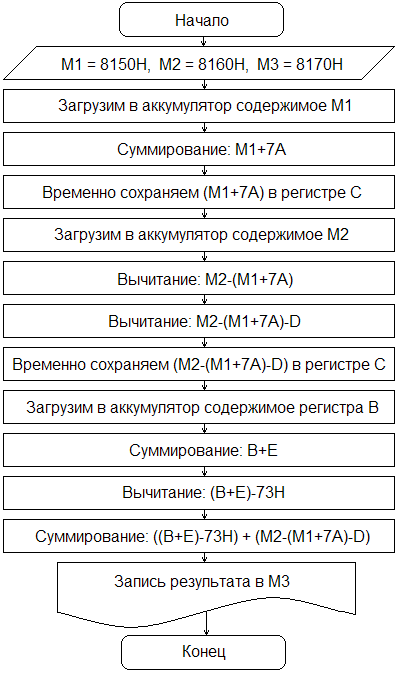
**Задача #1**

Найти значение функции M3=(B+E-73H)+M2-(M1+7AH)-D



**LDA 8150** ;M1→A

**ADI 7A** ;A:=A+7AH

**MOV C, A** ;A→C

**LDA 8160** ;M2→A

**SUB C** ;A:=A-C

**SUB D** ;A:=A-D

**MOV C, A** ;A→C

**MOV A, B** ;B→A

**ADD E** ;A:=A+E

**SUI 73** ;A:=A-73H

**ADD C** ;A:=A+C

**STA 8170** ;A→M3

**Задача #2**

Написать алгоритм задержки 255 мс ± 0,1%

|  |  |
| --- | --- |
| Поскольку 1 такт занимает 0,5 мкс, то по пропорции: | 1 такт - 0,5·10-6 с  *Х* тактов - 255·10-3 с |

 тактов.

Учитывая погрешность 0,1%, результат должен составить 510000 ± 510 тактов.

Для такого кол-ва тактов будем использовать не один регистр, а регистровую пару DE.

Код задержки будет таким:

**LXI D, Y**

**CALL DEL**

**… … …**

**DEL: DCX D** ;5 тактов

**MOV A, D** ;5 тактов

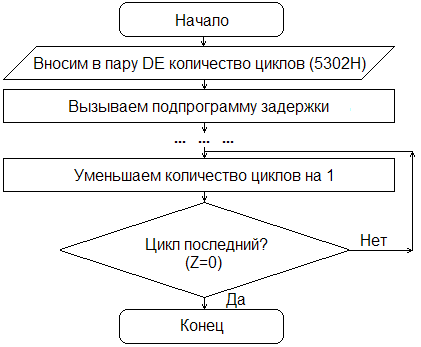
**ORA D** ;4 такта

**JNZ DEL** ;10 тактов, если условие выполняется, и 7 тактов, если не выполняется

**RET** ;10 тактов

Чтобы найти Y воспользуемся формулой: 510000 = Y(5(DCX)+5(MOV)+4(ORA)+10(JNZ)) + 10(RET).

510000 = 24·Y + 10 → Y = (510000–10) / 24 → Y = 21249,58 ≈ 21250 циклов. 2125010 = 530216.



В программе использована команда ORA D, которая не влияет на содержимое аккумулятора, но влияет на флаг нулевого результата Z, по которому мы судим об окончании.

Окончательный код задержки будет выглядеть следующим образом:

**LXI D, 5302** ;10 тактов

**CALL DEL** ;17 тактов

**… … …**

**DEL: DCX D** ;5 тактов

**MOV A, D** ;5 тактов

**ORA D** ;4 такта

**JNZ DEL** ;10 тактов, если условие выполняется, и 7 тактов, если не выполняется

**RET** ;10 тактов

В результате общее количество тактов будет: 24(один цикл)·21250 + 10(RET) – 3(JNZ) + 10(LXI) + 17(CALL) = 510034 тактов.

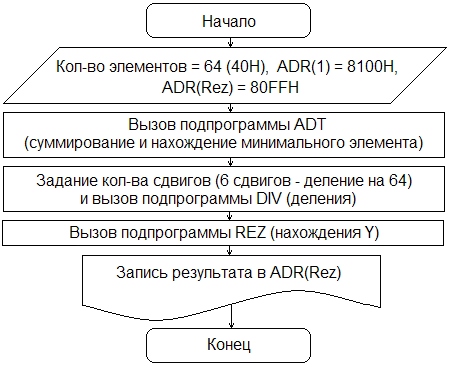
Погрешность составит: .

