1. Архитектура фон Неймана, принципы фон Неймана.

Архитектура фон Неймана

- Процессор состоит из блоков УУ и АЛУ
- УУ - дискретный конечный автомат.
  Структурно состоит из: дешифратора команд (операций), регистра команд, узла вычислений текущего исполнительного адреса, счётчика команд (регистр IP).
- АЛУ - под управлением УУ производит преобразование над данными (операндами). Разрядность операнда - длина машинного слова. (Машинное слово - машинно-зависимая величина, измеряемая в битах, равная разрядности регистров/шины данных)

Принципы фон Неймана

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

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

Каждая программа состоит из отдельных машинных команд; команда является указанием процессору произвести какую-либо элементарную операцию, например, копирования информации, сложение.

Выполнение программы

1. Определение формата файла (.COM или .EXE, в случае 8086)
2. Чтение и разбор заголовка
3. Считывание разделов исполняемого модуля (файла) в ОЗУ по необходимым адресам.
4. Подготовка к запуску, если требуется. (установка регистров; настройка окружения, загрузка библиотек (см. 1 ЛР, 2 часть))
5. Передача управления на точку входа.

Дальше выполняются инструкции заданные в самой программе.

Машинный код

Машинный код - набор команд, который напрямую интерпертируется процессором.
Каждая машинная инструкция выполняет определенное действие.

Исполняемые файлы

Исполняемый файл — файл, содержащий программу в виде, в котором она может быть исполнена компьютером.
Стадии получения: компиляция + линковка (компоновка).
Компилятор - программа для преобразования исходного текста другой программы на определенном ЯП в объектный модуль.
Линковщик (компоновщик) - связывает несколько объектных файлов в исполняемый файл

Язык ассемблера

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

3. Виды памяти ЭВМ. Запуск и исполнение программы.

Виды памяти

Байт - минимальная адресуемая единица памяти (8 бит).
Машинное слово - машинно-зависимая величина, измеряемая в битах, равная разрядности регистров/шины данных.
Параграф - 16 байт
Память делится на внешнюю и внутреннюю
К внутренней памяти относится:

- ОЗУ (оперативное запоминающее устройство)
- ПЗУ (постоянное запоминающее устройство). В ПЗУ хранится информация, которая записывается туда при изготовлении ЭВМ. Важнейшая микросхема ПЗУ - BIOS.

Выполнение программы

1. Определение формата файла (.COM или .EXE, в случае 8086)
2. Чтение и разбор заголовка

3. Считывание разделов исполняемого модуля (файла) в ОЗУ по необходимым адресам.
4. Подготовка к запуску, если требуется. (установка регистров; настройка окружения, загрузка библиотек (см. 1 ЛР, 2 часть))
5. Передача управления на точку входа.
Дальше выполняются инструкции заданные в самой программе.
ЗАПУСК ЧЕРЕЗ ИСПОЛНЯЕМЫЙ ФАЙЛ

Исполняемые файлы

Исполняемый файл — файл, содержащий программу в виде, в котором она может быть исполнена компьютером.
Стадии получения: компиляция + линковка (компоновка).
Компилятор - программа для преобразования исходного текста другой программы на определенном ЯП в объектный модуль.
Линковщик (компоновщик) - связывает несколько объектных файлов в исполняемый файл

4. Сегментная модель памяти в архитектуре 8086.

Шина адреса в 8086

- Шина адреса - 20 бит, что позволяет адресовать 2^20 (1мб) памяти, а не 2^16 б.
- Шина данных - 16 бит.

Сегментная модель памяти

Архитектура 8086 имеет четыре сегментных регистра (см. вопрос №3).
Логический адрес записывают как сегмент:смещение (и те, и те в 16 с/с). В реальном режиме для вычисления физического адреса, адрес из сегмента сдвигают влево на 4 разряда (можно сказать, что просто приписывают 0 в конце или умножают на 16) и добавляют смещение. Например, логический адрес 7522:F139 дает физический адрес 84359.
На шину передается именно физический адрес. Если результат больше, чем 2^20 - 1, то 21 бит отбрасывают.
Такой режим работы процессора называют реальным режимом адресации процессора
При такой адресации адреса 0400h:0001h и 0000h:4001h будут ссылаться на одну и ту же ячейку памяти, так как 400h × 16 + 1 = 0 × 16 + 4001h.

5. Процессор 8086. Регистры общего назначения.

Разрядность

Разрядность: 16 бит.

Регистры

Всего в 8086 14 регистров.

Регистры общего назначения: AX, BX, CX, DX.

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 обратится 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 * ЧИСЛО - при mul если число == байт, AL = AX / 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

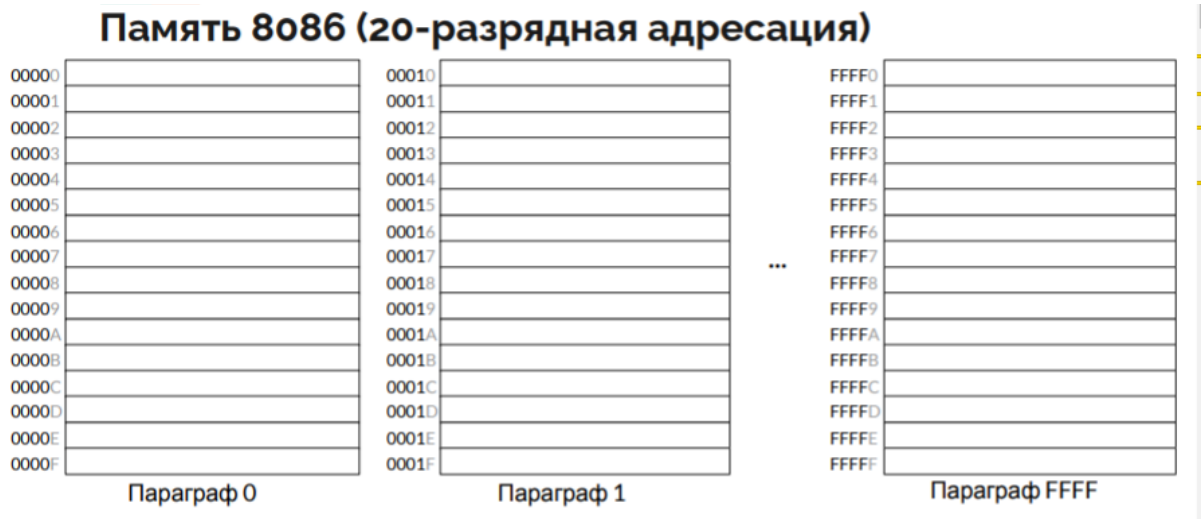
6. Процессор 8086. Сегментные регистры. Адресация в реальном режиме. Понятие сегментной части адреса и смещения.

Сегментные регистры: CS (code segment), SS (stack segment), DS (data segment), ES (extra segment).

Каждый сегментный регистр определяет адрес начала сегмента в памяти, при этом сегменты могут совпадать или пересекаться. По умолчанию регистр CS используется при выборке инструкций, регистр SS при выполнении операций со стеком, регистры DS и ES при обращении к данным.

- 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 × 16 + 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 × 16 + 1 = 0 × 16 + 4001h.



