 1	нет
JA JNBE	Если выше Если не ниже и не равно	CF = 0 и ZF = 0	нет

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JL JNGE	Если меньше Если не больше и не равно	SF <> OF	да
JGE JNL	Если больше или равно Если не меньше	SF = OF	да
JLE JNG	Если меньше или равно Если не больше	ZF = 1 или SF <> OF	да
JG JNLE	Если больше Если не меньше и не равно	ZF = 0 и SF = OF	да

Команда **TEST** <приемник>, <источник>

Аналогична CMP, но выполняет не арифметическое вычитание, а логическое умножение (те AND). Результат также, как и в CMP никуда не сохраняется.

Выставляются флаги SF, ZF, PF.

С помощью этой команды проверяют не равен ли какой-то регистр 0.

Прерывание

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

Виды прерываний:

- аппаратные (асинхронные) – события от внешних устройств
- внутренние (синхронные) – события в самом процессоре (деление на 0)
- программные – вызванные командой INT

Прерывание DOS 21h (33 в десятичной системе):

- Аналог системного вызова в современных ОС
- Используется наподобие вызова подпрограммы
- Номер функции передается через AH

Прерывание DOS - вывод на экран в текстовом режиме

Функция	Назначение	Вход	Выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающийся символом \$	-

02 – вывод одного символа в стандартный поток вывода. При этом в ah надо положить 02, в dlн нужно положить ascii-код символа и вызвать 21-ое прерывание.

09 – вывод строки символов. На вход этой функции передается адрес строки, то есть должно быть заполнено два регистра. В DS должна быть сегментная часть, а в DX смещение строки. Строка обязательно должна заканчиваться символом \$ (по нему определяется куда нужно вывести).

Прерывание DOS - ввод с клавиатуры

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL - ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL = FF	AL - ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL - ASCII-код символа
08	Считать символ без эха	-	AL - ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры	-	AL=0, если клавиша не была нажата, и FF, если была
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	

// “с эхом” означает, что символ не только будет считан, но и окажется на экране в положении курсора

Чтобы считать один символ мы помещаем в ah 01, вызываем прерывание и после этого из al считываем значение.

Чтобы считать строку нужно подготовить соответствующий буфер

Машинно-зависимые языки программирования

Лекция 3

Команды, которые мы не успели изучить (немного забегаем вперед, рассматривая команды не только процессора 8086):

CMOVcc <приемник> <источник> – команда условной пересылки данных. Будет пересылать значения из источника в приемник только при выполнении условия (аналогично команде Jcc).

XCHG <операнд1>, <операнд2> – обмен операндов между собой. Выполняется над двумя регистрами, либо над регистром и переменной.

? Почему она не может поменять местами две переменные?

Потому что шина адреса у нас всего одна и с оперативной памятью мы можем работать только в одну сторону (либо мы выставляем адрес и с него что-то читаем, либо выставляем и пишем). Выставить сразу два адреса нельзя, поэтому поменять сразу два значения памяти нельзя.

XLAT [адрес] /XLAB – трансляция в соответствии с таблицей.

Эта команда помещает в AL байт из таблицы¹ по адресу DS:BX со смещением относительно начала таблицы, равным AL.

Особенности:

1. Адрес, указанный в исходном коде, не обрабатывается компилятором и служит в качестве комментария.
2. Если в адресе явно указан сегментный регистр, он будет использован вместо DS (то есть смещение всегда берется из BX, а сегментная часть из указанного регистра)

Такая команда может использоваться, например, для транслитерации или для перевода 16-ричных чисел в символы (где мы значения от 0 до 9 будем конвертировать в цифры, а значения от 10 до 15 будем конвертировать в буквы от А до F).

Фактически, операция нужна для конвертации по словарю (причем “не линейно, а по массиву”). Мы объявляем целевой массив в памяти, настраиваем на него адреса DS:BX и в AL кладем номер, который хотим из этого массива получить. После этого этот номер подменяется значением из этого массива.

Да ну бред какой-то

Если вы хотите у меня спросить, как это работает, то лучше не спрашивайте

¹ Про таблицу – про себя надо читать это как массив, который не превышает 256 ключей памяти

LEA <приемник>, <источник> – вычисление эффективного адреса.

Вычисляет эффективный адрес источника и помещает его в приемник. Под адресом здесь тоже нужно понимать смещение.

Используется для:

1. Вычисления адресов на лету (позволяет вычислять адреса, описанные сложными методами адресации)
2. Быстрых вычислений

```
lea bx, [bx + bx * 4]
```

```
lea bx, [ax + 12]
```

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

(поддерживается сложение с константой и сложение/умножение какого-то регистра)

ADD; ADC; SUB; SBB <п>, <и> (двоичная арифметика)

С командами ADD и SUB мы уже знакомы. Команды ADC и SBB нужны для сложения каких-то больших чисел, которые в наши регистры не помещаются (работа с 32-разрядными числами).

—

▮ Есть у нас два слова. Старшая часть первого записана в dx, младшая в ax. Второе слово аналогично делит регистры bx и cx. Как мы можем их сложить? Складываем младшие половинки, в результате они могут сложиться как без переполнения, так и с переполнением (в этом случае у нас выставляется флаг переполнения), после нужно сложить старшие половинки, но так, чтобы учесть вопрос переполнения. Чтобы не делать это руками, используем ADC (сложение с учетом флага CF).

```
add ax, cx
```

```
adc dx, bx
```

Фактически, мы сложим приемник, источник и флаг CF. В результате в паре регистров dx и ax будет искомая сумма.

—

С вычитанием все точно также

```
sub ax, cx
```

```
sbb dx, bx
```

Обе команды выставляют сразу все 6 возможных арифметических флагов (CF, OF, SF, ZF, AF, PF).

IMUL, MUL (умножение чисел со знаком)

IMUL <источник>²

IMUL <приемник>, <источник>

IMUL <приемник>, <источник1>, <источник2>

// Команды с 2 и 3 операндами уже не из нотации процессора 8086, если мы захотим использовать эти возможности, то в начале программы нужно будет писать соответствующие директивы (н, .586)

IDIV, DIV (целочисленное деление со знаком)

IDIV <источник>

Результат округляется в сторону нуля, знак остатка совпадает со знаком делимого.

Он был немногословен

INC <п>, DEC <п> (инкремент, декремент)

- Увеличивает/уменьшает приемник на 1.
- В отличие от ADD, не изменяет CF³.
- OF, SF, ZF, AF, PF устанавливаются в соответствии с результатом.

NEG <п> – команда изменения знака (переводит число в доп код и прибавляет к нему единичку).

? Когда требуется поменять знак?

Например нам нужно выводить числа на экран. Для этого мы сначала должны определить знак, вывести его, а потом вывести само число.

² Как это работает, если верить интернету: когда размерность операнда составляет 8 бит, то команда **IMUL** производит умножение содержимого регистра AL на значение операнда и помещает результат в регистр AX.

Если операнд - 16-битное слово, команда **IMUL** производит умножение содержимого регистра AX на значение операнда и помещает результат в пару регистров DX:AX.

³ **CF** – флаг переноса. Возникает в случае перехода через разрядную сетку при работе с беззнаковыми числами.

DAA, DAS, AAA, AAS, AAM, AAD (десятичная арифметика)

❓ Что такое двоично-десятичные числа?

Задействуются не все возможные двоичные коды. Если число не упакованное, то из 256 возможных значений для байта будет использоваться только 10 (*что-то такое он там говорит*), то есть число – это байт от 00h до 09h. Если упакованное, то будет байт от 00h до 99h (цифры A..F не задействуются).