**Задача #3**

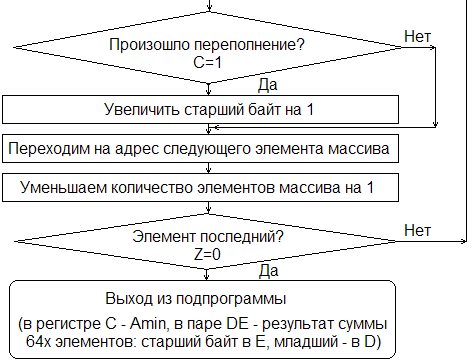
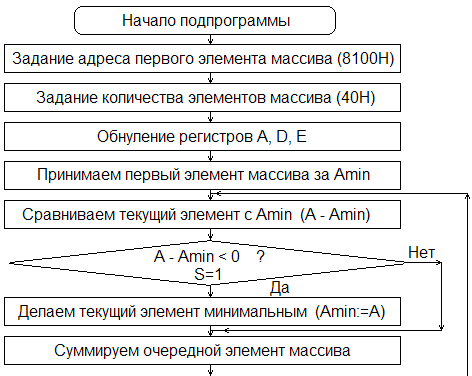
Вычислить **,** где *АСР* и *АMIN* – среднее арифметическое и минимальное значение массива однобайтных чисел объёмом в 64 элемента.

Значения *АСР* и *АMIN* могут быть в диапазоне: . Следовательно, *Y* может быть в диапазоне: . Как видно, результат умещается в один регистр (байт).

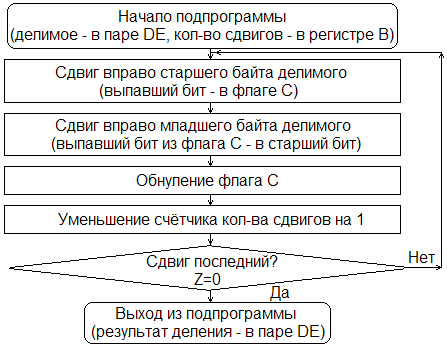
Общий алгоритм программы выглядит так:



Алгоритм подпрограммы ADT (суммирования и нахождения минимального элемента):

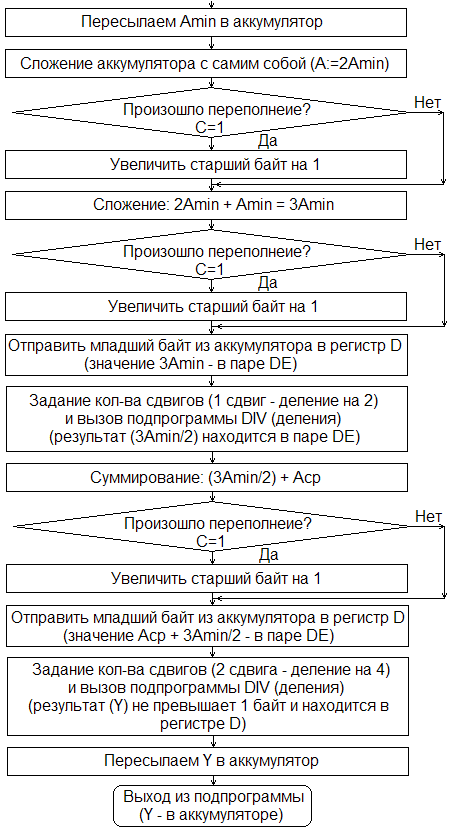
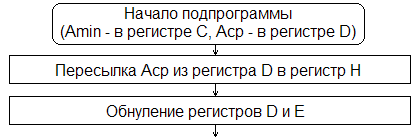


Алгоритм подпрограммы DIV (деления через циклический сдвиг вправо с переносом):



Алгоритм подпрограммы REZ (нахождение значения Y):

<!-- так как среднее значение (*АСР*) не превышает один байт, то оно находится не в паре DE, а в регистре D; это значение будет переправлено в регистр Н (пара HL уже не используется), чтобы освободить пару DE -->



Листинг программы расположен ниже:

**CALL ADT**  ;вызов подпрограммы ADT (суммирование и нахождение минимального

элемента)

**MVI B, 06**  ;задание кол-ва сдвигов (B:=06H – 6 сдвигов равносильно

делению на 64)

**CALL DIV**  ;вызов подпрограммы DIV (нахождение среднего значения)

**CALL REZ** ;вызов подпрограммы REZ (вычисление результата Y)

**STA 80FF ;**пересылаем значение Y (окончательный результат) в ячейку 80FFH

**RST 1**  ;выход из программы

**ADT: LXI H, 8100** ;задание адреса первого элемента (HL:=8100H)

**MVI B, 40**  ;задание кол-ва элементов (B:=64D=40H)

**XRA A**  ;обнуление аккумулятора

**LXI D, 0000** ;обнуление регистров E и D (пары DE) – старшего и младшего

байтов результата суммирования соответственно

**MOV** **C, M ;**будем считать первый элемент минимальным (С:=M(HL))

**X1: MOV A, M**  ;переслать в аккумулятор текущий элемент (A:=M(HL))

**CMP C** ;сравниваем содержимое аккумулятора с текущим наименьшим

значением (A-C)

**JP X2** ;при TS=0 (A-C≥0 → A≥C) переход на Х2

**MOV C, A** ;если же TS=1 (A-C<0 → A<C), сделать текущий элемент наименьшим

**X2: ADD D** ;суммирование (A:=D+A)

**MOV D, A** ;пересылка A→D

**JNC X3**  ;перейти на Х3 если нет переполнения

**INR E**  ;произошло переполнение → прибавить 1 к старшему байту

результата суммирования

**X3: INX H**  ;присвоить HL адрес следующей ячейки (HL:=HL+1)

**DCR B**  ;уменьшение счётчика кол-ва элементов на 1 (B:=B-1)

**JNZ X1**  ;если элемент не последний - продолжить суммирование

**RET**  ;выход из подпрограммы ADT

**DIV:** **MOV A, E** ;пересылаем старший байт в аккумулятор (E→A)

**RAR** ;циклический сдвиг вправо через ТС

**MOV E, A** ;возврат в Е старшего байта

**MOV A, D** ;пересылаем младший байт в аккумулятор (D→A)

**RAR** ;циклический сдвиг вправо через ТС

**MOV D, A** ;возврат в D младшего байта

**ORA A** ;обнуление флага переполнения (ТС:=0)

**DCR B**  ;уменьшение счётчика кол-ва сдвигов на 1 (B:=B-1)

**JNZ DIV**  ;если сдвиг не последний – продолжить

**RET**  ;выход из подпрограммы DIV

**REZ: MOV Н, D** ;пересылаем среднее значение в регистр Н

**LXI D, 0000** ;обнуление регистров E и D (пары DE) - старшего и младшего

байтов результата Y соответственно

**MOV A, C ;**пересылаем минимальное значение в аккумулятор (C→A)

**ADD A** ;суммирование (A:=AMIN+AMIN=2AMIN)

**JNC X4** ;перейти на Х4 если нет переполнения

**INR E ;**произошло переполнение → прибавить 1 к старшему байту

результата Y

**X4: ADD C ;**суммирование (A:=2AMIN+AMIN=3AMIN)

**JNC X5** ;перейти на Х5 если нет переполнения

**INR E ;**произошло переполнение → прибавить 1 к старшему байту

результата Y

**X5: MOV D, A** ;пересылаем младший байт из аккумулятора в регистр D

**MVI B, 01**  ;задание кол-ва сдвигов (B:=01H – 1 сдвиг равносилен

делению на 2)

**CALL DIV**  ;вызов подпрограммы DIV (деление на 2)

**MOV A, Н** ;пересылаем среднее значение в аккумулятор (Н→A)

**ADD D** ;суммирование АСР + младший байт (3АMIN/2)

**JNC X6** ;перейти на Х6 если нет переполнения

**INR E ;**произошло переполнение → прибавить 1 к старшему байту

результата Y

**X6: MOV D, A** ;пересылаем младший байт из аккумулятора в регистр D

**MVI B, 02**  ;задание кол-ва сдвигов (B:=02H – 2 сдвига равносильно

делению на 4)

