Задание 1

Ниже приведён подробный по шагам разбор данного кода на ассемблере для DOS, с комментариями по выбору регистров, функций, сегментов и алгоритмическому решению задачи.

**Общая задача программы**

Программа реализует следующую последовательность действий:

1. Вывод приглашения для ввода фамилии с клавиатуры (используются английские символы).
2. Чтение с клавиатуры всех нажатых клавиш до нажатия клавиши Enter. Для каждой клавиши программа сохраняет:
   * ASCII-код (в регистре AL после вызова прерывания 16h).
   * Scan-код (в регистре AH после вызова прерывания 16h).
3. Сохранённые коды помещаются в два буфера (asciiBuf и scanBuf). При этом переменная task1\_count фиксирует количество введённых символов.
4. После завершения ввода программа сначала выводит строку с ASCII-кодами, затем – для каждого введённого символа выводит соответствующий scan-код в шестнадцатеричном виде.
5. В конце программа ожидает нажатия клавиши и завершает работу через функцию DOS.

**2. Основной код (процедура main)**

**2.1. Инициализация и подготовка сегментов**

main proc far

push ds

xor ax,ax

push ax

mov ax,dseg

mov ds,ax

* **Описание:**
  + **Сохранение DS:**  
    Перед изменением DS сохраняется его значение (push ds).
  + **Обнуление AX и push 0:**  
    xor ax,ax обнуляет AX, затем этот ноль помещается в стек. Возможно, это делается для дальнейшего использования при возврате или просто для выравнивания стека.
  + **Установка сегмента данных:**  
    mov ax, dseg загружает адрес сегмента данных, затем mov ds, ax устанавливает DS.
* **Почему так:**  
  Это стандартная процедура инициализации DS для доступа к переменным, объявленным в сегменте данных.

**2.2. Подготовка буферов**

xor si,si ; Индекс для буферов

* **Описание:**  
  Регистр SI обнуляется и используется как индекс для сохранения символов в обоих буферах (asciiBuf и scanBuf).
* **Почему так:**  
  SI выбран, так как часто используется для работы с последовательными данными. Обнуление гарантирует, что мы начнём запись с начала буфера.

**2.3. Вывод приглашения**

; Вывод приглашения

mov dx, OFFSET task1Msg1

mov ah,09h

int 21h

* **Описание:**
  + Регистр DX получает адрес строки приглашения (task1Msg1).
  + Функция DOS с AH=09h выводит строку на экран до символа ‘$’.
* **Почему AH=09h:**  
  Эта функция предназначена для вывода строк, где завершение строки определяется символом ‘$’.

**2.4. Чтение клавиш – цикл ввода**

ReadLoop:

mov ah,00h

int 16h ; AH=scan, AL=ASCII

cmp al,13 ; Enter?

je ShowResults

mov asciiBuf[si],al

mov scanBuf[si],ah

inc si

cmp si,50

jb ReadLoop

* **Описание:**
  + **Вызов int 16h:**  
    При вызове int 16h с AH=00h происходит ожидание нажатия клавиши.
    - Регистр AL получает ASCII-код клавиши.
    - Регистр AH получает scan-код (код аппаратного сканирования).
  + **Проверка на Enter:**  
    Сравнение cmp al,13 проверяет, равен ли ASCII-код клавиши 13 (код клавиши Enter). Если да, переход в метку ShowResults.
  + **Запись в буферы:**  
    Если нажатая клавиша не Enter, коды записываются в буферы:
    - asciiBuf[si] получает содержимое AL (ASCII).
    - scanBuf[si] получает содержимое AH (scan-код).
  + **Инкремент индекса и ограничение:**  
    SI увеличивается на 1, затем сравнивается с 50 (максимальный размер буфера). Если SI меньше 50, переход к ReadLoop для чтения следующего символа.
* **Почему так:**  
  Использование int 16h для получения клавиш – стандарт в DOS. Ограничение ввода (50 символов) предотвращает выход за пределы буфера.

**2.5. Сохранение количества введённых символов**

ShowResults:

mov task1\_count,si

* **Описание:**  
  После завершения ввода переменная task1\_count получает значение SI – количество нажатых клавиш (без Enter).
* **Почему так:**  
  Это необходимо для корректного последующего цикла вывода – знать, сколько элементов обрабатывать.