Если такие числа пытаться складывать или как-то там изменять, то возникают проблемы с коррекцией:

$$19h + 1 = 1Ah \Rightarrow 20h$$

(типа должны были получить 20, а получили 1А ~~аааааа~~**чего, все же правильно**, в общем перфоманс ситуации в том, что у нас числа упакованные, поэтому про буквы мы ничего не знаем, поэтому так и прыгаем)

Вердикт: команды есть, но особо никому не сдались.

DAA, DAS – команды для коррекции упакованных дд чисел после сложения/вычитания

AAA, AAS – команды для коррекции неупакованных дд чисел после сложения/вычитания

AAM, AAD – коррекция для неупакованных дд чисел после умножения/деления.

```
inc al
daa
```

AND, OR, XOR, NOT, TEST – логические команды (см. лекцию 1)

SAR, SAL, SHR, SHL, ROR, ROL, RCR, RCL – арифметический, логический, циклический сдвиг.

- SAR – shift arithmetic right, SAL – shift arithmetic left (арифметический, значит число сдвигается с учетом знака и знак (старший бит) в результате сохраняется)
- SAL тождественна SHL
- SHR зануляет старший бит, SAR – сохраняет (знак)
- ROR, ROL – циклический сдвиг вправо/влево
- RCR, RCL – циклический сдвиг через CF (задействуется не 16 бит, а 17, младший бит попадет в CF, а CF попадет в старший бит при сдвиге вправо)

BT, BTR, BTS, BTC, BSF, BSR, SETcc – операции над битами и байтами

? Что такое байт?

Минимальная адресуемая единица памяти

Работать с битами напрямую нельзя, но у нас есть чудо команды (которые нам правда тоже вряд ли понадобятся):

- BT <база>, <смещение> – считать в CF значение бита из битовой строки

? Что такое битовая строка?

Последовательность битов, которая интерпретируется как независимые биты, а не как какие-то данные (рассуждения о масках). Пример – регистр флагов.

- BTS <база>, <смещение> – установить бит в 1
- BTR <база>, <смещение> – сбросить бит в 0
- BTC <база>, <смещение> – инвертировать бит
- BSF <приемник>, <источник> – прямой поиск бита (от младшего разряда)
- BSR <приемник>, <источник> – обратный поиск бита (от старшего разряда)
- SETcc <приемник> – выставляет приемник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc.

Организация циклов

- LOOP <метка> – уменьшает CX и выполняет “короткий” переход на метку, если CX не равен 0 (128 байт в обе стороны)
- LOOPE / LOOPZ <метка> – цикл “пока равно” / “пока ноль” (флаг ZF)
- LOOPNE / LOOPNZ <метка> – цикл “пока не равно” / “пока не ноль” (флаг ZF)

Декрементируется CX и выполняется переход, если CX не ноль и если не выполняется условие (см флаг ZF)

// нужно переслушать разбор отладчика здесь

Строковые операции: копирование, сравнение, сканирование, чтение, запись

Строка источник - DS:SI, строка-приемник - ES:DI

За один раз обрабатывается один байт (слово).

MOVS / MOVSB / MOVSW <п> <и> – копирование

CMPS / CMPSB / CMPSW <п> <и> – сравнение

SCAS / SCASB / SCASW <и> – сканирование (сравнение с AL / AX)

LODS / LODSB / LODSW <и> – чтение (в AL / AX)

STOS / STOSB / STOSW <п> – запись (из AL / AX)

Исторически есть два способа хранения строк:

1. Сишный – нулевой байт в конце
2. Паскалевский – в первых двух байтах (или одном) хранится длина, а дальше идет непосредственно набор символов.

Разница в том, что во втором случае длину мы знаем заранее и нам не нужно пробегать всю строку, чтобы ее узнать, но при этом длина ограничена (а в си типа нет?). Еще один плюс паскалевского представления – нулевой байт может быть частью строки. *(Какой в ассемблере метод он не сказал..)*

? Как работают команды выше

Это довольно составные команды. Они читают с адреса источника байт (или два байта или может быть четыре байта), увеличивают индексы SI или DI на единицу (или на два, или на 4). Далее, если команды связаны с записью, то они будут записывать прочитанные байты в приемник.

! несмотря на указание <п> и <и> в командах, на самом деле в коде они прописываться не будут

Все команды работают в несколько тактов.

Если мы хотим работать со строками, а не с одним символом, то используются префиксы (пишутся в одну строчку через пробел с командой):

REP / REPE / REPZ / REPNE / REPNZ

При добавлении подобного префикса получится “цикл”, который будет выполняться столько раз, сколько выставлен CX (как в loop).

Команды с Z кроме CX будут еще контролировать состояние флага ZF.

// ну мы поняли, самим разбираться придется

Управление флагами

STC / CLC / CMC – установить / сбросить / инвертировать CF

STD / CLD – установить / сбросить DF⁴

LAHF – загрузка флагов состояния в AH

SAHF – установка флагов состояния из AH

CLI / STI – запрет⁵ / разрешение прерываний (IF)

⁴ Если $DF = 0$, то обработка строк будет идти слева направо, то есть по возрастанию адресов памяти. Если же выставить его в 1, то команды обработки строк будут идти в обратную сторону (то есть, например, они будут копировать данные из источника в приемник по уменьшению адресов от старшего к младшему, то есть справа налево)

⁵ Программные прерывания блокировать глупо (мы их сами вызываем), а вот аппаратные прерывания блокировать (нажатие клавиатуры, прерывание таймера) имеет смысл (на время критичных системных операций)

Загрузка сегментных регистров

LDS <п>, <и> – загрузить адрес, используя DS

LES <п>, <и> – загрузить адрес, используя ES

LFS <п>, <и> – загрузить адрес, используя FS

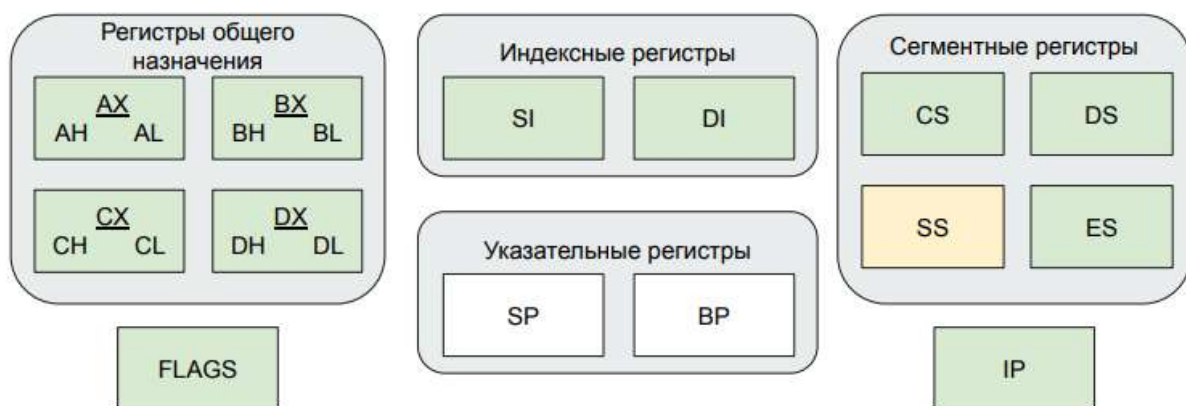
LGS <п>, <и> – загрузить адрес, используя GS

LSS <п>, <и> – загрузить адрес, используя SS

(все сегментные регистры, кроме CS, так как регистр с кодом напрямую менять нельзя)

<п> – регистр, <и> – переменная

Стек



SP – Stack Pointer (меняется автоматически процессором, но руками его тоже можно поменять)

BP – Base Pointer (вспомогательный регистр, используется программистом или компилятором)

Стек – структура данных, которая работает по принципу первым пришел, последним ушел (LIFO / FILO).

Сегмент стека – область памяти программы, используемая ее подпрограммами, а также (вынужденно) обработчиками прерываний.