**CALL DIV**  ;вызов подпрограммы DIV (деление на 4)

**MOV A, D** ;пересылаем значение Y в регистр аккумулятор (D→A)

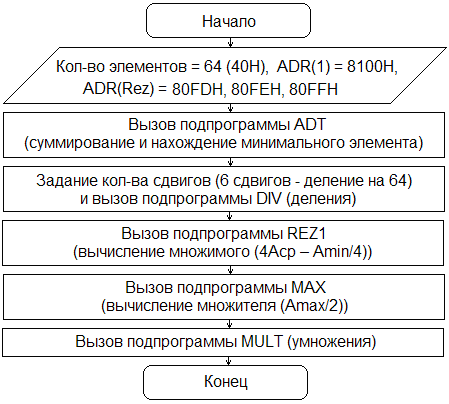
**RET**  ;выход из подпрограммы REZ

**Задача #4**

Вычислить **,** где *АСР*, *AMAX* и *АMIN* – среднее арифметическое, максимальное и минимальное значение массива однобайтных чисел объёмом в 64 элемента.

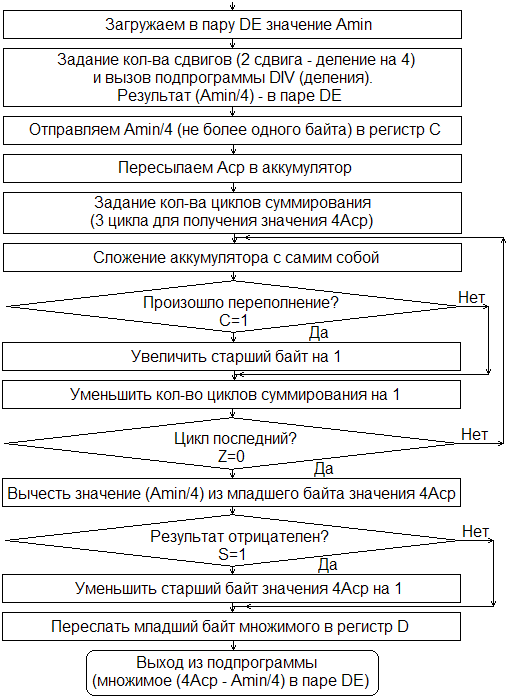
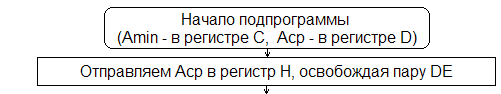
Значения *АСР*, *AMAX* и *АMIN* могут быть в диапазоне: . Минимальное значение (*Y=00*) может быть при *АСР* = *AMAX* = *АMIN = 00.* Максимальное значение будет тогда, когда массив будет состоять из 63х элементов, равных *FF*, и одного элемента, равного *00*; при этом: *AMAX = FF*, *АMIN = 00*, *АСР = FC*. В таком случае значение *Y* составит *1F41016*, что выходит за пределы пары регистров и умещается в три байта. Значение *Y* будет располагаться в ячейках памяти *80FFH*, *80FEH*, *80FDH,* причем младший байт – в ячейке *80FDH*, самый старший – в ячейке *80FFH*.

Общий алгоритм программы выглядит так:

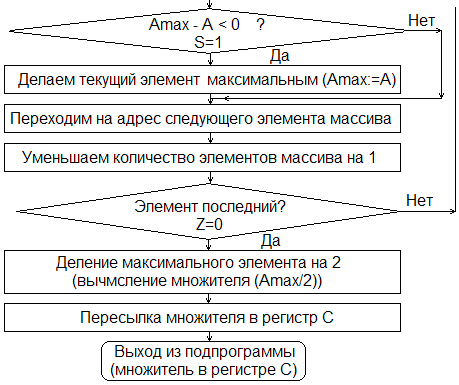
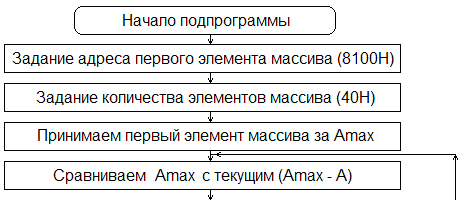


В программе используются подпрограммы ADT и DIV, алгоритм которых можно увидеть в **задаче #3**. Подпрограмма деления (DIV) идеально подходит и для **задачи #4**, а подпрограмму суммирования и нахождения минимального элемента массива (ADT) можно было бы дополнить нахождением и максимального элемента массива, если бы у на был хотя бы ещё один неиспользуемый регистр. Выйти из этой ситуации можно было бы используя ячейки ОЗУ для хранения промежуточных значений *AMAX* и *АMIN*. Но, так как *AMAX* требуется для расчёта лишь множителя, в программе была использована дополнительная подпрограмма нахождения максимального элемента и вычисления множителя (MAX).