**2.6. Вывод ASCII-кодов**

; Вывод ASCII

mov dx, OFFSET task1Msg2

mov ah,09h

int 21h

mov cx,task1\_count

xor si,si

ShowAscii:

mov dl,asciiBuf[si]

mov ah,02h

int 21h

inc si

loop ShowAscii

* **Описание:**
  + Сначала выводится строка-заголовок с помощью функции 09h (DX указывает на task1Msg2).
  + Регистр CX устанавливается равным количеству введённых символов.
  + SI обнуляется (начинаем с начала буфера).
  + В цикле:
    - Каждый символ из asciiBuf загружается в DL.
    - Функция DOS с AH=02h выводит символ, находящийся в DL.
    - Команда loop уменьшает CX и повторяет цикл, пока CX не станет 0.
* **Почему AH=02h:**  
  Функция 02h предназначена для вывода одиночного символа, переданного в DL.

**2.7. Вывод Scan-кодов в шестнадцатеричном виде**

; Вывод Scan-кодов

mov dx, OFFSET task1Msg3

mov ah,09h

int 21h

mov cx,task1\_count

xor si,si

ShowScan:

mov al,scanBuf[si]

call PrintHex

mov dl,' '

mov ah,02h

int 21h

inc si

loop ShowScan

* **Описание:**
  + Выводится строка-заголовок task1Msg3 функцией 09h.
  + CX устанавливается количеством введённых символов.
  + SI обнуляется.
  + В цикле:
    - Из буфера scanBuf извлекается scan-код, сохраняемый в AL.
    - Вызывается процедура PrintHex, которая выводит значение AL в виде двух шестнадцатеричных цифр.
    - После этого выводится пробел (ASCII-код пробела) для разделения кодов.
    - SI увеличивается, и цикл повторяется до обработки всех введённых значений.
* **Почему так:**  
  Отображение scan-кодов в шестнадцатеричном формате полезно, поскольку эти коды не всегда соответствуют печатным символам. Вывод в шестнадцатеричном виде делает их понятными.

**2.8. Ожидание клавиши и завершение программы**

; Ожидание любой клавиши перед выходом

mov ah,00h

int 16h

; Завершение программы

mov ax,4C00h

int 21h

* **Описание:**
  + После вывода результатов программа вызывает int 16h с AH=00h, что заставляет DOS ждать нажатия клавиши.
  + Затем, используя функцию 4Ch (через int 21h), программа завершается.
* **Почему так:**  
  Ожидание позволяет пользователю увидеть результаты, а функция завершения корректно возвращает управление операционной системе.

**3. Вспомогательные процедуры для вывода шестнадцатеричных цифр**

**3.1. Процедура PrintHex**

PrintHex PROC

push ax

push cx

mov ah,0

mov cl,16

div cl

call PrintDigit

mov al,ah

call PrintDigit

pop cx

pop ax

ret

PrintHex ENDP

* **Назначение процедуры:**  
  Выводит байт (находящийся в AL) как два шестнадцатеричных символа.
* **Пошаговый разбор:**
  1. **Сохранение регистров:**  
     Перед выполнением вычислений сохраняются значения регистров AX и CX, чтобы не повредить состояние вызывающего кода.
  2. **Деление на 16:**
     + Устанавливается CL = 16.
     + Команда div cl делит значение AX на 16:
       - Частное записывается в AL.
       - Остаток – в AH.
     1. **Вызов PrintDigit:**  
        Первый вызов процедуры выводит символ, соответствующий числу, хранящемуся в остатке (то есть – младшая шестнадцатеричная цифра).  
        Затем значение из AH (старшая цифра) помещается в AL, и процедура вызывается повторно.
  3. **Восстановление регистров:**  
     После завершения выводимых операций восстанавливаются CX и AX.
* **Почему так:**  
  Деление на 16 позволяет разделить байт на две шестнадцатеричные цифры. Последовательный вызов PrintDigit выводит сначала одну цифру, а затем другую.

**3.2. Процедура PrintDigit**

PrintDigit PROC

push ax

cmp al,10

jb Digit

add al,7

Digit: add al,'0'

mov dl,al

mov ah,02h

int 21h

pop ax

ret

PrintDigit ENDP

* **Назначение процедуры:**  
  Преобразовывает число (0–15, находящееся в AL) в его ASCII-символ в шестнадцатеричном формате и выводит его.
* **Пошаговый разбор:**
  1. **Проверка значения AL:**  
     Если AL меньше 10 (т.е. цифры от 0 до 9), программа переходит к метке Digit.
  2. **Обработка значений от 10 до 15:**  
     Если значение 10 или больше, к AL прибавляется 7. Это сдвигает диапазон [10…15] к диапазону, соответствующему буквенным символам A–F. Например, для 10 получается 10 + 7 = 17, а затем прибавлением '0' (48) – итоговый ASCII-код 'A'.
  3. **Формирование ASCII-символа:**  
     После метки Digit к AL прибавляется ASCII-код символа '0'. Таким образом, для чисел 0–9 получится правильное представление, а для чисел от 10 до 15 – символы A–F.
  4. **Вывод символа:**  
     Результат помещается в DL, затем вызывается функция DOS с AH=02h для вывода единственного символа.
  5. **Восстановление регистра AX:**  
     В конце извлекается сохранённое значение AX.
* **Почему так:**  
  Эта процедура универсальна для преобразования цифры в ASCII, что упрощает вывод шестнадцатеричного значения байта.