[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
```

7. Процессор 8086. Регистр флагов.

Флаг переноса CF

Команда CLC сбрасывает флаг CF.
Команда STC устанавливает флаг CF в единицу.
Команда CMC инвертирует значение флага CF.

Флаг направления DF

Этот флаг определяет направление обработки данных цепочечными командами (о них подробно расскажу в отдельной статье). Он должен устанавливаться или сбрасываться перед использованием этих команд.
Команда CLD сбрасывает флаг DF.
Команда STD устанавливает флаг DF в единицу.

Флаг прерывания IF

Этот флаг определяет, разрешены в данный момент прерывания или нет (о прерываниях тоже будет отдельная статья).
Команда CLI сбрасывает флаг IF (запрещает прерывания).
Команда STI устанавливает флаг IF в единицу (разрешает прерывания).

Команды LAHF и SAHF

Команда LAHF загружает младший байт регистра флагов в AH. Её удобно использовать, когда нужно получить значения сразу нескольких флагов. Порядок расположения флагов представлен на рисунке:

7	6	5	4	3	2	1	0
SF	ZF	0	AF	0	PF	1	CF

Команда SAHF выполняет обратную операцию – загружает содержимое AH в младший байт регистра флагов. Это позволяет одновременно изменить значения нескольких флагов. При этом биты 1, 3, 5 регистра AH игнорируются.

Регистры флагов в 8086.

Флаги - выставляются при выполнение операций, в основном арифметических. С помощью этих флагов можно определить что-нибудь определить, например было ли переполнение при последней выполненной операции.
Каждый флаг представляет собою 1 бит, выставляемый в 0 (флаг сброшен) или в 1 (флаг установлен). Не существует специальных команд, позволяющих обратится к этому регистру напрямую.
Хотя разрядность регистра FLAGS 16 бит, реально используют не все 16. Остальные были зарезервированны при разработке процессора, но так и не были использованны.
Вот за что отвечает каждый бит в регистре 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	IOPL	NT	-

- **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 чётности - 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 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.
- **IOPL (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.

Регистр флагов содержит группу флагов состояния, управляющий флаг и группу системных флагов[1]:

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как `ADD`, `SUB`, `MUL`, `DIV`.

Флаг направления (DF, бит 10 в регистре флагов) управляет строковыми инструкциями (`MOVS` , `CMPS` , `SCAS` , `LODS` и `STOS`): установка флага заставляет уменьшать адреса (обрабатывать строки от старших адресов к младшим), обнуление заставляет адреса увеличивать. Инструкции `STD` и `CLD` соответственно устанавливают и обнуляют флаг DF.

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

8. Команды пересылки данных.

MOV <приемник>, <источник>.

- Приемник: РОН (регистр общего назначения), сегментный регистр, переменная (то есть ячейка памяти)
- Источник: непосредственный операнд (например, число), РОН, сегментный регистр, переменная

Нельзя загрузить в сегментный регистр значение непосредственно из памяти. Поэтому для этого используют промежуточный регистр (в начале лабы всегда так делали).
Переменные не могут быть одновременно и источником, и приемником.

XCHG <операнд1>, <операнд2>

Обмен операндов между собой. Выполнятеся либо над двумя регистрами, либо регистр + переменная.

9. Команда сравнения.

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, выставляются флаги CF, PF, AF, ZF, SF, OF.

TEST <приемник>, <источник>

Аналог AND, но результат не сохраняется. Выставляются флаги SF, ZF, PF.
Можно использовать для проверки на ноль, например TEST bx, bx

10. 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.

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)

Для короткого и ближнего переходов непосредственный операнд (число) прибавляется к IP. Регистры и переменные заменяют старое значение в IP (CS:IP).

Команда безусловной передачи управления JMP

JMP <операнд>

- Передаёт управление в другую точку программы, не сохраняя какой-либо информации для возврата.
- Операнд - непосредственный адрес, регистр или переменная.

Команды условных переходов J.. (Зубков, Assembler, …, глава 2)

- Переход типа short или near
- Обычно используются в паре с CMP
- "Выше" и "ниже" - при сравнении беззнаковых чисел
- "Больше" и "меньше" - при сравнении чисел со знаком

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***

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 и равно/если перенос***	***CF = 0***	***нет***
JBE/JNA	***Если ниже или равно/если не выше***	***CF = 1 или ZF = 1***	***нет***
JB/JNAE/ JC	***Если ниже/если не выше и не равно/если перенос***	***CF = 1***	***нет***
JA/JNBE	***Если выше/если не ниже и не равно***	***CF = 0 и ZF = 0***	***нет***

- ***Знаковые***

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

11. 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.

$11111111_2 + 00000001_2 = (1)00000000_2$ (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.

12. Двоично-десятичная арифметика.

Десятичная арифметика DAA, DAS, AAA, AAS, AAM, AAD

Это ваще жесть какая-то...
Упакованный BCD-формат - это упакованное двоично-десятичное число - байт от 00h до 99h (цифры A..F не задействуются).

DAA

Команда DAA (Decimal Adjust AL after Addition) позволяет получать результат сложения упакованных двоично-десятичных данных в таком же упакованном BCD-формате. То есть она корректирует после сложения, пример:

```
mov AL,71H   ; AL = 0x71h
add AL,44H   ; AL = 0x71h + 0x44h = 0xB5h
daa          ; AL = 0x15h
             ; CF = 1 - перенос является частью результата 71 + 44 = 115
```

DAS

Команда DAS (Decimal Adjust AL after Subtraction) позволяет получать результат вычитания упакованных двоично-десятичных данных в таком же упакованном BCD-формате. То есть она корректирует после вычитания, пример:

```
mov AL,71H   ; AL = 0x71h
sub AL,44H   ; AL = 0x71h - 0x44h = 0x2Dh
das          ; AL = 0x27h
             ; CF = 0 - заем (перенос) является частью результата
```

ASCII-формат - это неупакованное двоично-десятичное число (байт от 00h до 09h).

AAA

Команда AAA (ASCII Adjust After Addition) позволяет преобразовать результат сложения двоично-десятичных данных в ASCII-формат. Для этого команда AAA должна выполняться после команды двоичного сложения ADD, которая помещает однобайтный результат в регистр AL. Если будет перенос, он запишется в AH.

Для того, чтобы преобразовать содержимое регистра AL к ASCII-формату, необходимо после команды AAA выполнить команду OR AL,0x30h (то есть сделать читаемым числом). Пример...:

```
sub AH,AH    ; очистка AH
mov AL,'6'   ; AL = 0x36h
add AL,'8'   ; AL = 0x36h + 0x38h = 0x6Eh
aaa          ; AX = 0x0104h
or AL,30H    ; AL = 0x34h = '4'
```

AAS

Команда AAS (ASCII Adjust After Subtraction) позволяет преобразовать результат вычитания двоично-десятичных данных в ASCII-формат. Для этого команда AAS должна выполняться после команды двоичного вычитания SUB, которая помещает однобайтный результат в регистр AL. Если был заем, будет вычитание 1 из AH.
Для того, чтобы преобразовать содержимое регистра AL к ASCII-формату, необходимо после команды AAS выполнить команду OR AL,0x30h (то есть сделать читаемым числом).
При положительном результате вычитания это выглядит следующим образом:

```
sub AH,AH    ; очистка AH
mov AL,'9'   ; AL = 0x39h
sub AL,'3'   ; AL = 0x39h - 0x33h = 0x06h
aas          ; AX = 0x0006h
or AL,30H    ; AL = 0x36h = '6'
```

при вычитании с получением результа меньше нуля:

```
sub AH,AH    ; очистка AH
mov AL,'3'   ; AL = 0x33h
sub AL,'9'   ; AL = 0x33h - 0x39h = 0xFAh
aas          ; AX = 0xFF04h
or AL,30H    ; AL = 0x34h = '4' (хз почему)
```

AAM
AAM imm8 (imm8 - система счисления aka ib ? вроде)

Команда AAM (ASCII Adjust AX After Multiply) позволяет преобразовать результат умножения неупакованных двоично-десятичных данных в ASCII-формат. Для этого команда AAM должна выполняться после команды беззнакового умножения MUL (но не после команды умножения со знаком IMUL), которая помещает двухбайтный результат в регистр AX. Команда AAM распаковывает результат умножения, содержащийся в регистре AL, деля его на второй байт кода операции ib (равный 0x0Ah для

безоперандной мнемоники AAM). Частное от деления (наиболее значащая цифра результата) помещается в регистр AH, а остаток (наименее значащая цифра результата) — в регистр AL .
Для того, чтобы преобразовать содержимое регистра AX к ASCII-формату, необходимо после команды AAM выполнить команду OR AX,0x3030h.
Пример:

```
mov AL,3     ; множимое в формате неупакованного BCD
помещается в регистр AL
mov BL,9     ; множитель в формате неупакованного BCD
помещается в регистр BL
mul BL       ; AX = 0x03 * 0x09 = 0x001Bh
aam          ; AX = 0x0207h
or AX,3030H  ; AX = 0x3237h, т.е. AH = '2', AL = '7'
```

AAD
AAD imm8

Команда AAD (ASCII Adjust AX Before Division) используется для подготовки двух разрядов неупакованных BCD-цифр (наименее значащая цифра в регистре AL, наиболее значащая цифра в регистре AH) для операции деления DIV, которая возвращает неупакованный BCD-результат.
Команда AAD устанавливает регистр AL в значение AL = AL + (imm8 * AH) , где imm8 – это второй байт кода операции ib (равный 0x0Ah для безоперандной мнемоники AAD), с последующей очисткой регистра AH.
После команды AAD регистр AX будет равен двоичному эквиваленту оригинального неупакованного двухзначного числа.
Пример:

```
mov AX,0207H   ; делимое в формате неупакованного BCD
помещается в регистр AX
mov BL,05H     ; делитель в формате неупакованного BCD
помещается в регистр BL
aad            ; AX = 0x001Bh
div BL         ; AX = 0x0205h
or AL,30H      ; AL = 0x35h = '5'
```

13. Команды побитовых операций. Логические команды.

- AND <приёмник>, <источник> - побитовое "И"
- OR <приёмник>, <источник> - побитовое "ИЛИ"
- XOR <приёмник>, <источник> - побитовое исключающее "ИЛИ"
- NOT <приёмник> - инверсия
- SHL <приёмник>, <счётчик> - сдвиг влево (SAL - эквивалентная команда)
- SHR <приёмник>, <счётчик> - сдвиг вправо

- SAR <приёмник>, <счётчик> - сдвиг вправо с сохранением знакового бита
- ROR <приёмник>, <счётчик> - циклический сдвиг вправо
- RCR <приёмник>, <счётчик> - циклический сдвиг вправо, через флаг переноса
- ROL <приёмник>, <счётчик> - циклический сдвиг влево
- RCL <приёмник>, <счётчик> - циклический сдвиг влево, через флаг переноса

Все эти команды меняют регистр FLAGS.

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>

Копирует байт или слово из приемника в источник. После выполнения команды, регистры SI и DI увеличиваются на 1 (или 2), если флаг DF = 0, или уменьшаются на 1 (или 2), если DF = 1.

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

Сравнивает байт или слово из приемника с источником. После выполнения команды, регистры SI и DI увеличиваются на 1 (или 2), если флаг DF = 0, или уменьшаются на 1 (или 2), если DF = 1.

SCAS/SCASB/SCASW <приёмник>

Сканирование (сравнение с AL/AX). После выполнения команды, регистры SI и DI увеличиваются на 1 (или 2), если флаг DF = 0, или уменьшаются на 1 (или 2), если DF = 1.

LODS/LODSB/LODSW <источник>

Чтение (в AL/AX). После выполнения команды, регистры SI и DI увеличиваются на 1 (или 2), если флаг DF = 0, или уменьшаются на 1 (или 2), если DF = 1.

STOS/STOSB/STOSW <приёмник>

Запись (из AL/AX). После выполнения команды, регистры SI и DI увеличиваются на 1 (или 2), если флаг DF = 0, или уменьшаются на 1 (или 2), если DF = 1.

В каждой команде источник и приемник можно опустить.

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, REPE и REPNE применяются со строковыми операциями. Каждый префикс заставляет строковую команду, которая следует за ним, повторяться указанное в регистре счетчике (E)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, конкретный тип префикса задается неявно той командой, перед которой он применен.
Все описываемые префиксы могут применяются 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. Команда трансляции по таблице.

XLAT <адрес>

XLATB

Помещает в AL байт из таблицы по адресу DS:BX со смещением относительно начала таблицы, равным AL. Адрес, указанный в исходном коде, не обрабатывается компилятором и служит в качестве комментария. Если в адресе явно указан сегментный регистр, он будет использоваться вместо DS.
Короче говоря, XLATB -> AL = DS:[(E)BX + AL]

Описание:
Команда XLAT, используя индекс из регистра AL, выбирает элемент из таблицы, расположенной в памяти по адресу DS:(E)BX, и помещает его в AL. Регистр AL должен содержать беззнаковый индекс элемента в таблице. Сама таблица адресуется регистровой парой DS:BX (для атрибута размера адреса равного 16 бит), или регистровой парой DS:EBX (для атрибута размера адреса равного 32 бита).

Для операнда команды XLAT допускается замещение сегмента по умолчанию.

Мнемоника XLATB является синонимом XLAT и используется для краткости, а также для того, чтобы показать размерность операции выборки (байт).

16. Команда вычисления эффективного адреса.

LEA <приёмник>, <источник>

Вычисляет эффективный адрес источника и помещает его в приёмник. Позволяет вычислить адрес, описанный сложным методом адресации (да и просто его загрузить).
Иногда используется для быстрых арифметических вычислений:
lea bx, [bx+bx*4]
lea bx, [ax+12]
Эти вычисления занимают меньше памяти, чем соответствующие MOV и ADD, и не изменяют флаги.

17. Структура программы на языке ассемблера. Модули. 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 по умолчанию - параграф (PARA)
- Тип по умолчанию - PRIVATE
- Класс - любая метка, взятая в одинарные кавычки. Все сегменты с одинаковым классом, будут расположенным друг за другом (в исполнимом файле, даже PRIVATE)

Структура программы на ассемблере (Зубков С. В., Assembler для DOS, Windows, …, глава 3):

- Модули (файлы исходного кода)
- Сегменты (описание блоков памяти)
- Составляющие программного кода:
  - команды процессора
  - инструкции описания структуры, выделения памяти, макроопределения
- Формат строки программы:
  - метка команда/директива операнды ; комментарий

Директива SEGMENT

Каждая программа, написанная на любом языке программирования, состоит из одного или нескольких сегментов. Обычно область памяти, в которой находятся команды, называют сегментом кода, область памяти с данными - сегментом данных и область памяти, отведённую под стек, - сегментом стека.

Выравнивание:

- **BYTE**
- **WORD**
- **DWORD**
- **PARA** (по умолчанию)
- **PAGE**

Тип:

- **PUBLIC** - 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** - 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** - 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** - 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 видеопамяти 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;
- **PRIVATE** (по умолчанию) - сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля.

Класс:
Это любая метка, взятая в одинарные кавычки. Сегменты одного класса расположатся в памяти друг за другом.

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.

18. Подпрограммы. Объявление, вызов.

Описание подпрограммы

```
имя_подпрограммы PROC [NEAR | FAR] ; по умолчанию NEAR, если не указать
    ;тело подпрограммы;
    ret [кол-во используемых локальный 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.