SP – указатель на вершину стека

В x86 стек растет вниз, в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента.

? Почему так сделали

Чтобы проще было отследить переполнение стека (если адрес дошел до 0, значит мы заполнили все) => экономия регистра (не используем доп регистр для хранения размера стека).

Команды непосредственной работы со стеком

PUSH <и> – поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP

POP <п> – считать данные из стека. Считывает значения адреса SS:SP и увеличивает SP на величину приемника.

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 <операнд> – вызов процедуры

- Сохраняет адрес следующей команды в стеке (уменьшает SP [в случае ближнего перехода на 2 байта, в случае дальнего на 4] и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передает управление на значение аргумента

RET / RETN / RETF <число> – возврат из процедуры

- Загружает из стека адрес возврата, увеличивает SP
- `retn` – ближний возврат (2 байта), `retf` – дальний возврат (4 байта)
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

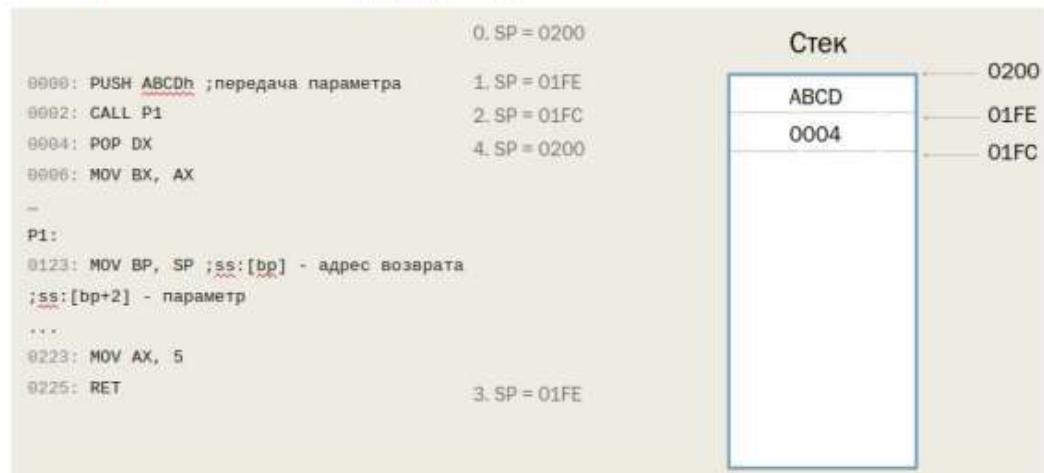
BP – base pointer

- Используется в подпрограмме для сохранения “начального” значения SP (типа чтобы в подпрограмме можно было писать что-то в стек, а по возвращении можно было вернуть состояние стека)
- Адресация параметров (то, что положила в стек вызывающая программа)
- Адресация локальных переменных (глобальные переменные обычно валяются в сегменте данных, локальные же там хранить нельзя, так как в этом случае при каждом вызове они будут перезаписываться)

Пример вызова подпрограммы №1



Пример вызова подпрограммы №2



Пример вызова подпрограммы №3



Соглашения о вызовах – как передавать параметры (стек или регистры), как возвращать результат (по си возврат идет через ax), способы освобождения памяти от параметров.

Стековый кадр – 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

Соединила лекции и данные из Репы Леши

1. Регистры общего назначения.

Регистры — специальные ячейки памяти, находящиеся физически внутри процессора, доступ к которым осуществляется не по адресам, а по именам. Поэтому, работают очень быстро.

Существуют регистры, которые могут использоваться без ограничений, для любых целей — **регистры общего назначения**.

В 8086 регистры 16 битные.

При использовании регистров общего назначения, можно обратиться к каждым 8 битам (байту) по-отдельности, используя вместо *X - *H или *L

Верхние 8 бит (1 байт) — AH, BH, CH, DH

Нижние 8 бит (1 байт) — AL, BL, CL, DL

- **AX** часто используется для хранения результата действий, выполняемых над двумя операндами. Например, используется при MUL и DIV (умножение и деление) ($AX = AL * \text{ЧИСЛО}$ - при mul если число == байт, $AL = AX / \text{ЧИСЛО}$, при div если число == байт)
- **BX** - базовый регистр в вычислениях адреса, часто указывает на начальный адрес (называемый базой) структуры в памяти;
- **CX** - счетчик циклов, определяет количество повторов некоторой операции;
- **DX** - определение адреса ввода/вывода, так же может содержать данные, передаваемые для обработки в подпрограммы.
- **SI (индекс источника) и DI (индекс приемника)** - Ещё есть два этих регистра, они называются индексными, то есть используются для индексации в массивах / матрицах и т.д. (другие регистры (кроме BX и BP) не будут там работать (на 8086)).
 - Могут использоваться в большинстве команд, как регистры общего назначения.
 - В этих регистрах нельзя обратиться к каждому из байтов по-отдельности

2. Сегментные регистры. Адресация в реальном режиме.

- Сегмент кода - регистр CS. Командой MOV изменить невозможно, меняется автоматически по мере выполнения команд.
- Сегмент данных. Основной регистр - DS, при необходимости дополнительных сегментов данных задействуются ES, FS, GS.
- Сегмент стека - регистр SS

Мы знаем что регистр IP "указывает" на следующую команду, мы также знаем что регистры у нас размером в 16 бит, таким образом, используя один регистр программист имеет доступ только к 2^{16} адресам, это примерно 64 кБ. Это достаточно мало, поэтому адресация в 8086 по 2^{20} адресам, то есть примерно 1 МБ памяти. Для этого используют **адрес начала сегмента и смещение**. Сегментные регистры как раз хранят адрес. Реальный адрес высчитывается так: сегментная часть $\times 16 + \text{смещение}$. (умножение на 16 = сдвиг на четыре бита влево)

При такой адресации адреса 0400h:0001h и 0000h:4001h будут ссылаться на одну и ту же ячейку памяти, так как $400h \times 16 + 1 = 0 \times 16 + 4001h$.



3. Понятие сегментной части адреса и смещения.

[SEG]:[OFFSET] => физический адрес:

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

5678h:1234h =>

$$\begin{array}{r} 56780 \\ + 1234 \\ \hline 579B4 \end{array}$$

4. Регистры работы со стеком.

Стек - структура данных, работающая по принципу LIFO (last in, first out) - последним пришёл, первым вышел.

Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний.

SP - указатель на вершину стека, **BP** - указатель на начало стека. BP используется в подпрограмме для сохранения "начального" значения SP, адресации параметров и локальных переменных.

В x86 стек *растет вниз*, в сторону уменьшения адресов. При запуске программы **SP** указывает на конец сегмента.

Команды непосредственной работы со стеком

- **PUSH <источник>**
 - Помещает данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- **POP <приемник>**
 - Считывает данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- **PUSHA**
 - Помещает в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.

- **POPA**
 - Загружает регистры из стека (SP игнорируется).

Вызов процедуры и возврат из процедуры

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

5. Структурная программа. Сегменты.

Любая программа состоит из сегментов

Виды сегментов:

- Сегмент кода
- Сегмент данных
- Сегмент стека

Описание сегмента в исходном коде:

имя SEGMENT READONLY *выравнивание тип разряд 'класс'*

...