**Итоговое обоснование выбора решений**

* **Сегментированная организация.**  
  Код разбит на сегменты: стек для временных данных и вызовов (stseg), данные для строк и буферов (dseg) и код в отдельном сегменте (cseg). Это соответствует архитектуре x86 в реальном режиме DOS.
* **Использование прерываний DOS/BIOS.**
  + **int 21h, функции 09h и 02h** используются для вывода строк и символов.
  + **int 16h, функция 00h** применяется для чтения нажатых клавиш с клавиатуры.  
    Такой выбор функций – стандарт для DOS-программ.
* **Работа с буферами и индексированием.**  
  Регистр SI используется как индекс при записи в массивы. Ограничение ввода (50 символов) предотвращает выход за границы буфера.
* **Модульное разделение кода.**  
  Для вывода шестнадцатеричных значений выделены процедуры PrintHex и PrintDigit, что делает код более структурированным и повторно используемым.
* **Сохранение состояния регистров.**  
  В ключевых местах (например, при вызове процедур PrintHex/PrintDigit) используется push/pop для сохранения значений регистров, что позволяет избежать потерь данных, необходимых в основном алгоритме.

В результате данная программа демонстрирует грамотное применение работы с клавиатурным вводом, записью в буферы, конвертацией числовых значений в шестнадцатеричный формат и выводом на экран с использованием стандартных функций DOS и BIOS.

Задание 2

Ниже приведён подробный пошаговый разбор данного кода на ассемблере для DOS, с объяснением использования регистров, сегментов, функций BIOS/DOS и логики реализации эхо-ввода с завершением по нажатию клавиши Esc.

**1. Структура сегментов**

**1.1. Стековый сегмент (stseg)**

stseg segment para stack

dw 64 dup(?)

stseg ends

* **Назначение:**  
  Этот сегмент выделяет память под стек, который используется для хранения временных значений регистров, адресов возврата при вызове процедур и для управления подпрограммными вызовами.
* **Подробности:**
  + Директива segment para stack указывает, что сегмент выравнивается по параграфной границе (16 байт) и будет использоваться как стек.
  + Инструкция dw 64 dup(?) резервирует 64 слова (каждое по 2 байта, итого 128 байт) для стека.
* **Почему именно так:**  
  Размер стека выбран исходя из сложности программы – здесь достаточно 64 слов, так как программа не выполняет глубоких вложенных вызовов и сложных операций, требующих большого количества сохранений регистров.

**1.2. Сегмент данных (dseg)**

dseg segment para

task2Msg db 13,10,'Enter text (echo input, Esc to finish): ','$'

dseg ends

* **Назначение:**  
  Этот сегмент содержит строковую константу, которая является сообщением для пользователя, приглашающим к вводу текста. Завершающий символ $ используется для функции вывода строк DOS.
* **Подробности:**
  + Символы 13 и 10 – это контрольные символы CR и LF, которые переводят строку на новую строку.
  + Текстовый вывод (через функцию DOS 09h) завершается символом $.
* **Почему именно так:**  
  Использование $ в конце строки – стандартное требование для функции вывода строк (AH=09h) в DOS, а CR и LF добавлены для форматирования вывода (переход на новую строку).

**1.3. Сегмент кода (cseg)**

cseg segment para

assume cs:cseg, ds:dseg, ss:stseg

* **Назначение:**  
  Здесь находится исполняемый код программы.
* **Директива assume:**
  + cs:cseg – кодовый сегмент находится в cseg.
  + ds:dseg – сегмент данных – dseg.
  + ss:stseg – стековый сегмент – stseg.
* **Почему так:**  
  Связывание регистров с соответствующими сегментами необходимо для корректной адресации – чтобы инструкции, обращаясь к данным или стеку, использовали правильные сегменты.