19. Подпрограммы. Возврат управления.

- 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 описание подпрограммы (процедуры).
- BP – base pointer
    - Используется в подпрограмме для сохранения "начального" значения SP
    - Адресация параметров
    - Адресация локальных переменных

20. Макроопределения.

Макроопределения. Назначение.

Макроопределение (макрос) - именованный участок программы, который ассемблируется каждый раз, когда его имя встречается в тексте программы.
Роль макросов в ассемблере такая же, как макросов в си. Очень гибкий и мощный инструмент, чтобы писать код общего вида, который во время работы препроцессора будет заменяться на конкретные выражения.
Из методички:

Определения:
Макроопределение - специальным образом оформленная последовательность предложений языка ассемблера, под управлением которой ассемблер (точнее, его часть, называемая макрогенератором или препроцессором) порождает макрорасширения макрокоманд.
Макрорасширение - последовательность предложений языка ассемблера (обыкновенных директив и команд), порождаемая

макрогенератором при обработке макрокоманды под управлением макроопределения и вставляемая в исходный текст программы вместо макрокоманды.
Макрокоманда (или макровызов) - предложение в исходном тексте программы, которое воспринимается макрогенератором как команда (приказ), предписывающая построить макрорасширение и вставить его на ее место.
В макрокоманде могут присутствовать параметры, если они были описаны в макроопределении.
Макроопределение без параметров однозначно определяет текст макрорасширения.
Макроопределение с параметрами описывает множество (возможно, очень большое) возможных макрорасширений, а параметры, указанные в макрокоманде, сужают это множество до одного единственного макрорасширения.
Определение макроса в программе:
имя MACRO параметры
...
ENDM

Пример:
load_reg MACRO register1, register2
push register1
pop register2
ENDM

Сравнение макросов с подпрограммами

Плюсы:
Так как текст макрорасширения вставляется на место макрокоманды, то нет затрат времени, как для подпрограмм, на подготовку параметров, передачу управления и выполнение других работ при выполнении программы
Минусы:
При многочисленных вызовах МО (макроопределения) разрастается объем модуля программы,
Фактические значения параметров макрокоманд должны быть известны препроцессору или могли быть вычислены им (нельзя использовать в качестве фактического параметра МО значения переменных или регистров, так как они могут быть известны только при выполнении программы).
Замечания.

Имена формальных параметров МО-й локализованы в них, т.е. вне определения могут использоваться для обозначения других объектов.
Число формальных параметров ограничено лишь длиной строки, обрабатываемой ассемблером.
МО-я должны предшествовать обращениям к ним.
Нет ограничений, кроме физических, на число предложений в теле МО.
В листинге предложениям макрорасширений предшествуют ЦБЗ, указывающие глубину их вложения в макроопределениях.

Макроопределения. Директивы присваивания и отождествления.

Директива присваивания =

Директива присваивания служит для создания целочисленной макропеременной или изменения её значения и имеет формат:
Макроимя = Макровыражение
Макровыражение (или Константное выражение) - выражение, вычисляемое препроцессором, которое может включать целочисленные константы, макроимена, вызовы макрофункций, знаки операций и круглые скобки, результатом вычисления которого является целое число
Операции:

- арифметические (+, -, *, /. MOD)
- логические
- сдвигов
- отношения

Директивы отождествления EQU, TEXTEQU

Директива для представления текста и чисел:
Макроимя EQU нечисловой текст и не макроимя ЛИБО число
Макроимя EQU <Операнд>
Макроимя TEXTEQU Операнд
Пример:

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

Макроопределения. Макрооперации.

% - вычисление выражение перед представлением числа в символьной форме (пример ниже - из методички)

```
MP_REC MACRO P
   MOV AX,P
IF P
```

```
    MP_REC  %(P-1) ;;перед записью в MPасш-ние вычислить
P-1
ENDIF
    ENDM
```

<> - подстановка текста без изменений (полезно, когда есть вероятность пересечения имени какого-либо макроса с простым (или не очень) текстом, который мы хотим вставить)
& - склейка текста (A&B ==> AB, если параметры - макропараметры, то они склеятся)
! - считать следующий символ текстом, а не знаком операции (A!&B ==> A&B)
;; - исключение строки из макроса (После препроцессора эта строчка исчезнет (если одна ";", то комментарий остается); Дословно из методички: "текст не выносится в макрорасширение")

Макроопределения. Блоки повторения.

REPT

Повтор фиксированное число раз
REPT <число>
...
ENDM

IRP или FOR (конкретное имя зависит от компилятора)

Подстановка фактических параметров по списку на место формального
IRP form,<fact_1[,fact_2,...]>
...
ENDM

IRPC или FORC (конкретное имя зависит от компилятора)

Подстановка символов строки на место формального параметра
IRPC form,fact
...
ENDM

WHILE

Классический цикл while
WHILE cond
...
ENDM

Примеры (взяты из методички)

REPT

Резервирование 3-х байтов с начальными значениями 0, 3, 6

```
A=0
MB0 LABEL BYTE
REPT 3
   DB A
   A=A+3
ENDM
```

IRP (FOR)

Определение переменных A0, A1, A2, A3 с начальными значениями 0,1,2,3 соответственно

```
IRP X,<0,1,2,3> ;;параметры - числа
   A&X DB X
ENDM
```

IRPC (FORC)

Описание переменных полей данных с начальными значениями 'A', 'B', 'C' соответственно

```
IRPC X,<"ABC">
   DB '&X&'
ENDM
```

WHILE (что-то очень загроможденное, без изменений взятое из методички. здесь нас больше интересует то, как в макросе можно менять параметр и проверять условие)

Стандартные макрофункция @SubStr и директива SubStr могут порождать множество подстрок типа text с числовыми и нечисловыми значениями, причём при одних и тех же значениях параметров директива SubStr определит (переопределит) макропеременную типа text, а макрофункция @SubStr вернёт значение, совпадающее со значением макропеременной. Следующий вложенный цикл позволяет перебрать и вывести значения подмножеств строки 1234

```
j=1
while j LE 4
        i=1
        WHILE i le 5-j
                names SubStr <ABCD>,i,j
                %ECHO 'names SubStr <ABCD>,i,j' out: names , i, j
                %ECHO '@SubStr (ABCD,i,j)' out: @SubStr (ABCD,i,j)
                i=i+1
```

```
        endM
        j=j+1
endm
```

Макроопределения. Директивы условного ассемблирования.