имя ENDS

- Выравнивание - расположение начала сегмента с адреса, кратного какому-либо значению. Варианты:
 - BYTE,
 - WORD (2 байта),

- DWORD (4 байта),
- PARA (16 байт, по умолчанию),
- PAGE (256 байт).
- Тип:
 - PUBLIC (сегменты с одним именем объединятся в один) ----- заставляет компоновщик соединить все сегменты с одинаковым именем. Новый объединенный сегмент будет целым и непрерывным. Все адреса (смещения) объектов, а это могут быть, в зависимости от типа сегмента, команды или данные, будут вычисляться относительно начала этого нового сегмента;
 - STACK (для стека) ----- определение сегмента стека. Заставляет компоновщик соединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра SS. Комбинированный тип STACK (стек) аналогичен комбинированному типу PUBLIC, за исключением того, что регистр SS является стандартным сегментным регистром для сегментов стека. Регистр SP устанавливается на конец объединенного сегмента стека. Если не указано ни одного сегмента стека, компоновщик выдаст предупреждение, что стековый сегмент не найден. Если сегмент стека создан, а комбинированный тип STACK не используется, программист должен явно загрузить в регистр SS адрес сегмента (подобно тому, как это делается для регистра DS);
 - COMMON (сегменты будут “наложены” друг на друга по одним и тем же адресам памяти); ----- располагает все сегменты с одним и тем же именем по одному адресу. Все сегменты с данным именем будут перекрываться и совместно использовать память. Размер полученного в результате сегмента будет равен размеру самого большого сегмента;
 - AT <начало> - расположение по фиксированному физическому адресу, параметр - сегментная часть этого адреса ----- располагает сегмент по абсолютному адресу параграфа (параграф — объем памяти, кратный 16, поэтому последняя шестнадцатеричная цифра адреса параграфа равна 0). Абсолютный адрес параграфа задается выражением xxxx. Компоновщик располагает сегмент по заданному адресу памяти (это можно использовать, например, для доступа к видеопамати или области ПЗУ), учитывая атрибут комбинирования. Физически это означает, что сегмент при загрузке в память будет расположен, начиная с этого абсолютного адреса параграфа, но для доступа к нему в соответствующий сегментный регистр должно быть загружено заданное в атрибуте значение. Все метки и адреса в определенном таким образом сегменте отсчитываются относительно заданного абсолютного адреса;

```

DS2 SEGMENT AT 0b800h
    CA LABEL byte
    ORG 80 * 2 * 2 + 2 * 2
    SYMB LABEL word
DS2 ENDS

```

- PRIVATE - вариант по умолчанию. ----- сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля.
- Класс - метка, позволяющая объединить сегменты (расположить в памяти друг за другом).

пример

```

; Сегмент данных
) DATAS SEGMENT PARA PUBLIC 'DATA'
    LIMIT2 DW 1
    LIMIT16 DW 15
    SIGNBINNUMBER DB 16 DUP('0'), '$'
    UNSIGNHEXNUMBER DB 4 DUP('0'), '$'
    SIGNFORHEX DB ' '
) DATAS ENDS

```

Директива ASSUME

ASSUME *регистр* : *имя сегмента*

- Не является командой
- Нужна для контроля компилятором правильности обращения к переменным

Модель памяти

.model модель, язык, модификатор

- TINY - один сегмент на всё
- SMALL - код в одном сегменте, данные и стек - в другом
- COMPACT - допустимо несколько сегментов данных
- MEDIUM - код в нескольких сегментах, данные - в одном
- LARGE, HUGE
- Язык - C, PASCAL, BASIC, SYSCALL, STDCALL. Для связывания с ЯВУ и вызова подпрограмм.
- Модификатор - NEARSTACK/FARSTACK

Определение модели позволяет использовать сокращённые формы директив определения сегментов.

Конец программы и точка входа

END start

- start - имя метки, объявленной в сегменте кода и указывающее на команду, с которой начнётся исполнение программы.
- Если в программе несколько модулей, только один может содержать начальный адрес.

6. Прерывание 21h. Примеры ввода-вывода.

- Аналог системного вызова в современных ОС.
- Используется наподобие вызова подпрограммы.
- Номер функции передается через AH.

Еще одним важным случаем, когда нам требуется прерывание DOS - завершение программы. Чтобы ассемблер перестал читать подряд строки кода нам требуется положить в ah код DOS завершения программы (04Ch) и вызвать прерывание. Код завершения программы (ошибка или нет) кладется и берется из al.

функция	назначение	вход	выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом \$	-
01	Считать символ из stdin с эхом	-	AL – ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL=FF	AL – ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL – ASCII-код символа

08	Считать символ без эха	-	AL – ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры - AL=0, если клавиша не была нажата, и FF, если была		
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	-

```

INT 21h
MOV AH, 4Ch
INT 21h

```

завершение, код можно поместить в регистр AL

```

; Ввода символа
INPUTSYM:
MOV AH, 1
INT 21h
RET

```

```

; Вывод сообщения
OUTPUTMSG:
MOV AH, 9
INT 21h
RET

```

7. Стек. Назначение. Примеры использования.

Стек работает по правилу LIFO / FILO (последним пришёл, первым вышел)

Сегмент стека — область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний.

Используется для временного хранения переменных, передачи параметров для подпрограмм, адрес возврата при вызове процедур и прерываний.

Регистр SP — указывает на вершину стека

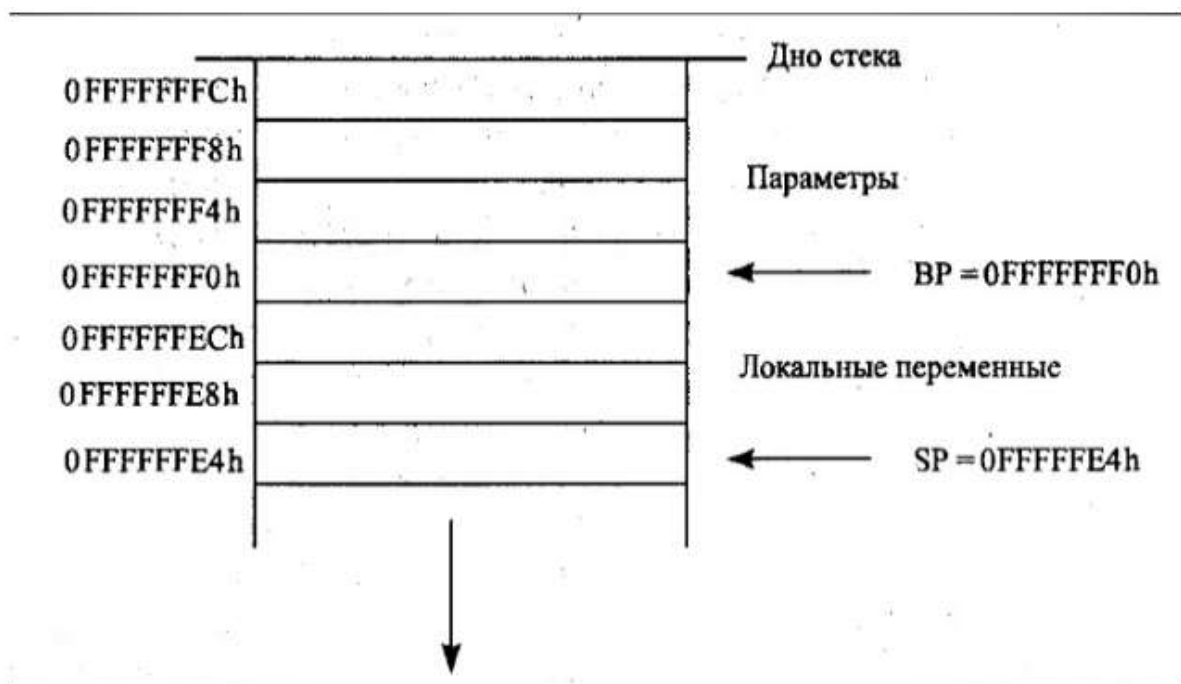
В x86 стек "растет вниз", в сторону уменьшения адресов (от максимально возможного адреса). При запуске программы SP указывает на конец сегмента.

BP (Base Pointer)

Используется в подпрограмме для сохранения "начального" значения SP.