Алгоритм подпрограммы REZ1 (вычисления множимого):

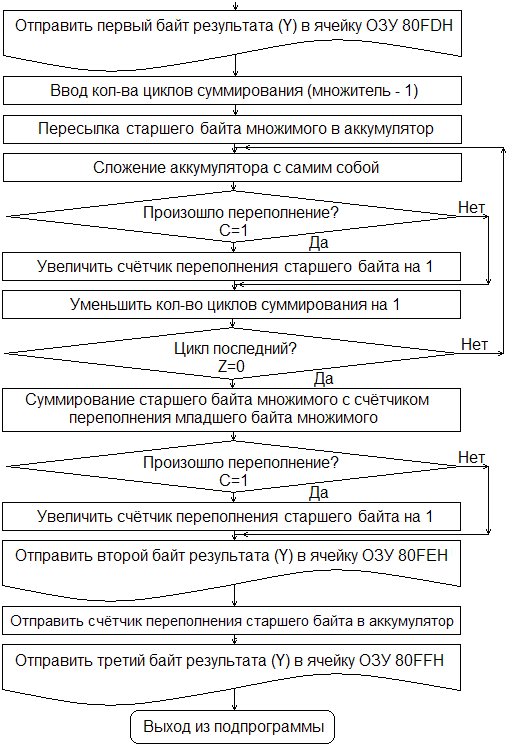
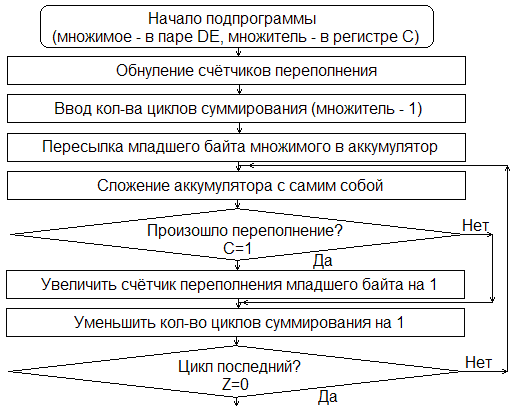


Алгоритм подпрограммы MAX (нахождение максимального элемента и вычисление множителя):



Алгоритм подпрограммы MULT (умножения и записи результата (*Y*) в ОЗУ):

<!-- Операция умножения реализована посредством многократного суммирования (кол-во циклов суммирования = множитель-1). Поскольку множимое состоит из двух байт (находится в паре DE), то суммирование производится сначала для младших байтов (счётчик переполнения – в регистре H), а потом для старших байтов (счётчик переполнения – в регистре L). Таким образом, первый (младший) байт *Y* находится в регистре D, второй байт – состоит из суммы старшего байта множимого (E) и счётчика переполнения младшего байта множимого (H), третий (старший) байт – в регистре L -->



Листинг программы приведён ниже:

**CALL ADT**  ;вызов подпрограммы ADT (суммирование и нахождение

минимального элемента)

**MVI B, 06**  ;задание кол-ва сдвигов (B:=06H – 6 сдвигов равносильно

делению на 64)

**CALL DIV**  ;вызов подпрограммы DIV (нахождение среднего значения)

**CALL REZ1** ;вызов подпрограммы REZ1 (вычисление (4ACP – AMIN/4))

**CALL MAX** ;вызов подпрограммы MAX (вычисление AMAX/2)

**CALL MULT** ;вызов подпрограммы MULT (умножение)

**RST 1**  ;выход из программы

**ADT: LXI H, 8100** ;задание адреса первого элемента (HL:=8100H)

**MVI B, 40**  ;задание кол-ва элементов (B:=64D=40H)

**XRA A**  ;обнуление аккумулятора

**LXI D, 0000** ;обнуление регистров E и D (пары DE) – старшего и младшего

байтов результата суммирования соответственно

**MOV** **C, M ;**будем считать первый элемент минимальным (С:=M(HL))

**X1: MOV A, M**  ;переслать в аккумулятор текущий элемент (A:=M(HL))

**CMP C** ;сравниваем содержимое аккумулятора с текущим наименьшим

значением (A-C)