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

IFB <par> - истинно, если параметр не определён (то есть фактический параметр par не был задан в МКоманде)
IFNB <par> - истинно, если параметр определён
IFIDN <s1>,<s2> - истинно, если строки совпадают
IFDIF <s1>,<s2> - истинно, если строки разные
IFDEF/IFNDEF <name> - истинно, если имя объявлено/не объявлено

Макроопределения. Директивы управления листингом
Листинг - файл, формируемый компилятором и содержащий текст ассемблерной программы, список определённых меток, перекрёстных ссылок и сегментов.
TITLE (только 1 раз), SUBTTL - заголовок, подзаголовок на каждой странице
PAGE высота, ширина
NAME - имя программы
.LALL - включение полных макрорасширений, кроме ;;
.XALL - по умолчанию
.SALL - не выводить тексты макрорасширений
.NOLIST - прекратить вывод листинга

Комментарии

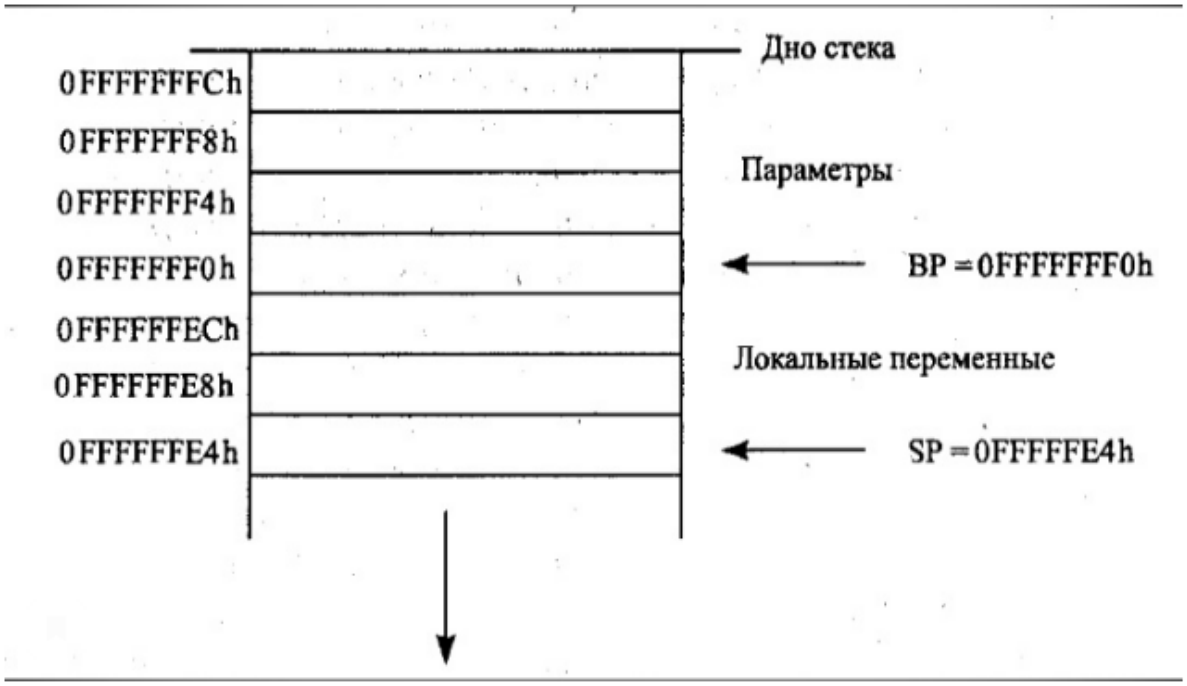
```
comment @
… многострочный текст ...
@
```

21. Стек. Аппаратная поддержка вызова подпрограмм.

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.**
BP (Base Pointer)

Используется в подпрограмме для сохранения "начального" значения SP. Так же, используется для адресации параметров и локальных переменых. 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).

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. Передает 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.

22. Прерывания. Обработка прерываний в реальном режиме работы процессора.

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	

23. Процессор 80386. Режимы работы. Регистры.

Ну а теперь настало время поговорить о режимах работы МП i80386. Микропроцессор i80386 может работать в трёх режимах: реальный режим, защищённый режим, виртуальный режим.

Реальный режим (эмуляция i8086) — это режим, имеющийся во всех процессорах Intel. Используется преимущественно с целью установки процессора для работы в защищенном режиме, а также с целью выполнения программ микропроцессоров предыдущих поколений [4]. При подаче сигнала сброса или при включении питания процессор всегда начинает работу в реальном режиме. В реальном режиме МП i80386 имеет такую же базовую архитектуру, что и МП i8086, но обеспечивает доступ к 32-разрядным регистрам. В реальном режиме МП 80386 для получения физического адреса использует тот же механизм что и процессор i8086. Этот механизм получения адреса в микропроцессоре i80386 был назван реальным режимом адресации. В реальном режиме размеры памяти и обработка прерываний МП i38086 полностью совпадают с аналогичными функциями МП i8086.

Единственным способом выхода из реального режима является явное переключение в защищённый режим, которое производится установкой специального флага в одном из системных регистров [4].

Защищённый режим позволяет полностью использовать все архитектурные возможности микропроцессора, но не позволяет выполнять программы, разработанные для i8086. С точки зрения программиста, защищённый режим, по сравнению с реальным, предоставляет большее адресное пространство и поддерживает новый механизм адресации. Это позволяет делать более гибкими используемые методы программирования и выполнять более крупные программы и программные пакеты.

Режим виртуального i8086. Переход в этот режим возможен, если процессор уже находится в защищённом режиме. В виртуальном режиме, в отличие от защищённого, возможна работа программ реального режима.

В этом лабораторном цикле мы не будем подробно рассматривать защищённый и виртуальный режимы.

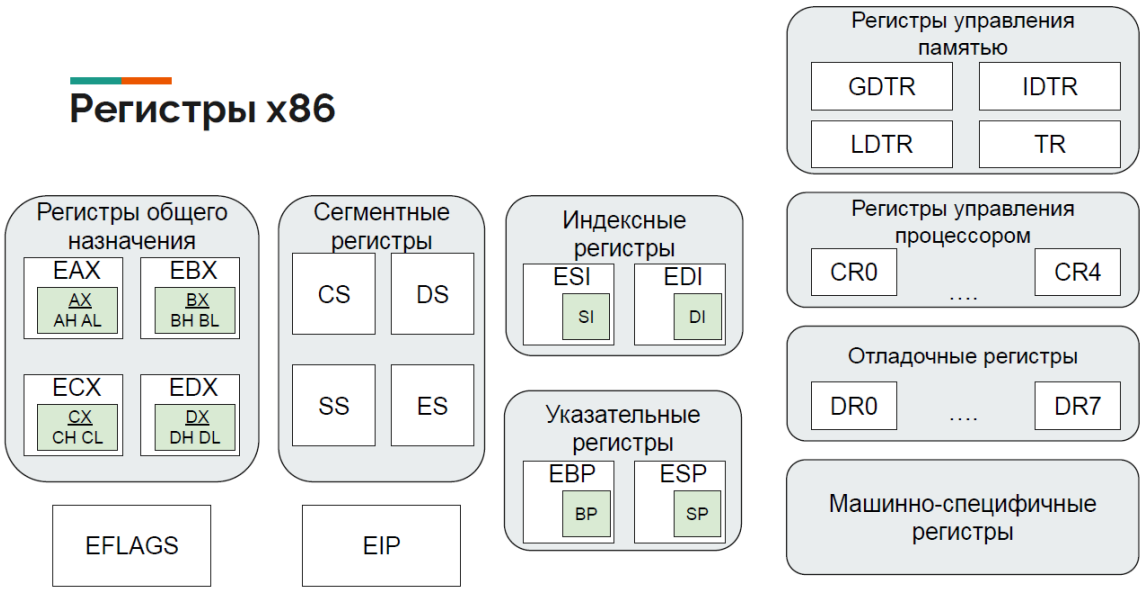
Наши лабораторные работы мы будем проводить только в реальном режиме работы микропроцессора.