Так же, используется для адресации параметров и локальных переменных.

При вызове подпрограммы параметры кладут на стек, а в BP кладут текущее значение SP. Если программа использует стек для хранения локальных переменных, SP изменится и таким образом можно будет считывать переменные напрямую из стека (их смещения запишутся как BP + номер параметра)



Команды непосредственной работы со стеком

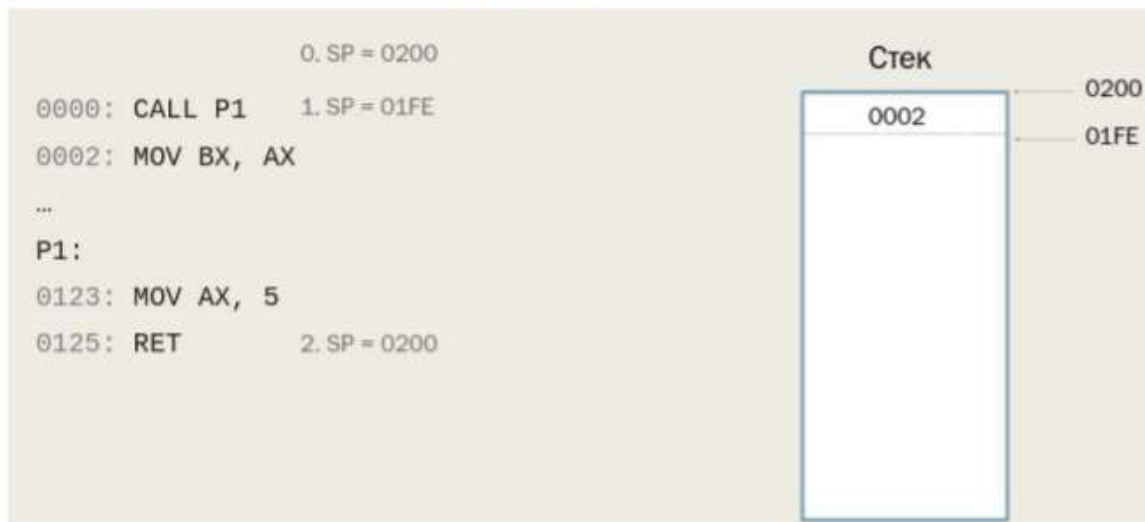
- **PUSH <источник>**
 - Помещает данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- **POP <приемник>**
 - Считывает данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- **PUSHA**
 - Помещает в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- **POPA**

- Загружает регистры из стека (SP игнорируется).

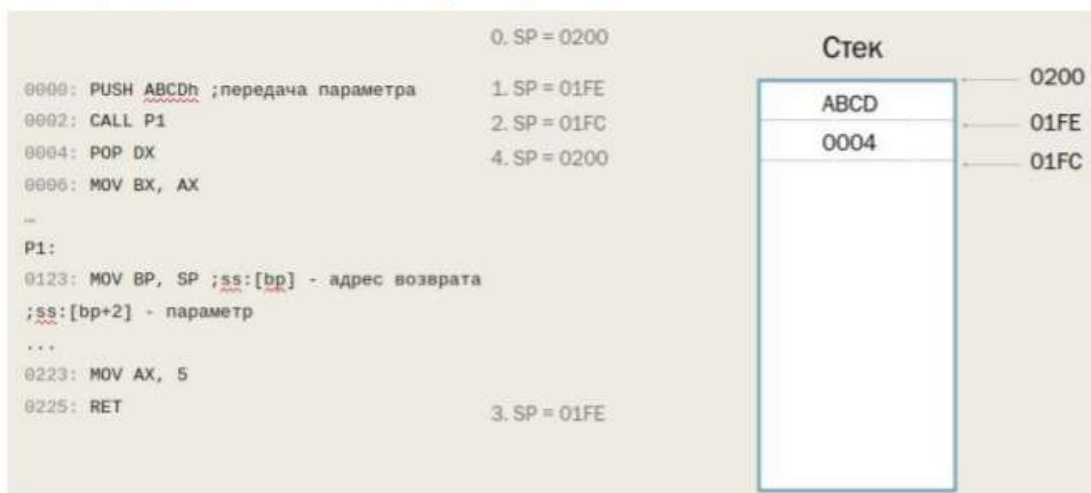
CALL и RET

- CALL <операнд> — передает управление на адрес <операнд>
 - Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- RET <число> — загружает из стека адрес возврата, увеличивая SP.
 - Если указать операнд, то можно очистить стек для очистки стека от параметров (<число> будет прибавлено к SP)

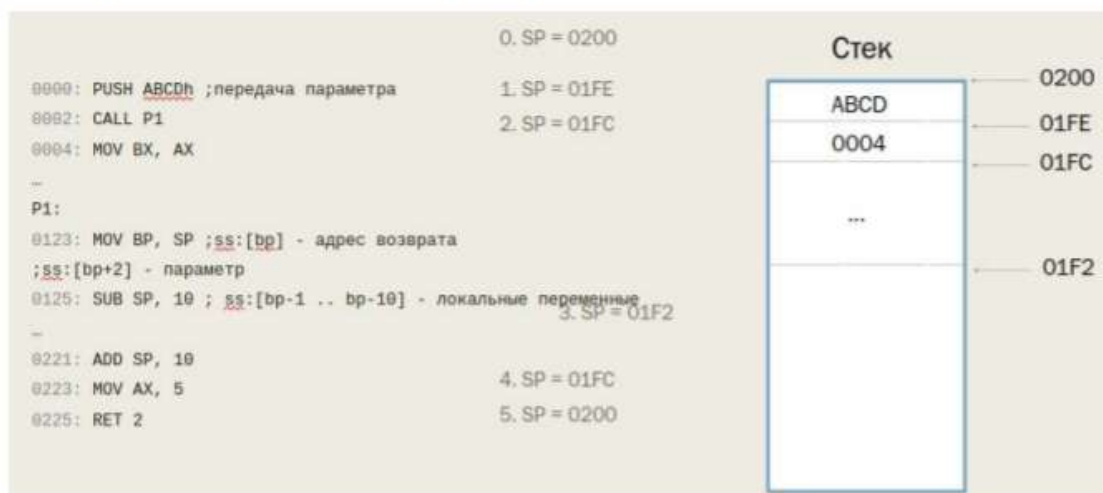
Пример вызова подпрограммы №1



Пример вызова подпрограммы №2



Пример вызова подпрограммы №3



8. Регистры флагов.

Флаги выставляются при операциях, но не обязательно все сразу. Например INC и DEC не затрагивают флаг CF, в отличие от ADD и SUB.

Также есть команды рассчитанные на флаги, например CMP, которая выставляет флаги такие, как если бы произошло вычитание аргументов.

Как мы помним, регистры у нас размером в 16 бит.