**JP X2** ;при TS=0 (A-C≥0 → A≥C) переход на Х2

**MOV C, A** ;если же TS=1 (A-C<0 → A<C), сделать текущий элемент

наименьшим

**X2: ADD D** ;суммирование (A:=D+A)

**MOV D, A** ;пересылка A→D

**JNC X3**  ;перейти на Х3 если нет переполнения

**INR E**  ;произошло переполнение → прибавить 1 к старшему байту

результата суммирования

**X3: INX H**  ;присвоить HL адрес следующей ячейки (HL:=HL+1)

**DCR B**  ;уменьшение счётчика кол-ва элементов на 1 (B:=B-1)

**JNZ X1**  ;если элемент не последний - продолжить суммирование

**RET**  ;выход из подпрограммы ADT

**DIV:** **MOV A, E** ;пересылаем старший байт в аккумулятор (E→A)

**RAR** ;циклический сдвиг вправо через ТС

**MOV E, A** ;возврат в Е старшего байта

**MOV A, D** ;пересылаем младший байт в аккумулятор (D→A)

**RAR** ;циклический сдвиг вправо через ТС

**MOV D, A** ;возврат в D младшего байта

**ORA A** ;обнуление флага переполнения (ТС:=0)

**DCR B**  ;уменьшение счётчика кол-ва сдвигов на 1 (B:=B-1)

**JNZ DIV**  ;если сдвиг не последний – продолжить

**RET**  ;выход из подпрограммы DIV

**REZ1: MOV Н, D** ;пересылаем ACP в регистр Н

**MVI E, 00 ;**обнуление регистра Е

**MOV D, C ;**пересылка АMIN в регистр D

**MVI B, 02**  ;задание кол-ва сдвигов (B:=02H – 2 сдвигa равносильно

делению на 4)

**CALL DIV**  ;вызов подпрограммы DIV (деление на 4)

**MOV C, D** ;обратная пересылка в регистр С значения АMIN/4

**MOV A, H ;**отправляем ACP в аккумулятор

**MVI E, 00 ;**обнуление регистра E

**MVI B, 03 ;**записываем в регистр В количество циклов суммирования

(03 = 4 - 1)

**X4: ADD H ;**к аккумулятору прибавляем по одному значению ACP

**JNC X5 ;**если нет переполнения – перейти на Х5

**INR E ;**если произошло переполнение – увеличить старший байт на 1

**X5: DCR B ;**уменьшить кол-во циклов суммирования на 1

**JNZ X4 ;**если цикл не последний – повторить суммирование

**SUB C ;**вычитание: 4АСР – AMIN/4

**JP X6 ;**если результат вычитания положителен – перейти на Х6

**DCR E ;**если результат вычитания отрицателен – вычесть 1 из

старшего байта

**X6: MOV D, A ;**переслать младший байт результата (4АСР – AMIN/4) в регистр

D (весь результат находится в паре DE)