32-разрядные:

- регистры, кроме сегментных
- шина данных
- шина адреса (2^32 = 4ГБ ОЗУ)

Регистры

EDX = Extended DX (обращение к частям остается (DH, DL))
Добавлены регистры поддержки работы в защищенном режиме (обеспечивание разделения доступа программ между собой, между программами и ОС и тд; эти регистры справа на картинке)

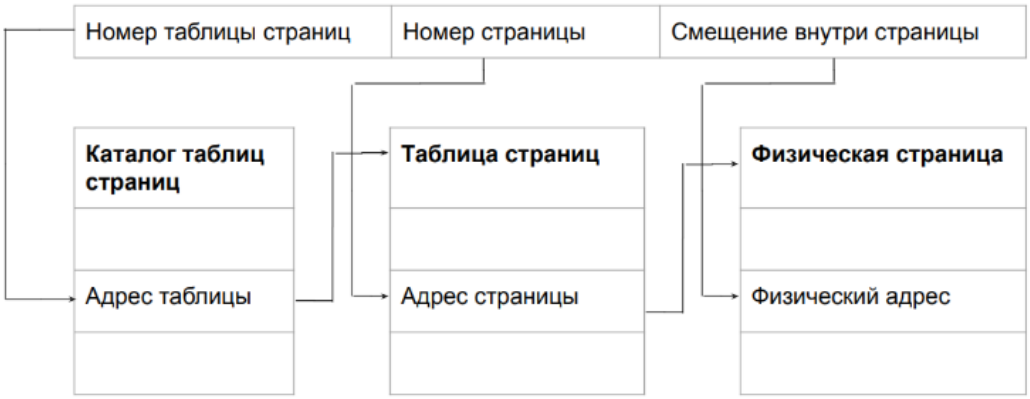


Добавлены регистры поддержки работы в защищенном режиме (обеспечивание разделения доступа программ между собой, между программами и ОС и тд; эти регистры справа на картинке)

23.1 Доп. про модели памяти, было в прошлых экзаменах + будет на Осях (это я дополнил в 2022 году)

Модели памяти

● Плоская - код и данные используют одно и то же пространство
● Сегментная - сложение сегмента и смещения (используется в реальном режиме; знакома нам)
● Страничная - виртуальные адреса отображаются на физические постранично



○ виртуальная память - метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (файл, или раздел подкачки);
○ основной режим для большинства современных ОС;
○ в x86 минимальный размер страницы - 4096 байт;
○ основывается на таблице страниц - структуре данных, используемой системой виртуальной памяти в ОС компьютера для хранения сопоставления между виртуальным адресом и физическим адресом. Виртуальные адреса используются выполняющимся процессом (программа имеет информацию только о виртуальных адресах), в то время как физические адреса используются аппаратным обеспечением.
Таблица страниц является ключевым компонентом преобразования виртуальных адресов, который необходим для доступа к данным в памяти.

● (из интернета) модель памяти в режиме V86. С точки зрения программиста эта модель памяти работает точно также как в обычном реальном режиме. Т.е. память делится на сегменты по 64 Кбайт, ячейки

внутри которых адресуются с помощью двух слов, записываемых в виде СЕГМЕНТ:СМЕЩЕНИЕ, максимальная адресуемая память 1 Мбайт и пр.

Однако в режиме V86 выполняются все проверки защиты защищенного режима, из-за чего в некоторых случаях не будут работать некоторые инструкции.

Преимущества страничной модели:
● Программы полностью изолированы друг от друга

● В память можно загрузить больше программ, чем памяти доступно (долго не использующиеся данные загружаются на диск и освобождают место)

Управление памятью в x86
https://frolov-lib.ru/books/bsp/v13/ch2_1.htm
Сегментные регистры меняют назначение: они внешне выглядят 2 байтными, но на деле они 8 байтные, просто 6 байт - теневые регистры, используются процессором для кеширования дескрипторов страниц

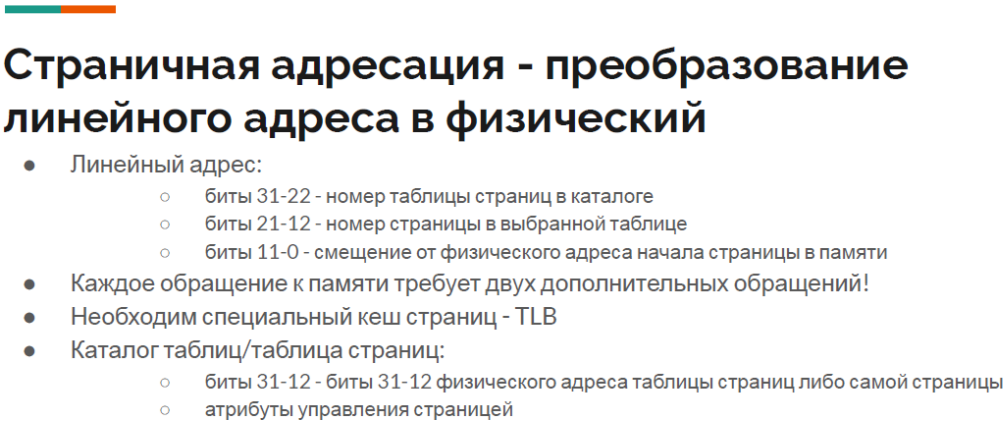
Управление памятью в x86

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

Регистры управления памятью

- GDTR: 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов (GDT) и 16-битный размер (лимит, уменьшенный на 1)
- IDTR: 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов обработчиков прерываний (IDT) и 16-битный размер (лимит, уменьшенный на 1)
- LDTR: 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий текущую таблицу локальных дескрипторов
- TR: 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий TSS текущей задачи

Страничное преобразование - преобразование линейного адреса в физический.



Процессоры x86-64. Регистры. Режимы работы.

64 разрядные. AMD – с 2001, Intel – с 2003
Режимы работы
● Legacy mode - совместимость с 32-разрядными процессорами;
● Long mode – 64-разрядный режим с частичной поддержкой 32-разрядных программ (32разрядный код не должен пересекаться с 64разрядным).
Рудименты V86 и сегментной модели памяти упразднены (DOS программы нельзя теперь запустить, только на 32разрядной системе можно запустить).

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

Соглашения о вызовах

Соглашение о вызове — формализация правил вызова подпрограмм, которое должно
включать:
● способ передачи параметров;
● способ возврата результата из функции;
● способ возврата управления.
Соглашения о вызовах определяются в рамках отдельных языков высокого уровня, а
также - различных программных API, в т. ч. API операционных систем

24. Математический сопроцессор. Типы данных.

Математический сопроцессор

Отдельное опциональное устройство на материнской плате, с 80486DX встроен в процессор.

Типы данных

- Целое слово (16 бит);
- Короткое целое (32 бита);
- Длинное слово (64 бита);
- Упакованное десятичное (80 бит);
- Короткое вещественное (32 бита);
- Длинное вещественное (64 бита);
- Расширенное вещественное (80 бит).