**2. Основной код (процедура main)**

**2.1. Инициализация сегмента данных**

main proc far

push ds

xor ax,ax

push ax

mov ax,dseg

mov ds,ax

* **Что происходит:**
  1. **Сохранение DS:**
     + Инструкция push ds сохраняет текущее значение регистра DS на стек, чтобы позже можно было вернуть исходное состояние, если это потребуется.
  2. **Обнуление AX:**
     + xor ax, ax обнуляет регистр AX. Это быстрый способ установить нулевое значение.
  3. **Push 0:**
     + push ax кладёт ноль в стек. Этот шаг может использоваться для выравнивания или подготовки для дальнейших вызовов (иногда используется в качестве фиктивного сегмента, хотя здесь может быть вызван по традиции – сохранение нулевого значения).
  4. **Установка сегмента данных:**
     + mov ax, dseg загружает адрес сегмента данных.
     + mov ds, ax устанавливает DS равным адресу сегмента данных, что необходимо для доступа к переменным (например, к строке task2Msg).
* **Почему так:**  
  При запуске программы значение DS может быть неопределённым, поэтому важно явно инициализировать DS значением сегмента, в котором расположены данные.

**2.2. Вывод приглашения пользователю**

; Вывод приглашения

mov dx, OFFSET task2Msg

mov ah,09h

int 21h

* **Что происходит:**
  + Регистр DX получает адрес (смещение) строки task2Msg (сообщение для ввода).
  + Функция DOS с кодом 09h (AH=09h) выводит строку, начиная с адреса, указанного в DX, до первого символа $.
* **Почему AH=09h:**  
  Эта функция предназначена для вывода строк, где символ завершающий строку – знак доллара $.

**2.3. Цикл эхо-ввода (InputLoop)**

InputLoop:

mov ah,00h ; Чтение клавиши

int 16h ; AH = scan code, AL = ASCII

cmp al,27 ; Проверка на Esc

je ExitProg

mov ah,0Eh ; Вывод символа

int 10h

jmp InputLoop

* **Описание цикла:**
  1. **Чтение клавиши:**
     + Инструкция mov ah, 00h подготавливает регистр для вызова BIOS-функции чтения клавиши.
     + Команда int 16h вызывает BIOS-прерывание, которое ожидает нажатия клавиши. После этого:
       - Регистр AL содержит ASCII-код нажатой клавиши.
       - Регистр AH содержит scan-код (аппаратный код клавиши).
  2. **Проверка символа Esc:**
     + cmp al,27 сравнивает содержимое AL с 27h (ASCII-код для клавиши Esc).
     + Если нажат Esc (AL = 27h), команда je ExitProg переходит к метке для завершения программы.
  3. **Эхо-вывод введенного символа:**
     + mov ah,0Eh подготавливает регистр для функции BIOS, которая выводит символ на экран.
     + Команда int 10h вызывает видеопрерывание и выводит символ, находящийся в AL, на экран.
  4. **Повтор цикла:**
     + Инструкция jmp InputLoop безусловно возвращает выполнение в начало цикла, ожидая следующего нажатия клавиши.
* **Почему так:**  
  Этот цикл реализует эхо-ввод – при каждом нажатии клавиши вводимый символ сразу же выводится на экран. Прерывание ввода (Esc) позволяет пользователю завершить ввод.

**2.4. Завершение ввода – метка ExitProg**

ExitProg:

; Ожидание любой клавиши

mov dx, OFFSET task2Msg ; Произвольный адрес для примера

mov ah,08h

int 21h

* **Что происходит:**
  + После нажатия клавиши Esc выполнение переходит к метке ExitProg.
  + Здесь вызывается функция DOS с AH=08h, которая предназначена для ожидания нажатия клавиши без вывода на экран (без эха).
  + В данном примере в DX загружается произвольное значение – адрес task2Msg; хотя значение DX здесь не имеет значения для функции 08h, поскольку она опирается лишь на AH для ожидания ввода.
* **Почему именно так:**  
  Ожидание любой клавиши даёт пользователю возможность увидеть окончательный результат перед выходом из программы. Использование функции 08h – стандартный способ реализации такого ожидания.

**2.5. Завершение программы**

mov ax,4C00h

int 21h

* **Что происходит:**
  + Регистр AX получает значение 4C00h, где 4Ch – номер функции DOS для завершения программы, а 00h – код возврата.
  + Вызов int 21h завершает выполнение программы, возвращая управление операционной системе.
* **Почему так:**  
  Функция DOS с AH=4Ch является стандартным способом корректного завершения программы в DOS, гарантируя освобождение ресурсов и правильный возврат управления.

**3. Завершение процедуры и директива END**

main endp

cseg ends

end main

* **Объяснение:**
  + main endp завершает описание процедуры main.
  + cseg ends закрывает сегмент кода.
  + Директива end main сообщает ассемблеру, что точка входа в программу – это метка main.
* **Почему так:**  
  Это стандартное оформление в ассемблере, которое позволяет корректно сформировать объектный файл и указать точку старта программы.

