**ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ ВЫПОЛНЕНИЯ**

**УЧЕБНЫХ ПРОГРАММ**

В начале программы должна указываться директива ORG 100H, указывающая адрес первой инструкции (ВСЕГДА начинается со 100h). Область адресов менее 100h используется для служебной информации.

**ПРОГРАММА 1**

(Загрузка требуемых данных в два заданных РОНа и в заданные

ячейки памяти)

- Записать заданные шестнадцатеричные числа в регистры С и D.

- Переписать содержимое регистров в память по заданным адресам.

- Продемонстрировать расположение заданных чисел в заданных ячейках памяти.

**ПРОГРАММА 2**

(Работа процессора при обращении к стеку)

- Инициализировать стек.

- Записать первое заданное число в старший байт регистра ВХ (BH).

- Записать второе заданное число в младший байт регистра ВХ (BL).

- Записать в стек по двум заданным адресам заданные числа.

- Извлечь данные в старший и младший байты регистра DX.

- Инкрементировать DH и декрементировать DL

- Запомнить модифицированные данные в стеке по выбранным ранее адресам.

**ПРОГРАММА 3**

(Суммирование одного байта содержимого аккумулятора с одним байтом

содержимого ячейки памяти)

- Записать в Рг ВХ заданный адрес ячейки памяти.

- Записать по данному адресу в память заданное число.

- Суммировать заданное содержимое аккумулятора с содержимым ячейки памяти.

- Результат сохранить в памяти по изначально заданному адресу.

Обратить внимание на изменение результата при повторении суммирования в автоматическом режиме

**ПРОГРАММА 3.1**

(Вычитание из двухбайтного содержимого аккумулятора

двухбайтного содержимого ячейки памяти)

По аналогии с ПРОГРАММОЙ 3 произвести вычитание из двухбайтного содержимого аккумулятора двухбайтного содержимого ячейки памяти.

**ПРОГРАММА 3.2**

(Сложение двоично-десятичных чисел)

Разработать программу сложения двоично-десятичных чисел размером в байт. Рассмотреть действие команды DAA для различных значений суммируемых операндов. Результат зафиксировать в памяти.

**ПРОГРАММА 3.3**

(Вычитание двоично-десятичных чисел)

Разработать программу вычитания двоично-десятичных чисел размером в байт. Учесть, что команда «DAA» не корректирует результат вычитания двоично-десятичных чисел. Вычитание эквивалентно сложению в дополнительных кодах (5 – 3 = 5 + (–3)). Дополнительный код положительного числа есть само число, а дополнительный код отрицательного числа – дополнение до максимально возможного числа (при заданном числе разрядов) плюс единица.

То есть для отрицательного числа: Nдоп = (99 – N) + 1.

Получить результат вычитания, используя алгоритм сложения двоично-десятичных чисел.

**ПРОГРАММА 3.4**

(сложение двух 32-разрядных чисел)

(**16-разрядные числа сохраняются по адресам, кратным 2**)

Желательно выбирать числа с переносом между младшим и старшим словами.

Разработать программу сложения двух заданных 32-разрядных двоичных чисел. Объяснить назначение команды «ADC». Операнды команды изначально разместить в четырёх ячейках памяти. Результат зафиксировать в памяти.

**ПРОГРАММА 3.5**

(вычитание двух 32-разрядных чисел)

(16-разрядные числа сохраняются по адресам, кратным 2)

Желательно выбирать числа с займом из старшего слова.

Выполняется аналогично программе 3.4, но вместо команды «ADC» используется команда «SBB» (вычитание с займом: operand1 = operand1 – operand2 – CF). Результат зафиксировать в памяти.

**ПРОГРАММА 4**

(Обнуление области памяти)

Приведённая ниже программа реализует автоматическое обнуление оперативной памяти.

sub ax, ax ; Обнуление АКК.

mov bx, 0180h

label1:

mov [bx], ax

inc bx

jmp label1

Рассмотреть выполнение программы в автоматическом и (фрагментарно) в пошаговом режиме. Обратить внимание на стирание части программы. Объяснить подобное стирание. Записать в отчёт оставшуюся часть программы.

Обратить внимание на некорректность программы (отсутствие Останова).

**ПРОГРАММА 4.1**

(Загрузка памяти заданным значением с указанного по указанный адрес)

Видоизменить ПРОГРАММУ 4 для занесения в память заданной комбинации с указанного по указанный в программе адрес.

**ПРОГРАММА 5**

(Формирование Слова Состояния Программы (PSW) при выполнении различных групп команд МП)

Программу выполнять в пошаговом режиме.