Регистр FLAGS

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	-

- 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
- TF (trap flag) - флаг трассировки
- IF (interrupt enable flag) - флаг разрешения прерываний
- DF (direction flag) - флаг направления
- OF (overflow flag) - флаг переполнения
- IOPL (I/O privilege flag) - уровень приоритета ввода-вывода
- NT (nested task) - флаг вложенности задач

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	-	

зеленый DF - флаг управления

черные - флаги состояния

синие - системные флаги

CF TF IF DF - можно изменять командами

- **CF (carry flag)** - флаг переноса - устанавливается в 1, если результат предыдущей операции не уместился в приемник и произошел перенос или если требуется заем при вычитании. Иначе 0.
- **PF (parity flag)** - флаг четности - устанавливается в 1, если младший байт результата предыдущей операции содержит четное количество единиц.
- **AF (auxiliary carry flag)** - вспомогательный флаг переноса - устанавливается в 1, если в результате предыдущей операции произошел перенос из 3 в 4 или заем из 4 в 3 биты.
- **ZF (zero flag)** - флаг нуля - устанавливается в 1, если результат предыдущей команды равен 0.
- **SF (sign flag)** - флаг знака - всегда равен старшему биту результата.
- **TF (trap flag)** - флаг трассировки - предусмотрен для работы отладчиков в пошаговом режиме. Если поставить в 1, после каждой команды будет происходить передача управления отладчику.
- **IF (interrupt enable flag)** - флаг разрешения прерываний - если 0 процессор перестает обрабатывать прерывания от внешних устройств.
- **DF (direction flag)** - флаг направления - контролирует поведение команд обработки строк. Если 0, строки обрабатываются слева направо, если 1 справа налево.
- **OF (overflowflag)** - флаг переполнения - устанавливается в 1, если результат предыдущей операции над числами со знаком выходит за допустимые для них пределы.
- **IOPPL (I/O privilege level)** - уровень приоритета ввода-вывода - а это на 286, на не нужно пока.
- **NT (nested task)** - флаг вложенности задач - а это на 286, на не нужно пока.

9. Команды условной и безусловной передачи управления.

Условный переход - переход, происходящий при выполнении какого-то условия.

Безусловный переход - переход, не зависящий от чего-либо (совершаемый в любом случае).

- **Виды безусловных переходов**
 - JMP - оператор безусловного перехода.
 - Для короткого и ближнего переходов непосредственный операнд (число) прибавляется к IP. Регистры и переменные заменяют старое значение в IP (CS:IP).

Вид перехода	Дистанция перехода
short (короткий)	-128..+127 байт
near (ближний)	в том же сегменте (без изменения CS)
far (дальний)	в другой сегмент (со сменой CS)

- **Виды условных переходов**

- *Не зависящие от знака*

Команда	Описание	Состояние флагов для выполнения 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 = 0
JNE/JNZ	Если не равно/если не ноль	ZF = 0
JP/JPE	Есть четность/четное	PF = 1
JNP/JPO	Нет четности/нечетное	PF = 0

JCXZ	CX = 0	
------	--------	--

○ **Беззнаковые**

<i>Команда</i>	<i>Описание</i>	<i>Состояние флагов для выполнения перехода</i>	<i>Знаковый</i>
<i>JB/JNAE/JC</i>	<i>Если ниже/если не выше и не равно/если перенос</i>	<i>CF = 1</i>	<i>нет</i>
<i>JNB/JAE/JNC</i>	<i>Если не ниже/если выше и равно/если перенос</i>	<i>CF = 0</i>	<i>нет</i>
<i>JBE/JNA</i>	<i>Если ниже или равно/если не выше</i>	<i>CF = 1 или ZF = 1</i>	<i>нет</i>
<i>JB/JNAE/JC</i>	<i>Если ниже/если не выше и не равно/если перенос</i>	<i>CF = 1</i>	<i>нет</i>
<i>JA/JNBE</i>	<i>Если выше/если не ниже и не равно</i>	<i>CF = 0 и ZF = 0</i>	<i>нет</i>

○ **Знаковые**

<i>Команда</i>	<i>Описание</i>	<i>Состояние флагов для выполнения перехода</i>	<i>Знаковый</i>
----------------	-----------------	---	-----------------

JL/JNGE	Если меньше/если не больше и не равно	SF != OF	да
JGE/JNL	Если больше или равно/если не меньше	SF = OF	да
JLE/JNG	Если меньше или равно/если не больше	ZF = 1 или SF != OF	да
JG/JNLE	Если больше/если не меньше и не равно	ZF = 0 и SF = OF	да

- **Команды, выставяющие флаги и использующиеся при переходах к передаче управления**

- **CMP <приемник>, <источник>**

- Источник - число, регистр или переменная.
- Приемник - регистр или переменная; не может быть переменной одновременно с источником.
- Вычитает источник из приёмника, результат никуда не сохраняется, выставяются флаги CF, PF, AF, ZF, SF, OF.
- *Возможно, если равны числа, то ZF == 1, если первое < второго, то CF == 1*

- **TEST <приемник>, <источник>**

- Аналог AND, но результат не сохраняется. Выставяются флаги SF, ZF, PF.

Инструкция AND выполняет логическое И между всеми битами двух операндов. Результат записывается в первый операнд. Синтаксис:

AND ЧИСЛО1, ЧИСЛО2

ЧИСЛО1 может быть одним из следующих:

- Область памяти (MEM)
- Регистр общего назначения (REG)

ЧИСЛО2 может быть одним из следующих:

- Область памяти (MEM)
- Регистр общего назначения (REG)
- Непосредственное значение (IMM)

()

10. Организация многомодульных программ.

Как и на других языках программирования, программа на ассемблере может состоять из нескольких файлов - модулей. При компиляции (трансляции) каждый модуль превращается в объектный файл, далее при компоновке объектные файлы соединяются в единый исполняемый модуль.

Модули обычно состоят из описания сегментов будущей программы с помощью директивы SEGMENT.

Пример:

```
имя SEGMENT [READONLY] выравнивание тип разряд 'класс'  
...  
имя ENDS
```

Параметры:

- • Выравнивание - расположение начала сегмента с адреса, кратного какому-либо значению. Варианты:
 - BYTE;
 - WORD (2 байта);
 - DWORD (4 байта);
 - PARA (16 байт, по умолчанию);
 - PAGE (256 байт).
- • Тип:
 - PUBLIC (сегменты с одним именем объединятся в один);
 - STACK (для стека); COMMON (сегменты будут “наложены” друг на друга по одним и тем же адресам памяти);
 - AT <начало> - расположение по фиксированному физическому адресу, параметр - сегментная часть этого адреса;
 - PRIVATE - вариант по умолчанию.
- • Класс - метка, позволяющая объединить сегменты (расположить в памяти друг за другом).

11. Подпрограммы. Объявления, вызов.

Описание подпрограммы

```
имя_подпрограммы PROC [NEAR | FAR] ; по умолчанию NEAR, если не указать  
; тело подпрограммы;  
    ret [кол-во используемых локальных переменных] ; ничего не указывается,  
если не использовались локальные переменные на стеке  
имя_подпрограммы ENDP
```

Вызов подпрограммы

; вызов любой (в плане расстояния) подпрограммы

```
call имя_подпрограммы
```

CALL - вызов процедуры, RET - возврат из процедуры

- CALL
 - 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/RETN/RETF
 - Загружает из стека адрес возврата, увеличивает SP
 - Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров
 - Отличие RETN и RETF в том, что 1ая команда делает возврат при ближнем переходе, 2ая - при дальнем (различие в кол-ве байт, считываемых из стека при возврате). Если используется RET, то ассемблер сам выберет между RETN и RETF в зависимости от описания подпрограммы (процедуры).
- BP – base pointer
 - Используется в подпрограмме для сохранения "начального" значения SP
 - Адресация параметров
 - Адресация локальных переменных

12. Арифметические команды.

ADD и ADC

ADD <приемник>, <источник> — сложение. Не делает различий между знаковыми и беззнаковыми числами.

ADC <приемник>, <источник> — сложение с переносом. Складывает приёмник, источник и флаг CF.

SUB и SBB

SUB <приемник>, <источник> — вычитание. Не делает различий между знаковыми и беззнаковыми числами.

SBB <приемник>, <источник> — вычитание с займом. Вычитает из приёмника источник и дополнительно - флаг CF.

Флаг CF можно рассматривать как дополнительный бит у результата.

$$11111111_2 + 00000001_2 = (1)00000000_2 \text{ (флаг установлен)}$$

Можно использовать ADC и SBB для сложения вычитания и больших чисел, которые по частям храним в двух регистрах.