**Итоговое обоснование решений**

1. **Сегментированная организация памяти:**  
   Разделение на стековый, данных и кодовый сегменты (stseg, dseg, cseg) соответствует модели сегментированной памяти x86 и обеспечивает правильную работу с адресацией.
2. **Инициализация DS:**  
   Перед выполнением работы с данными DS явно устанавливается равным сегменту dseg, что необходимо для корректного доступа к строке и другим переменным.
3. **Вывод приглашения:**  
   Использование функции 09h (через int 21h) позволяет вывести строку до символа “$”, что является стандартом в DOS-программировании.
4. **Чтение клавиш и эхо-вывод:**
   * Прерывание int 16h с AH=00h позволяет дождаться нажатия клавиши, возвращая ASCII (в AL) и scan-код (в AH).
   * Цикл проверки нажатия Esc (код 27h) обеспечивает завершение ввода по требованию пользователя.
   * Функция int 10h с AH=0Eh используется для вывода символа с немедленным эффектом (эхо), что улучшает интерактивность программы.
5. **Ожидание перед завершением:**  
   После завершения ввода программа ожидает нажатия клавиши (через int 21h, AH=08h), что позволяет пользователю увидеть результат работы программы до её закрытия.
6. **Корректное завершение программы:**  
   Функция DOS с AH=4Ch гарантирует правильное завершение работы программы и возвращение управления операционной системе.

Таким образом, этот код демонстрирует классический способ реализации эхо-ввода в DOS: приглашение к вводу, чтение клавиш с выводом введённых символов и завершение ввода по нажатию клавиши Esc, что является хорошей практикой в программировании под DOS на ассемблере.

Задание 3

Ниже приведён подробный по шагам разбор данного кода на ассемблере для DOS с объяснением назначения каждого сегмента, процедуры и инструкции, а также обоснованием выбора регистров и алгоритма решения задачи.

**Общая идея программы**

Программа запрашивает ввод двух дат (например, дней рождения) у пользователя, затем парсит введённые строки, извлекая значения дня, месяца и года. После этого вычисляется «средняя» дата на основе первых двух дат. Результат (сформированная строка с датой в формате «день.месяц.год») выводится на экран. Программа демонстрирует работу с вводом/выводом, преобразованием строки в число (atoi), вычислением среднего значения и форматированием числа обратно в строку.

**1. Структурное разделение кода**

**1.1. Стековый сегмент (stseg)**

stseg segment para stack

dw 256 dup(?)

stseg ends

* **Назначение:**  
  Выделение области для стека, где будут сохраняться временные значения регистров, адреса возврата, данные при вызове процедур и при работе с push/pop.
* **Подробности:**  
  Здесь резервируется 256 слов (каждое слово – 2 байта, итого 512 байт), что достаточно для данной программы, так как вызывается несколько процедур (например, для конвертации строк в числа и форматирования) с вложенными вызовами.

**1.2. Сегмент данных (dseg)**

dseg segment para

prompt1 db 'IVANOV: $'

prompt2 db 'PETROVA: $'

days db 2 dup(0)

months db 2 dup(0)

years dw 2 dup(0)

date\_buffer db 11, 0, 11 dup(0)

result\_str db 'birthday: $'

birthday\_day db '00.00.0000$'

year\_str db '0000$'

newline db 0Dh, 0Ah, '$'

count dw 0

dseg ends

* **Назначение и структура данных:**
  + **Строковые приглашения:**  
    prompt1 и prompt2 – строки для приглашения ввода данных (например, фамилии или даты) для двух записей. Символ $ служит завершающим маркером для функции вывода (функция DOS 09h).
  + **Массивы для хранения даты:**
    - days – хранит введённое значение дня для двух дат (2 байта для каждого элемента).
    - months – аналогично для месяца.
    - years – массив для хранения года (каждый элемент – 2 байта).
  + **Буфер для ввода даты:**  
    date\_buffer – предназначен для приема строки с датой. Первый байт (11) задаёт максимальную длину ввода, второй – фактическое число введённых символов, затем – сама строка.
  + **Строки для вывода результата:**  
    result\_str – заголовок «birthday:»,  
    birthday\_day – строка результата в формате «00.00.0000».  
    year\_str – временный буфер для преобразованного года.  
    newline – содержит символы перевода строки (CR/LF) для форматированного вывода.
  + **Счетчик введённых дат:**  
    Переменная count (типа word) используется для подсчёта количества успешно обработанных дат, что позволяет организовать обработку двух вводов.