Представление вещественных чисел

- Нормализованная форма представления числа (1,...*2^{exp});
- Экспонента увеличена на константу для хранения в положительном виде;
- Пример представления 0,625 в коротком вещественном типе:
  - 1/2 + 1/8 = 0,101b;
  - 1,01b*2^-1;
  - Бит 31 - знак мантиссы, 30-23 - экспонента, увеличенная на 127, 22-0 - мантисса без первой цифры;
  - 0 01111110 01000000000000000000000.
- Все вычисления FPU - в расширенном 80-битном формате.

25. Математический сопроцессор. Регистры.

В сопроцессоре доступно 8 80-разрядных регистров (R0..R7).

- R0..R7, адресуются не по именам, а рассматриваются в качестве стека ST. ST соответствует регистру - текущей вершине стека, ST(1)..ST(7) - прочие регистры
- SR - регистр состояний, содержит слово состояния FPU. Сигнализирует о различных ошибках, переполнениях. Отдельные биты описывают и состояния регистров и в целом сигнализируют об ошибках (переполнениях и тп) при последней операции.
- CR - регистр управления. Контроль округления, точности (тоже 16 разрядный). Через него можно настраивать правила округления чисел и контроль точности (с помощью специальных битов устанавливать параметры, гибкие настройки)
- TW - 8 пар битов, описывающих состояния регистров: число (00), ноль (01), не-число (10), пусто (11) (изначально все пустые, проинициализированы единичками)
- FIP, FDP - адрес последней выполненной команды и её операнда для обработки исключений

26. Математический сопроцессор. Классификация команд.

Все команды сопроцессора оперируют регистрами стека сопроцессора. Если операнд в команде не указывается, то по умолчанию используется вершина стека сопроцессора (логический регистр st(0)).
Если команда выполняет действие с двумя операндами по умолчанию, то эти операнды – регистры st(0) и st(1).

1. **Команды пересылки данных:**
   - команды взаимодействия со стеком (загрузка - выгрузка вещественного числа, целого, BCD (упакованного); смена мест регистров)
   - FLD - загрузить вещественное число из источника (переменная или ST(n)) в стек. Номер вершины в SR увеличивается
   - FST/FSTP - скопировать/считать число с вершины стека в приёмик
   - FILD - преобразовать целое число из источника в вещественное и загрузить в стек
   - FIST/FISTP - преобразовать вершину в целое и скопировать/считать в приёмник
   - FBLD, FBSTP - загрузить/считать десятичное BCD-число
   - FXCH - обменять местами два регистра (вершину и источник) стека

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

- FSQRT - вычисляет квадратный корень ST(0

3. Команды сравнений:

- COM, FCOMP, FCOMPP - сравнить и вытолкнуть из стека
- FUCOM, FUCOMP, FUCOMPP - сравнить без учёта порядков и вытолкнуть
- FICOM, FICOMP, FICOMP - сравнить целые
- FCOMI, FCOMIP, FUCOMI, FUCOMIP (P6)

устанавливает биты ZF, PF, CF регистра EFLAGS в соответствии с таблицей.

- FTST - сравнивает с нулём
- FXAM - выставляет флаги в соответствии с типом числа

- Команды сравнения сравнивают значение в вершине стека с операндом. По умолчанию (если операнд не задан) происходит сравнение регистров ST(0) и ST(1). В качестве операнда может быть задана ячейка памяти или регистр. Команда устанавливает флаги (основные и в регистре SR), биты C0, C2, C3 регистра swr в соответствии с таблицей. Сбрасывает в 0 признак C1 при пустом стеке после выполнения команды.

4. Трансцедентные операции сопроцессора:

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

sin (fsin), cos(fsin) - принимают значение в радианах в некотором диапазоне.
tg, arctg, 2^x - 1, y * log2x...

5. Константы FPU:

- FLD1 – 1,0
- FLDZ - +0,0
- FLDPI - число Пи
- FLDL2E - log2e
- FLDL2T - log210
- FLDLN2 – ln(2)
- FLDLG2 – lg(2)

6. Команды управления:

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

Команда CPUID (с 80486)

Идентификация процессора CPU, предназначена для считывания программным обеспечением информации о продавце, семействе, модели и поколении процессора, а также специфической для процессора дополнительной информации (поддерживаемые наборы команд, размеры буферов, кэшей, разнообразные расширения архитектуры и т.п.)

Перед выполнением команды CPUID в регистр EAX должно помещаться входное значение, которое и указывает — какую информацию необходимо выдать.

- Если EAX = 0, то в EAX - максимальное допустимое значение (1 или 2), а EBX:ECX:EDX – 12-байтный идентификатор производителя (ASCII-строка).
- Если EAX = 1, то в EAX - версия, в EDX - информация о расширениях –
    - EAX - модификация, модель, семейство
    - EDX: наличие FPU, поддержка V86, поддержка точек останова, CR4, PAE, APIC, быстрые системные вызовы, PGE, машинно-специфичный регистр, CMOVcc, MMX, FXSR (MMX2), SSE

- Если EAX = 2, то в EAX, EBX, ECX, EDX возвращается информация о кэшах и TLB

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ ПО FPU, ДЕТАЛЬНО НА ЭКЗАМЕНЕ НЕ БУДУТ СПРАШИВАТЬ.

Исключения FPU

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

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

Расширение, которое было встроено для увеличения эффективности обработки больших потоков данных (изображение, звук, видео..) - простые операции над массивами однотипных чисел

- **8 64-битных регистров MM0..MM7 - мантиссы регистров FPU. При записи в MMn экспонента и знаковый бит заполняются единицами (отрицательная бесконечность получается)**
- **Пользоваться одновременно и FPU, и MMX не получится, требуется FSAVE+FRSTOR**

Типы данных MMX:

- учетверённое слово (64 бита);
- упакованные двойные слова (2);
- упакованные слова (4);
- упакованные байты (8).

- Команды MMX перемещают упакованные данные в память или обычные регистры целиком, но арифметические и логические операции выполняют поэлементно.
- насыщение - замена переполнения/антипереполнения превращением в максимальное/минимальное значение

28. Расширения процессора. MMX. Классификация команд.

- **8 64-битных регистров MM0..MM7 - мантиссы регистров FPU. (то есть работает с целыми) При записи в MMn экспонента и знаковый бит заполняются единицами**
- **Пользоваться одновременно и FPU, и MMX не получится, требуется FSAVE+FRSTOR**
- **Насыщение - замена переполнения/антипереполнения превращением в максимальное/минимальное значение (Светлый цвет + светлый цвет максимум равно белый (но не будет переполнения и темного цвета))**

Классификация команд:

В режиме с насыщением, результаты операции, которые переполняются сверху или снизу отсекаются к границе datarange соответствующего типа данных

В режиме без насыщения, результаты, которые переполняются как в обычной процессорной арифметике

Тип данных	Нижний предел		Верхний предел	
	Шестнадцат.	Десятичн.	Шестнадцат.	Десятичн.
Знаковый байт	80H	-128	7FH	127
Знаковое слово	8000H	-32768	7FFFH	32767
Беззнаковый байт	00H	0	FFH	255
Беззнаковое слово	0000H	0	FFFFH	65535

1. Команды пересылки данных:

- пересылка двойных/учетверенных слов;
- упаковка со знаковым насыщением слов (приемник -> младшая половина приемника, источник -> старшая половина приемника, в случае наличия значащих разрядов в отбрасываемых частях происходит насыщение);
- упаковка слов с беззнаковым насыщением, распаковка и объединение старших элементов источника и приемника через 1

• MOVD, MOVQ - пересылка двойных/учетверённых слов

• PACKSSWB, PACKSSDW - упаковка со знаковым насыщением слов в байты/двойных слов в слова. Приёмник -> младшая половина приёмника, источник -> старшая половина приёмника

Если значение слова больше или меньше границ диапазона знакового байта, то результат упаковки насыщается соответственно до 7Fh или до 80h. (до 7FFFh или до 8000h.)