Пример: Сложим два 32-битных числа. Пусть одно из них хранится в паре регистров DX:AX (младшее двойное слово - DX, старшее AX). Другое в паре BX:CX

```
add ax, cx
```

```
adc dx, bx
```

Если при сложении двойных слов произошел перенос из старшего разряда, то это будет учтено командой adc.

Эти 4 команды (ADD, ADC, SUB, SBB) меняют флаги: CF, OF, SF, ZF, AF, PF

MUL и IMUL

MUL <источник> — выполняет умножение чисел без знака. <источник> не может быть число (нельзя: MUL 228). Умножает регистр AX (AL), на <источник>. Результат остается в AX, либо DX:AX, если не помещается в AX.

IMUL — умножение чисел со знаком.

1. IMUL <источник>. Работает так же, как и MUL
2. IMUL <приёмник>, <источник>. Умножает источник на приемник, результат в приемник.
3. IMUL <приёмник>, <источник1>, <источник2>. Умножает источник1 на источник2, результат в приёмник.

Флаги: OF, CF

DIV и IDIV

DIV <источник> — выполняет деление чисел без знака. <источник> не может быть число (нельзя: DIV 228). Делимое должно быть помещено в AX (или DX:AX, если делитель больше байта). В первом случае частное в AL, остаток в AH, во втором случае частное в AX, остаток в DX.

IDIV <источник> — деление чисел со знаком. Работает так же как и DIV. Округление в сторону нуля, знак остатка совпадает со знаком делимого.

INC, DEC, NOT

INC <приемник> — увеличивает примник на 1.

DEC <приемник> — уменьшает примник на 1.

Меняют флаги: OF, SF, ZF, AF, PF

NEG <применик> — меняет знак приемника.

13. Команды побитовых операций.

Операции над битами и байтами

- **BT <база>, <смещение>**
 - Считывает в CF значение бита из битовой строчки.
- **BTS <база>, <смещение>**
 - Устанавливает бит в 1.
- **BTR <база>, <смещение>**
 - Сбрасывает бит в 0.
- **BTC <база>, <смещение>**
 - Инвертирует бит.
- **BSF <приемник>, <смещение>**
 - Прямой поиск бита (от младшего разряда).
- **BSR <приемник>, <смещение>**
 - Обратный поиск бита (от старшего разряда).
- **SETcc <приемник>**
 - Выставляет приемник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc.

jcc метка Проверка состояния флагов в зависимости от кода операции (оно отражает проверяемое условие):

 - если проверяемое условие истинно, то перейти к ячейке, обозначенной операндом;
 - если проверяемое условие ложно, то передать управление следующей команде.

Логический, арифметический, циклический сдвиг

- **SAL (SHL)**
 - Арифметический сдвиг влево.
- **SHR**
 - Арифметический сдвиг направо, зануляет старший бит.

- **SAR**
 - Арифметический сдвиг направо, сохраняет знак.
- **ROR (ROL)**
 - Циклический сдвиг вправо (влево).
- **RCR (RCL)**
 - Циклический сдвиг вправо (влево) через CF.

14. Команды работы со строками.

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

За один раз обрабатывается один байт (слово).

- **MOVS / MOVSB / MOVSW <приёмник>, <источник>** - копирование

Алгоритм работы

- выполнить копирование байта, слова или двойного слова из операнда источника в операнд приемник, при этом адреса элементов предварительно должны быть загружены:
 - адрес источника — в пару регистров ds:esi/si (ds по умолчанию, допускается замена сегмента);
 - адрес приемника — в пару регистров es:edi/di (замена сегмента не допускается);
- в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
 - если df=0, то увеличить содержимое этих регистров на длину структурного элемента последовательности;
 - если df=1, то уменьшить содержимое этих регистров на длину структурного элемента последовательности;
- если есть префикс повторения, то выполнить определяемые им действия (см. команду rep).

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

Алгоритм работы:

- выполнить вычитание элементов (источник - приемник), адреса элементов предварительно должны быть загружены:
 - адрес источника — в пару регистров ds:esi/si;
 - адрес назначения — в пару регистров es:edi/di;
- в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
 - если df=0, то увеличить содержимое этих регистров на длину элемента последовательности;
 - если df=1, то уменьшить содержимое этих регистров на длину элемента последовательности;
- в зависимости от результата вычитания установить флаги:
 - если очередные элементы цепочек не равны, то cf=1, zf=0;
 - если очередные элементы цепочек или цепочки в целом равны, то cf=0, zf=1;
- при наличии префикса выполнить определяемые им действия (см. команды rep/repne).

- SCAS / SCASB / SCASW <приёмник> - сканирование (сравнение с AL/AX (в зависимости от размеров приемника))

Алгоритм работы:

- выполнить вычитание (элемент цепочки-(eax/ax/al)). Элемент цепочки локализуется парой es:edi/di. Замена сегмента es не допускается;
- по результату вычитания установить флаги;
- изменить значение регистра edi/di на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:
 - df=0 — величина положительная, то есть просмотр от начала цепочки к ее концу;
 - df=1 — величина отрицательная, то есть просмотр от конца цепочки к ее началу.
- LODS / LODSB / LODSW <источник> - ЧТЕНИЕ (В AL/AX)

Алгоритм работы:

- загрузить элемент из ячейки памяти, адресуемой парой ds:esi/si, в регистр al/ax/eax. Размер элемента определяется неявно (для команды lods) или явно в соответствии с применяемой командой (для команд lodsb, lodsw, lodsd);
- изменить значение регистра si на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:
 - df=0 — значение положительное, то есть просмотр от начала цепочки к ее концу;
 - df=1 — значение отрицательное, то есть просмотр от конца цепочки к ее началу.

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

Алгоритм работы:

- записать элемент из регистра al/ax/eax в ячейку памяти, адресуемую парой es:di/edi. Размер элемента определяется неявно (для команды stos) или конкретной применяемой командой (для команд stosb, stosw, stosd);
- изменить значение регистра di на величину, равную длине элемента цепочки. Знак этого изменения зависит от состояния флага df:
 - df=0 — увеличить, что означает просмотр от начала цепочки к ее концу;
 - df=1 — уменьшить, что означает просмотр от конца цепочки к ее началу.
- Префиксы: REP / REPE / REPZ / REPNE / REPNZ
- REP - повторить следующую строковую операцию
- REPE - повторить следующую строковую операцию, если равно
- REPZ - Повторить следующую строковую операцию, если нуль
- REPNE - повторить следующую строковую операцию, если не равно
- REPNZ - повторить следующую строковую операцию, если не нуль

Префиксы REP (F3h), REPE (F3h) и REPNE (F2h) применяются со строковыми операциями. Каждый префикс заставляет строковую команду, которая следует за ним, повторяться указанное в регистре счетчика (ECX) (в случае нашей

модели процессора 8086 - cx) количество раз или, кроме этого, (для префиксов REPE и REPNE) пока не встретится указанное условие во флаге ZF.

Пример использования: REP LODS AX

Мнемоники REPZ и REPNZ являются синонимами префиксов REPE и REPNE соответственно и имеют одинаковые с ними коды. Префиксы REP и REPE / REPZ также имеют одинаковый код F3h, конкретный тип префикса задается неявно той командой, перед которой он применен.

Все описываемые префиксы могут применяться только к одной строковой команде за один раз. Чтобы повторить блок команд, используется команда LOOP или другие циклические конструкции.

Затрагиваемые флаги: OF, DF, IF, TF, SF, ZF, AF, PF, CF

15. Прерывания. Обработка прерываний.

Прерывание означает временное прекращение основного процесса вычислений для выполнения некоторых запланированных или незапланированных действий, вызванных работой устройств или программы.