**1.3. Сегмент кода (cseg)**

cseg segment para

assume cs:cseg, ds:dseg, ss:stseg

* **Назначение и инструкция assume:**  
  Здесь находится основной исполняемый код. Директива assume указывает ассемблеру, что:
  + Код находится в сегменте cseg (CS),
  + Данные в сегменте dseg (DS),
  + Стек в сегменте stseg (SS).

Это обеспечивает правильную адресацию памяти в модели сегментирования x86.

**2. Основной блок – процедура task (точка входа)**

task proc far

push ds

xor ax, ax

push ax

mov ax, dseg

mov ds, ax

* **Инициализация DS:**
  + Сохраняем текущее значение DS (через push ds), затем обнуляем AX (xor ax,ax) и сохраняем 0 в стек (push ax) – эта техника часто используется для выравнивания или подготовки стека.
  + Далее mov ax, dseg и mov ds,ax устанавливают сегмент данных для доступа к переменным.

**2.1. Ввод первой даты**

; Ввод первой даты

mov dx, offset prompt1

mov ah, 09h

int 21h

call input\_date

call parse\_date

* **Вывод приглашения:**  
  Регистр DX получает адрес строки prompt1 ("IVANOV: $"), затем функция DOS (AH=09h) выводит её.
* **Ввод и разбор даты:**  
  Сначала вызывается процедура input\_date для получения ввода от пользователя (через буфер date\_buffer). Затем вызывается parse\_date для преобразования строки в числовые значения дня, месяца и года.
* **Использование count:**  
  Переменная count из сегмента данных используется для индексации текущей даты (при обработке двух записей).

**2.2. Ввод второй даты**

; Ввод второй даты

mov dx, offset prompt2

mov ah, 09h

int 21h

call input\_date

call parse\_date

* **Аналогично первой дате:**  
  Выводится приглашение prompt2 ("PETROVA: $"), затем повторно считывается строка в date\_buffer и производится её разбор.

**2.3. Вычисление результата**

; Вычисление результата

call compute\_average

* **Что происходит:**  
  После ввода и разбора двух дат вызывается процедура compute\_average, которая вычисляет среднее значение года, а также среднего дня и месяца. Результирующая дата форматируется и записывается в строку birthday\_day.

**2.4. Вывод результата**

; Вывод результата

mov dx, offset result\_str

mov ah, 09h

int 21h

mov dx, offset birthday\_day

mov ah, 09h

int 21h

* **Вывод заголовка и отформатированной даты:**  
  Сначала выводится строка result\_str ("birthday: $"), затем – строка birthday\_day с результатом (формат "дд.мм.гггг$").

**2.5. Ожидание клавиши и завершение**

; Ожидание нажатия клавиши

mov ah, 08h

int 21h

ret

task endp

* **Ожидание ввода:**  
  Функция DOS (AH=08h) ожидает нажатия клавиши, позволяя пользователю увидеть результат до завершения программы.
* **Завершение:**  
  После ожидания управление возвращается вызывающему коду (в данном случае – завершает процедуру task).

**3. Процедуры ввода и обработки дат**

**3.1. input\_date**

input\_date proc near

call clear\_keyboard

mov dx, offset date\_buffer

mov ah, 0Ah

int 21h

mov dx, offset newline

mov ah, 09h

int 21h

ret

input\_date endp

* **Назначение:**  
  Эта процедура очищает клавиатурный буфер (чтобы убрать предыдущие нажатия), затем вызывает функцию DOS (AH=0Ah) для ввода строки в буфер date\_buffer. После ввода выводится перевод строки (строка newline).
* **Почему так:**  
  Функция 0Ah работает по принципу «буферизированного ввода»: первый байт задаёт максимальную длину, второй – реальное количество введённых символов, а затем сама строка. Перед вводом очищается клавиатурный буфер с помощью процедуры clear\_keyboard.

**3.2. parse\_date**

parse\_date proc near

mov di, [count]

mov si, offset date\_buffer + 2

; Парсинг дня (дд)

call atoi\_2

mov days[di], al

add si, 3 ; пропустить точку

; Парсинг месяца (мм)

call atoi\_2

mov months[di], al

add si, 3 ; пропустить точку

; Парсинг года (гггг)

call atoi\_4

shl di, 1

mov years[di], ax

shr di, 1

inc word ptr [count]