• PACKUSWB - упаковка слов в байты с беззнаковым насыщением. .Если значение слова больше или меньше границ диапазона беззнакового байта, то результат упаковки насыщается соответственно до FFh или до 00h.

• PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ –
Команда PUNPCKH распаковывает старшие элементы операнда-источника и операнда-назначения в операнд-назначение. Элементы двух операндов записываются в результат через один, т.е. в старший элемент результата помещается старший элемент операнда-источника, в следующий более младший элемент — старший элемент операнда-назначения, далее — следующий элемент из операнда-источника, элемент из операнда-назначения и т.д. до полного заполнения всех элементов результата, этот результат затем помещается в операнд-назначение.

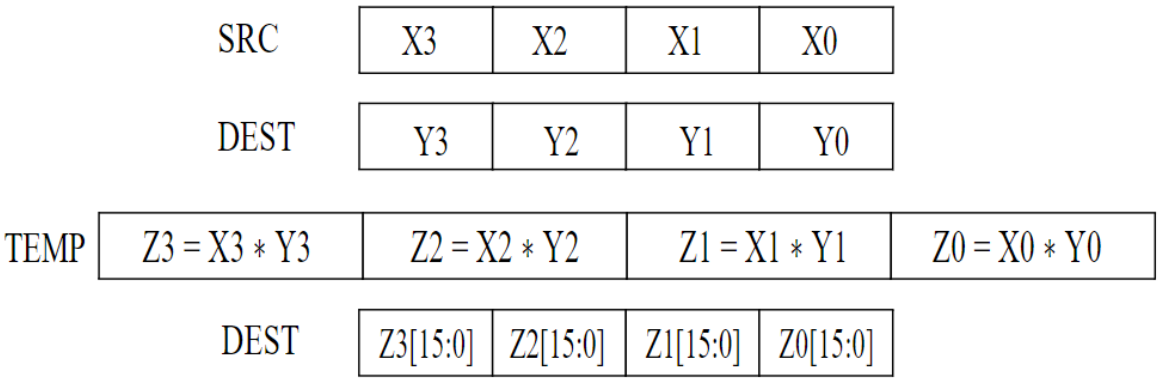
2. Арифметические операции:

- поэлментное сложение (перенос игнорируется, побайтовое сложение)
- сложение/вычитание с насыщением
- беззнаковое сложение с насыщением
- вычитание (заем игнорируется)
- вычитание с насыщением
- старшое/младшее умножение (сохраняет старшую или младшую части результата в приемник)
- умножение и сложение (перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших)

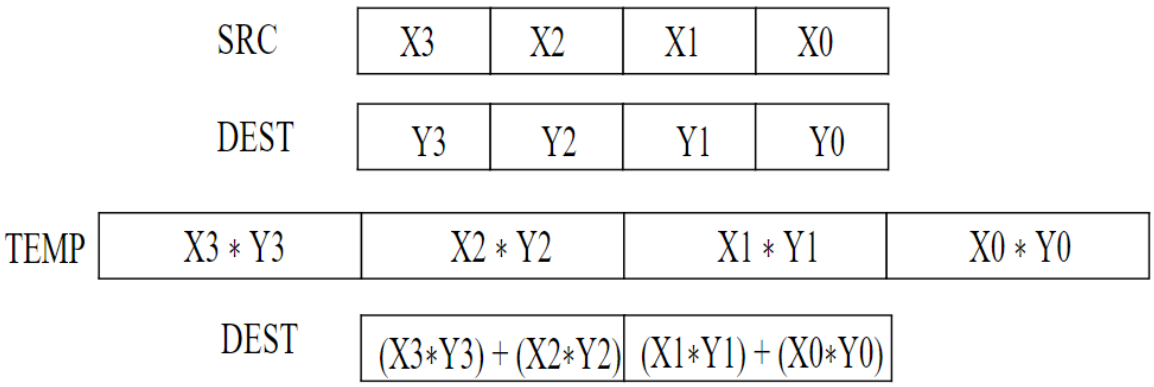
- PADDB, PADDW, PADDD - поэлементное сложение, перенос игнорируется
- PADDSB, PADDSW - сложение с насыщением
- PADDUSB, PADDUSW - беззнаковое сложение с насыщением

- PSUBB, PSUBW, PDUBD - вычитание, заём игнорируется
- PSUBSB, PSUBSW - вычитание с насыщением
- PSUBUSB, PSUBUSW - беззнаковое вычитание с насыщением

- PMILHW, PMULLW - старшее/младшее умножение (сохраняет старшую или младшую части результата в приёмник) (почему?)
  (low)



- PMADDWD - умножение и сложение. Перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших



3. Команды сравнения:

- проверка на равенство (Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе - нулями)
- проверка на больше (Если элемент приёмника больше, то заполняется единицами, иначе - нулями)

- PCMPEQB, PCMPEQW, PCMPEQD - проверка на равенство. Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе – нулями
- PCMPGTB, PCMPGTW, PCMPGTD - сравнение. Если элемент приёмника больше, то заполняется единицами, иначе - нулями

4. Логические операции:

- PAND - логическое И
- PANDN - логическое НЕ-И (штрих Шеффера) (источник*НЕ(приёмник))
- POR - логическое ИЛИ
- PXOR - исключающее ИЛИ

5. Сдвиговые операции:

- PSLLW, PSLLD, PSLLQ - логический влево
- PSRLW, PSRLD, PSRLQ - логический вправо
- PSRAW, PSRAD - арифметический вправо

Арифметический сдвиг отличается от логического тем, что он не изменяет значение старшего бита, и предназначен для чисел со знаком.

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

SSE (*Streaming SIMD Extensions*) - расширение инструкций процессора для потоковой обработки в режиме SIMD (*Single Instruction Multiple Data*), т.е. когда требуется применять однотипные операции к потоку данных. Расширение SSE разработано компанией Intel и было впервые применено в процессоре Intel Pentium III с ядром Katmai. Отсюда изначальное название KNI (Katmai New Instructions). Технология SSE позволила преодолеть проблемы MMX - при использовании MMX невозможно было одновременно использовать инструкции сопроцессора, так как его регистры задействовались для MMX и работы с вещественными числами.

Расширение SSE (Pentium III, 1999) - Решение проблемы параллельной работы с FPU

Регистры:

- 8 128-разрядных регистров
- свой регистр флагов

- Основной тип - вещественные одинарной точности (32 бита, в 1 регистре 4 числа)
- Целочисленные команды работают с регистрами MMX

Команд больше чем в MMX, типы:

- Пересылки
- Арифметические
- Сравнения
- Логические
- Преобразования типов
- Целочисленные
- Упаковки
- Управления состоянием
- Управления кэшированием

30. Расширения процессора. SSE. Классификация команд.