В зависимости от источника различают прерывания:

- **Аппаратные** (внешние) – реакция процессора на физический сигнал от некоторого устройства. Возникают в случайные моменты времени, а значит – асинхронные
- **Программные** (внутренние) – возникает в заранее запланированный момент времени — синхронные
- **Исключения** – разновидность программных прерываний, реакция процессора на некоторую не стандартную ситуацию возникшую во время выполнения команды;

Вектор прерываний – адрес процедуры обработки прерываний. Адреса размещаются в специальной области памяти доступной для всех подпрограмм. Вектор прерывания одержит 4 байта: старшее слово содержит сегментную составляющую адреса процедуры обработки исключения, младшее — смещение.

Вызов прерывания осуществляется с помощью директивы **INT номер_прерывания**.

00h – 1Fh – прерывания BIOS

20h – 3Fh – прерывания DOS

40h – 5Fh – зарезервировано

60h – 7Fh – прерывания пользователя

80h – FFh – прерывания Бейсика

Порядок выполнения INT:

- 1) В стек помещается содержимое регистра флагов FLAGS
- 2) Флаги трассировки и прерывания устанавливаются в нуль (TF = 0, IF = 0)
- 3) Вычисляется адрес соответствующего вектора прерываний – 4*номер_прерывания
- 4) Содержимое сегментного регистра CS помещается в стек

5) Содержимое второго слова вектора прерываний заносится в CS

6) Содержимое управляющего регистра IP заносится в стек

7) Первое слово вектора прерываний помещается в IP

Процедура используемая для обработки прерывания должна быть дальней (FAR). Возврат с прерывания осуществляется при помощи директивы **IRET**, которая восстанавливает CS:IP и FLAGS. Все параметры в процедуру и из нее передаются через регистры, какое – зависит от прерывания.

- 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

Прерывание DOS - вывод на экран в текстовом режиме

Функция	Назначение	Вход	Выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом \$	-

Прерывание DOS - ввод с клавиатуры

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL - ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL = FF	AL - ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL - ASCII-код символа
08	Считать символ без эха	-	AL - ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры	-	AL=0, если клавиша не была нажата, и FF, если была
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	

16. Работа с портами ввода-вывода.

Порты ввода-вывода

- Порты ввода-вывода - механизм 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

РК 2020 ГОДА

1. Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 1A29h:37B4h

Вычисляем по схеме

1) сег побитового сдвигаем на 4 разряда влево / умножить на 16

2) Прибавить offset

ответ 1DA44h

[SEG]:[OFFSET] => физический адрес:

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

5678h:1234h =>

56780

⬇1234

579B4

2. Какова разрядность регистра IP?
разрядность всех регистров по 16
ответ 16
3. Какая формулировка соответствует характеристикам короткого перехода?

Виды **переходов** (передачи управления)

Короткий (short) - в пределах адресов -128..+127 от текущего значения IP (1 байт).

Ближний (near) - в пределах того же сегмента (2 байта).

Дальний (far) - на произвольный адрес (4 байта).

ответ Метка занимает 1 байт, переход допустим в диапазоне от -128 до 127 байт от текущего значения IP

4. Какая команда выполняет переход, соответствующий условию "больше"?

ответ JG

Команда	Описание	Состояние флагов для выполнения перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнение	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	Состояние флагов для выполнения перехода	Знаковый
JB JNAE JC	Если ниже Если не выше и не равно Если перенос	CF = 1	нет
JNB JAE JNC	Если не ниже Если выше или равно Если нет переноса	CF = 0	нет
JBE JNA	Если ниже или равно Если не выше	CF = 1 или ZF = 1	нет
JA JNBE	Если выше Если не ниже и не равно	CF = 0 и ZF = 0	нет

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JL JNGE	Если меньше Если не больше и не равно	SF <> OF	да
JGE JNL	Если больше или равно Если не меньше	SF = OF	да
JLE JNG	Если меньше или равно Если не больше	ZF = 1 или SF <> OF	да
JG JNLE	Если больше Если не меньше и не равно	ZF = 0 и SF = OF	да

5. Имеется описание следующего сегмента:

DS SEGMENT

ORG 100h

I DW 0

A DB 1

DS ENDS

Чему равно OFFSET A?

Команда ORG сдвигает на заданное кол-во байт память, и все после нее будет записано с таким смещением.

ответ 102h

6. В каких командах языка ассемблера применяется сегментный префикс?

ответ При работе со значениями переменных

Сегментный префикс.

Pandas edited this page on 14 Jun 2017 · 1 revision

Конструкция вида

```
AS:X
```

где AS — какойто сегментный регистр (DS, ES, SS, CS), а X какая-то метка (ближний адрес) или другой регистр (SI, DI, SP, IP). Явно указывает, какой регистр мы используем для получения номера сегмента, к которому мы обращаемся.

Н-р: `mov ax, es:x`

7. Какая операция с сегментным регистром недопустима?

ответ Загрузка константы

8. `mov ax, [bx][si]+10`

ответ $bx+si+10$

9. Что такое неупакованное двоично-десятичное число?

ответ Десятичная цифра, хранящаяся в байте

Неупакованный двоично-десятичный тип — байтовое представление десятичной цифры от 0 до 9. **Неупакованные десятичные числа** хранятся как байтовые значения без знака по одной цифре в каждом байте.

10. Что такое упакованное двоично-десятичное число?

ответ Две десятичные цифры, хранящиеся в полубайтах одного байта

Десятичные числа — специальный вид представления числовой информации, в основу которого положен принцип кодирования каждой десятичной цифры числа группой из четырех бит. При этом каждый байт числа содержит одну или две десятичные цифры в так называемом двоично-десятичном коде (BCD — Binary-Coded Decimal). Микропроцессор хранит BCD-числа в двух форматах (рис. 3):

- упакованном формате — в этом формате каждый байт содержит две десятичные цифры. Десятичная цифра представляет собой двоичное значение в диапазоне от 0 до 9 размером 4 бита. При этом код старшей цифры числа занимает старшие 4 бита. Следовательно, диапазон представления десятичного упакованного числа в одном байте составляет от 00 до 99;
- неупакованном формате — в этом формате каждый байт содержит одну десятичную цифру в четырех младших битах. Старшие четыре бита имеют нулевое значение. Это так называемая зона. Следовательно, диапазон представления десятичного неупакованного числа в одном байте составляет от 0 до 9.

11. Какая команда осуществляет циклический сдвиг битов вправо

Логический, арифметический, циклический сдвиг. SAR, SAL, SHR, SHL, ROR, ROL, RCR, RCL

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

SAR - арифметический сдвиг вправо

SAL - арифметический сдвиг влево

SHR - логический сдвиг вправо

SHL - логический сдвиг влево

ROR - циклический сдвиг вправо

ROL - циклический сдвиг влево

RCR - циклический сдвиг вправо через перенос

RCL - циклический сдвиг влево через перенос

12. Что делает команда TEST?

Команда **TEST**

TEST <приёмник>, <источник>

- Аналог AND, но результат не сохраняется
- Выставляются флаги SF, ZF, PF

ответ Побитовое И без сохранения результата

13. Укажите, на какую величину уменьшится указатель вершины стека после выполнения следующих команд:

PUSH AX ; - 2б

PUSH BX ; - 2б

POP CX ; + 2б

CALL F ; подпрограмма объявлена в том же сегменте -2б

=> уменьшится на 4

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)

14. Укажите, на какую величину изменится указатель вершины стека после выполнения следующей команды:

RETN 4

CALL - вызов процедуры, RET - возврат из 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/**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

ответ на 6, так как 2 байта - адрес возврата, и еще 4 - операнд команды.

RETN 4, то ответ будет 8, так как 4 байта возврат, так как длинный переход

15. укажите, какая характеристика соответствует команде NEG

NEG - изменение знака

NEG <приёмник>

Переводит число в дополнительный код.