ret

parse\_date endp

* **Алгоритм:**
  + **Инициализация:**  
    Регистр DI загружает текущее значение переменной count (номер текущей даты, 0 или 1). SI устанавливается на начало текстовой части буфера (date\_buffer + 2 – пропускаем первые два служебных байта, управляемые функцией 0Ah).
  + **Парсинг дня:**  
    Вызывается atoi\_2 для преобразования двух символов в число. Полученное значение (в AL) сохраняется в массив days по индексу DI.  
    Затем SI увеличивается на 3, чтобы пропустить разделитель (точку и, возможно, пробел).
  + **Парсинг месяца:**  
    Аналогично вызывается atoi\_2 для месяца, и результат записывается в массив months.
  + **Парсинг года:**  
    Вызывается atoi\_4 для преобразования четырёх символов в число (год). При записи в массив years используется арифметика с регистром DI: сначала DI умножается на 2 (так как массив years – массив word'ов), затем результат записывается, после чего DI возвращается в исходное значение.  
    Наконец, переменная count увеличивается на 1, чтобы отслеживать число обработанных дат.
* **Почему так:**  
  Разделение на процедуры atoi\_2 и atoi\_4 позволяет универсально преобразовывать строковые представления чисел в двоичные значения.

**3.3. clear\_keyboard**

clear\_keyboard proc near

push ax

clear\_loop:

mov ah, 01h

int 16h

jz buffer\_empty

mov ah, 00h

int 16h

jmp clear\_loop

buffer\_empty:

pop ax

ret

clear\_keyboard endp

* **Назначение:**  
  Очищает клавиатурный буфер перед новым вводом.
* **Алгоритм:**  
  В цикле вызывается функция DOS (int 16h, AH=01h) – проверка наличия символа в буфере. Если входной буфер не пуст, производится чтение символа (AH=00h) для его удаления. Цикл завершается, когда функция int 16h возвращает ноль (нет ожидающих символов).
* **Почему так:**  
  Это необходимо, чтобы предыдущие нажатия не попали в новый ввод (избежать накладок).

**3.4. atoi\_2 и atoi\_4**

**atoi\_2**

atoi\_2 proc near

xor ax, ax

mov al, [si] ; первая цифра

sub al, '0'

mov bx, 10

mul bx ; AX = AL \* 10

mov bl, [si+1] ; вторая цифра

sub bl, '0'

add al, bl

ret

atoi\_2 endp

* **Назначение:**  
  Преобразует две цифры (символы) в число (от 00 до 99).
* **Алгоритм:**  
  Извлекается первая цифра, превращается в число путём вычитания кода символа '0', умножается на 10, затем добавляется вторая цифра (аналогичным способом). Результат оказывается в AL.

**atoi\_4**

atoi\_4 proc near

xor ax, ax

mov cx, 4

atoi4\_loop:

mov bx, 10

mul bx ; AX = AX \* 10

mov bl, [si]

sub bl, '0'

xor bh, bh

add ax, bx

inc si

loop atoi4\_loop

ret

atoi\_4 endp

* **Назначение:**  
  Преобразует четыре символа в число (год, например, от 0000 до 9999).
* **Алгоритм:**  
  В цикле (4 итерации) значение в AX умножается на 10, затем к нему добавляется следующая цифра, преобразованная из символа. Регистр SI инкрементируется после обработки каждого символа.

**4. Вычисление среднего и форматирование результата**

**4.1. compute\_average**

compute\_average proc near

; Вычисление среднего года

mov ax, years[0]

add ax, years[2]

shr ax, 1

mov bx, ax ; сохраняем средний год в BX

; Очистка year\_str перед форматированием

mov di, offset year\_str

mov cx, 4

clear\_year\_str:

mov byte ptr [di], '0'

inc di

loop clear\_year\_str

; Восстанавливаем значение AX из BX перед форматированием

mov ax, bx

call format\_4\_digits

; Установка ES для корректного копирования

mov ax, ds

mov es, ax

; Копирование отформатированного года из year\_str в birthday\_day (начиная с позиции 6)

mov si, offset year\_str

mov di, offset birthday\_day+6

mov cx, 4

rep movsb

; Вычисление среднего дня

xor ax, ax

mov al, days[0]

add al, days[1]

shr ax, 1

call format\_2\_digits

mov [birthday\_day], ah

mov [birthday\_day+1], al

; Вычисление среднего месяца

xor ax, ax

mov al, months[0]

add al, months[1]