ВНИМАНИЕ. В отличие от PSW процессора K580, PSW процессора i8086 содержит только флаги (**не содержит аккумулятор**):

РгF = [ x x x x OF DF IF TF | SF ZF x AF x PF x CF ]

Формирование PSW изучить на примере выполнения программы:

mov ax, 8000h

mov bx, 0001h

xor ax, bx

cmc

push ax

pushf

pop cx

pop dx

end:jmp end

Проанализировать значение Слова Состояния Программы, формируемое при выполнении каждой команды. Результат представить в виде:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| команда | ax | bx | cx | dx | flags | sp |
| mov ax, 0х8000h | 8000 | 0000 | 0000 | 0000 | f244 | 0100 |
| mov bx, 0х1 | 8000 | 0001 | 0000 | 0000 | f244 | 0100 |
| xor ax, bx | 8001 | 0001 | 0000 | 0000 | f280 | 0100 |
| и т.д. |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

CF = 1: перенос из старшего разряда или займ в старший разряд при вычитании

PF = 1: в младшем байте результата чётное число единиц

AF = 1: межтетрадный перенос при двоично-десятичной арифметике

ZF = 1: результат операции равен нулю

SF = 1: результат отрицательный (старший разряд =1)

OF = 1: переполнение разрядной сетки

1. Команды «**mov ax, 0х8000**h» и «**mov bx, 0х1**» не влияют на флаги (остаются флаги предыдущих команд):

f 2 4 4

Флаговый РгF = 1 1 1 1 0 0 1 0 0 1 0 0 0 1 0 0

x x x x OF DF IF TF SF ZF x AF x PF x CF

Аккумулятор = 10000000 00000000

2. Команда «**xor ax, bx**» (РгА ИСКЛ. ИЛИ РгВ) влияет на флаги SF, ZF, PF. После неё:

И так далее.

**ПРОГРАММА 6**

(Проверка действия команд условного перехода)

Программу выполнять в пошаговом режиме

Во флаговый Рг заносятся коды либо ветвления в цикле, либо выхода из него, и проверяется выполнение команд условного перехода: jnz, jz, jnc, jc, jpo, jpe, jns, js.

jpo – если PF = 0; jpe – если PF = 1; jns – если знак = 0 (+), js – если знак = 1 (–)

Действия восьми приведённых команд рассмотреть на примере приведённой ниже программы (рассмотрена команда jnz). Продемонстрировать работу программы для всех восьми команд в пошаговом режиме. Для каждой из них рассмотреть вариант ветвления, а также выхода из цикла.

mov byte ah, 0x02 ; Зп в сташий байт АКК числа 02h.

cycle:

sahf ; Записать данные из регистра АН в младшие 8 битов регистра флагов (ZF = 0).

; Биты АН: 7 6 5 4 3 2 1 0

; [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]

; биты 1, 3, 5 зарезервированы.

jnz cycle

end:jmp end

**ПРОГРАММА 7**

(Вызов подпрограммы и возврат из неё)

Программу выполнять в пошаговом режиме

Самостоятельно разработать программу с вызовом подпрограммы и возвратом из неё. Использовать команды: «mov sp», «call subroutine» и «ret». В подпрограмме выполнить любую арифметическую операцию с сохранением в памяти результата.

**ПРОГРАММА 8**

(Действие команд INT и IRET при работе с прерывающими программами)

Программу выполнять в пошаговом режиме

ВНИМАНИЕ. Прерывания с номерами от 0 до 1Fh зарезервированы, можно использовать прерывания с номерами от 20h до FFh.

Осуществить проверку действия команд INT и IRET при заданном номере (векторе) прерывания на примере выполнения программы:

mov sp, 0x0180

mov word [0x80], int\_handler ; В ячейку с адресом 80h заносится адрес обработчика

; прерывания.

int 20h ; Генерируется прерывание. Адрес ячейки, содержащий адрес

; обработчика прерывания: 20h \* 4 = 80h.

nop ; Для наблюдения работы программы.

end:jmp end

int\_handler: ; Обработчик прерывания.

nop ; Для наблюдения работы программы.

nop

iret ; Возврат из обработчика прерывания.

**ПРОГРАММА 9**

(Прерывающие друг друга вложенные программы)

Программу выполнять в пошаговом режиме

Осуществить проверку действия команд INT и IRET при условии наличия нескольких прерывающих друг друга вложенных программ.

Проверку действия команд INT и IRET осуществить на примере выполнения программы:

mov sp, 0x0180 ; Инициализация указателя стека.