shr ax, 1

call format\_2\_digits

mov [birthday\_day+3], ah

mov [birthday\_day+4], al

ret

compute\_average endp

* **Средний год:**  
  Сначала из массива years берутся два значения (при условии, что элементы расположены подряд – первый по адресу years[0] и второй по адресу years[2], так как каждый элемент – 2 байта). Они суммируются, после чего результат делится на 2 (сдвиг вправо на 1). Это среднее значение записывается в BX, а затем форматируется с помощью процедуры format\_4\_digits для перевода числа в строку (в буфере year\_str).
* **Копирование отформатированного года:**  
  Отформатированная строка year\_str копируется в позицию в строке birthday\_day, начиная с 7-го символа (позиция 6, так как индексация с 0).
* **Средний день и месяц:**  
  Аналогично вычисляются средние значения дня и месяца на основании массивов days и months. Результат передаётся в процедуру format\_2\_digits, которая возвращает два ASCII-символа. Результирующие символы записываются в соответствующие позиции строки birthday\_day:
  + День – позиция 0 и 1 (формат "дд"),
  + Месяц – позиция 3 и 4 (формат "мм").
* **Почему так:**  
  Вычисление среднего производится по отдельности для дня, месяца и года, что позволяет составить итоговую дату в формате «дд.мм.гггг». После этого данные форматируются для корректного вывода.

**4.2. format\_2\_digits**

format\_2\_digits proc near

mov bl, 10

div bl ; делим AX на 10, остаток в AH

add ax, '00' ; преобразование в ASCII

xchg al, ah ; меняем порядок цифр

ret

format\_2\_digits endp

* **Назначение:**  
  Преобразует двузначное число в его ASCII-представление.
* **Алгоритм:**
  + Делим число на 10 (используется делитель в BL).
  + После деления в AL оказывается целая часть, в AH – остаток.
  + К сумме прибавляется '0' для преобразования в символы ASCII.
  + Затем меняем местами AL и AH, чтобы цифры оказались в правильном порядке (сначала старшая цифра, затем младшая).

**4.3. format\_4\_digits**

format\_4\_digits proc near

mov si, offset year\_str + 3 ; заполняем с конца

mov cx, 4

mov bx, 10

format4\_loop:

xor dx, dx

div bx ; деление: DX:AX / BX

add dl, '0'

mov [si], dl

dec si

loop format4\_loop

ret

format\_4\_digits endp

* **Назначение:**  
  Преобразует число (находящееся в AX, представляющее год) в строку из 4 цифр и записывает результат в буфер year\_str.
* **Алгоритм:**
  + Устанавливается указатель SI на конец буфера (offset year\_str + 3), так что цифры будут заполняться с конца.
  + Цикл выполняется 4 раза (каждая итерация обрабатывает одну цифру). В цикле:
    - Число в AX делится на 10, остаток (в DL) – это очередная цифра.
    - Остаток преобразуется в ASCII (прибавлением '0').
    - Полученный символ записывается в позицию, на которую указывает SI, после чего SI декрементируется.
  + Таким образом, число раскладывается в строковый формат с учётом порядка цифр.

**Итоговое обоснование решений**

1. **Структурирование по сегментам:**  
   Программа разделена на стеки (stseg), данные (dseg) и код (cseg) – это стандартная организация для DOS-программ на ассемблере. Такое разделение обеспечивает правильную адресацию памяти и упрощает поддержку кода.
2. **Ввод и парсинг даты:**
   * Процедура input\_date использует функцию DOS (0Ah) для получения строки, при этом предварительно очищая клавиатурный буфер.
   * Затем parse\_date разбивает строку на три части – день, месяц, год – с использованием специализированных процедур atoi\_2 и atoi\_4 для преобразования символов в число.
3. **Вычисление среднего:**  
   После ввода двух дат, их компоненты (день, месяц, год) суммируются и делятся на два для получения среднего значения. Результат форматируется отдельно для дня и месяца (с помощью format\_2\_digits) и года (через format\_4\_digits).
4. **Форматирование и вывод:**  
   Сформированное текстовое представление среднего года копируется в итоговую строку birthday\_day, а в ней же заменяются соответствующие позиции для дня и месяца. В конце выводятся предварительно сформированные строки (например, result\_str и birthday\_day) через функцию DOS (09h).
5. **Использование регистров и процедур:**
   * Регистры AX, BX, SI, DI и CX используются для промежуточных вычислений, индексирования и деления, что обеспечивает эффективное исполнение программы.
   * Процедуры разделены на логические блоки (ввод, парсинг, вычисление, форматирование), что повышает читаемость и поддержку кода.
   * Push/pop используются для сохранения важных регистров при вызове процедур, чтобы не нарушить состояние основного алгоритма.

Таким образом, программа демонстрирует комплексное решение задачи ввода двух дат, их обработки и вычисления среднего значения с последующим форматированным выводом результата. Каждый этап реализован с учётом особенностей архитектуры x86 в DOS, что делает программу наглядным примером работы с вводом, арифметикой и строковыми операциями на ассемблере.