; Адрес ячейки, содержащий адрес обработчика прерывания, определяется:

; address = № прерывания \* 4.

mov word [0x80], sub1 ; Зп в ячейку с адресом 0x80 адреса подпрограммы 1 (20h \* 4 = 80h).

mov word [0x84], sub2 ; Зп в ячейку с адресом 0x84 адреса подпрограммы 2 (21h \* 4 = 84h).

mov word [0x88], sub3 ; Зп в ячейку с адресом 0x88 адреса подпрограммы 3 (22h \* 4 = 88h).

mov word [0x8C], sub4 ; Зп в ячейку с адресом 0x8C адреса подпрограммы 4 (23h \* 4 = 8Ch).

mov word [0x90], sub5 ; Зп в ячейку с адресом 0x90 адреса подпрограммы 5 (24h \* 4 = 90h).

int 20h ; Вызов подпрограммы 1

nop ; Для наблюдения работы программы.

end:jmp end ; Останов.

sub1: ; Подпрограмма 1

int 21h ; Вызов подпрограммы 2

iret ; Возврат из подпрограммы 1

sub2: ; Подпрограмма 2

int 22h ; Вызов подпрограммы 3

iret ; Возврат из подпрограммы 2

sub3: ; Подпрограмма 3

int 23h ; Вызов подпрограммы 4

iret ; Возврат из подпрограммы 3

sub4: ; Подпрограмма 4

int 24h ; Вызов подпрограммы 5

iret ; Возврат из подпрограммы 4

sub5: ; Подпрограмма 5

iret ; Возврат из подпрограммы 5

**ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ ВЫПОЛНЕНИЯ**

**ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ**

**Сортировка «пузырьком»**

Используется при обработке одномерных массивов. Одномерный массив – конечный набор элементов одинаковой длины, размещённых в смежных ячейках памяти. Положение каждого элемента в массиве определяется его номером (индексом). Полный адрес элемента массива складывается из начального (базового) адреса массива и индекса, умноженного на длину элемента в байтах.

Простейший из многочисленных методов сортировки – метод пузырьковой сортировки. Его суть – в циклическом последовательном просмотре всех элементов массива, сравнении значений соседних элементов и их обмене местами, если они расположены в порядке, отличном от заданного.

После первого просмотра всего массива наибольший (наименьший) элемент находится в нужной позиции. Итерации повторяются до расположения всех элементов в заданной последовательности.

Недостаток подобного способа обмена – наличие большого числа сравнений, что замедляет всю процедуру. Для ускорения процесса необходимо ввести дополнительную переменную – *число перестановок при каждом просмотре*. Данная переменная должна уменьшаться по мере последовательного прохождения итераций и сбрасываться в ноль после каждого просмотра массива. Как только она достигает нуля (т.е. перестановок нет) процедура сортировки останавливается.

Данная методика используется в нескольких индивидуальных заданиях.

**Деление массива на массивы положительных и отрицательных чисел**

Выполнение программы начать с заполнения памяти заданными элементами массива. Для хранения результирующих массивов выделить две другие области памяти.

Число считается положительным при условии, что его старший разряд равен нулю, и считается отрицательным, если он равен единице. Деление на массивы положительных и отрицательных чисел осуществляется на основании анализа состояния соответствующего флага и занесении анализируемого числа в тот или иной массив.

При выполнении данной программы необходимо с осторожностью относиться к использованию команд «jns» (ветвление, если число положительное) и «js» (ветвление, если число отрицательное), поскольку число с большим модулем (достигающим старшего разряда) может быть воспринято как отрицательное. Использование данных команд допустимо лишь в случае, если модули чисел не достигают старших разрядов.

**Умножение двух чисел младшими разрядами вперёд**

**со сдвигом частичных сумм вправо и неподвижным множимым**

Рг AL

(множимое)

Рг СХ 🡪

(частичные суммы)

Рг DL 🡪

(множитель)

СМ

Устройство

управления

Ст

Операнды из ОП

У1

(обнул.)

У2 (ЗП сумм)

У3 (сдв.)

У4 (сдв.)

У1

У6

*Z*зн

У6 (-1)

У5

*n*

Если младший разряд множителя (РгDL) = 1, то производится сложение: (РгCH + РгAL) → РгСХ, если же младший разряд множителя = 0, то сложения не производится. После чего (в обоих случаях) – сдвиг всего РгСХ. Сначала РгСХ хранит частичные суммы, а после вычислений – результат (произведение). В конце программы результаты переписываются в память.

**Поиск максимального нечётного (минимального чётного) числа в массиве**

Выполнение программы начать с заполнения памяти заданными элементами массива. Один из способов решения задачи (но не единственный) заключается в её совмещении с сортировкой элементов массива по убыванию (по возрастанию). Также возможна реализация путём модификации самого алгоритма сортировки.

**Транспонирование матрицы**

Массив не обязательно отражает линейный набор элементов. Двумерный массив представляет набор данных, доступ к которым производится по двум индексам – номеру строки и номеру столбца. С помощью подобных массивов удобно задавать матрицы элементов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Столбец 1 | Столбец 2 | Столбец 3 | Столбец 4 |
| Строка 1 | *a*11 | *a*12 | *a*13 | *a*14 |
| Строка 2 | *a*21 | *a*22 | *a*23 | *a*24 |
| Строка 3 | *a*31 | *a*32 | *a*33 | *a*34 |
| Строка 4 | *a*41 | *a*42 | *a*43 | *a*44 |

Однако физическое представление данных отличается от логического. Это связано с линейной организацией памяти, в которой хранятся элементы матрицы. То есть последовательность ячеек памяти является одномерным массивом, в котором нужно расположить двумерный массив элементов матрицы.

В данном случае используется отображение двумерного массива в одномерной памяти либо по столбцам, либо по строкам:

|  |  |  |
| --- | --- | --- |
| *a*11 |  | *a*11 |
| *a*21 |  | *a*12 |
| *a*31 |  | *a*13 |
| *a*41 |  | *a*14 |
| *a*12 |  | *a*21 |
| *a*22 |  | *a*22 |
| *a*32 |  | *a*23 |
| *a*42 |  | *a*24 |
| *a*13 |  | *a*31 |
| *a*23 |  | *a*32 |
| *a*33 |  | *a*33 |
| *a*43 |  | *a*34 |
| *a*14 |  | *a*41 |
| *a*24 |  | *a*42 |
| *a*34 |  | *a*43 |
| *a*44 |  | *a*44 |

Отображение по столбцам Отображение по строкам

Транспонирование матрицы подразумевает перестановку элементов строк *a*ij и столбцов *a*ji, в результате которой строки и столбцы меняются местами. Транспонировать можно не только квадратные матрицы (с равным числом строк и столбцов).

Пример транспонирования квадратной матрицы 3\*3:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *a*11 | *a*12 | *a*13 | транспонирование | *a*11 | *a*21 | *a*31 |
| *a*21 | *a*22 | *a*23 | *a*12 | *a*22 | *a*32 |
| *a*31 | *a*32 | *a*33 | *a*13 | *a*23 | *a*33 |

Элементы исходной (не транспонированной) матрицы и элементы транспонированной матрицы находятся в одномерной памяти. Чаще используется отображение по строкам.

Для упрощения программирования рекомендуется выбирать значения элементов матрицы: *a*11 = 01h; *a*12 = 02h; … *a*32 = 08h; *a*33 = 09h.

**Числа Фибоначчи**

Числа Фибоначчи – элементы числовой последовательности, в которой первые два числа раны единице, а каждый последующий член равен сумме двух предыдущих, то есть:

*а*1 = *а*2 =1;

*а*3 = *а*2 + *а*1 = 2;

*а*4 = *а*3 + *а*2 = 3;

*а*5 = *а*4 + *а*3 = 5;

*а*6 = *а*5 + *а*4 = 8; …

При установке счётчика циклов необходимо учитывать, что, как правило, за одну итерацию вычисляется два числа Фибоначчи. Кроме того, необходимо учитывать нестандартность первого цикла (для *а*1 и *а*2).

Необходимо помнить, что **для косвенной адресации в процессоре i8086 используются только четыре 16-разрядных регистра: SI, DI, BX, BP**.

**Подсчёт разрядов и их комбинаций в двоичных числах**

(нулей, единиц, их пар, различных переходов)

Решение подобных задач производится путём поразрядных логических операций (И, ИЛИ, …) исследуемых чисел и чисел вспомогательных. После чего производится сдвиг исследуемого числа (в зависимости от алгоритма – влево или вправо). Необходимо выбирать вспомогательные числа, ориентируясь на установку определённых флагов. Подбор данных чисел – основная задача указанного метода.

При установке конкретного флага, необходимо инкрементировать соответствующий счётчик.

При необходимости произвести инверсию какого-либо числа, можно произвести поразрядную операцию ИСКЛЮЧАЮЩЕЕ ИЛИ данного числа со всеми единицами.

**Сравнение на равенство сумм чётных и нечётных разрядов**

Выполнение данной задачи (как и задачи предыдущей) предполагает использование поразрядных логических операций (И, ИЛИ, …) исследуемых чисел и чисел вспомогательных. После чего производится сдвиг исследуемого числа. По мере выполнения программы, осуществляется инкрементирование счётчиков чётных и нечётных разрядов.

В случае равенства чётных и нечётных разрядов, по фиксированному адресу выводится условная комбинация (например, 0xaaaa), а при их неравенстве – другая условная комбинация (например, 0xffff).