**RET**  ;выход из подпрограммы REZ1

**MAX: LXI H, 8100** ;задание адреса первого элемента (HL:=8100H)

**MVI B, 40**  ;задание кол-ва элементов (B:=64D=40H)

**MOV** **A, M ;**будем считать первый элемент максимальным (A:=M(HL))

**X7: CMP M** ;сравниваем максимальный элемент с текущим (A-M(HL))

**JP X8** ;при TS=0 (A-M(HL)≥0 → A≥M(HL)) переход на Х8

**MOV A, M** ;если же TS=1 (A-M(HL)<0 → A<M(HL)), сделать текущий

элемент максимальным

**X8: INX H**  ;присвоить HL адрес следующей ячейки (HL:=HL+1)

**DCR B**  ;уменьшение счётчика кол-ва элементов на 1 (B:=B-1)

**JNZ X7**  ;если элемент не последний - продолжить суммирование

**RAR** ;деление AMAX на 2

**ORA A** ;обнуление флага переполнения (ТС:=0)

**MOV C, A** ;пересылка значения AMAX/2 в регистр C

**RET**  ;выход из подпрограммы MAX

**MULT: LXI H, 0000 ;**обнуление регистров H и L

**MOV B, C** ;загрузить множитель (AMAX/2) в регистр В

**DCR B**  ;уменьшение на 1 множителя

**MOV A, D** ;пересылка младшего байта множимого в аккумулятор

**X9: ADD D**  ;суммируем по одному значению

**JNC X10**  ;если нет переполнения – перейти на Х10

**INR H**  ;если было переполнение – увеличить счётчик переполнения

младшего байта на 1

**X10: DCR B**  ;уменьшить кол-во циклов суммирования на 1

**JNZ X9**  ;если цикл не последний – повторить суммирование

**STA 80FD**  ;отправить первый (самый младший) байт результата (Y) в

ячейку ОЗУ 80FDH

**MOV B, C** ;загрузить множитель (AMAX/2) в регистр В

**DCR B**  ;уменьшение на 1 множителя

**MOV A, E** ;пересылка старшего байта множимого в аккумулятор

**X11: ADD E**  ;суммируем по одному значению

**JNC X12**  ;если нет переполнения – перейти на Х12

**INR L**  ;если было переполнение – увеличить счётчик переполнения

старшего байта на 1

**X12: DCR B**  ;уменьшить кол-во циклов суммирования на 1

**JNZ X11**  ;если цикл не последний – повторить суммирование

**ADD H** ;суммирование старшего байта множимого с счётчиком

переполнения младшего байта множимого

**JNC X13** ;если нет переполнения – перейти на Х13

**INR L** ;если было переполнение – увеличить счётчик переполнения

старшего байта на 1

**X13: STA 80FE**  ;отправить второй байт результата (Y) в ячейку ОЗУ 80FDH

**MOV A, L** ;переслать третий байт результата (Y) в аккумулятор

**STA 80FF**  ;отправить третий байт (самый старший) результата (Y) в

ячейку ОЗУ 80FFH

**RET** ;выход из подпрограммы MULT

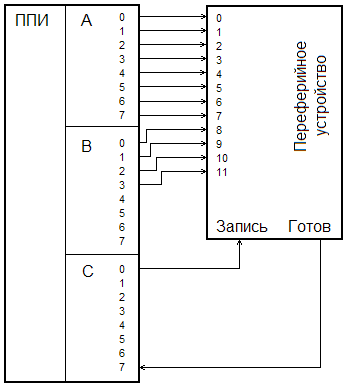
**Задача #5**

Разработать драйвер вывода информации (разрядностью 12 бит) на внешнее устройство с помощью параллельного интерфейса ППИ (К580ВВ55).

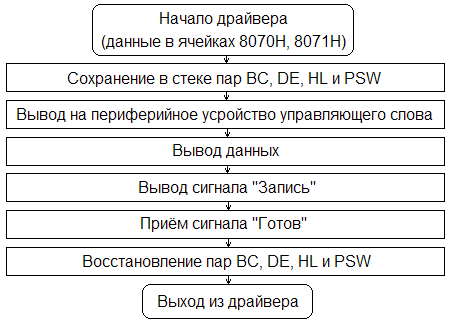
Поскольку разрядность данных составляет 12 бит, для их вывода понадобится порт А и порт В. Отправлять сигнал «Запись» и принимать сигнал «Готов» будем портом С. Следовательно, управляющее слово будет таким: 100010002 = 8816.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 00 | 0 | 1 | 0 | 0 | 0 |
|  | Режим работы группы А - 0 | Направление порта А - вывод | Направление порта С(7-4) - ввод | Группа В | Направление порта В - вывод | Направление порта С(3-0) - вывод |

Схема обмена данными ППИ с периферийным устройством:



Алгоритм работы драйвера:



**DRV: PUSH B** ;сохранение с стеке пары ВС

**PUSH D** ;сохранение с стеке пары DE

**PUSH H** ;сохранение с стеке пары HL

**PUSH PSW** ;сохранение с стеке аккумулятора и флагов

**MVI A, 10001000**

**OUT <Пер. Устр.>** ;вывод управляющего слова на периферийное устройство

**LXI H, 8070** ;загрузка в пару HL адреса младшего байта

(1-8й бит) данных

**MOV A, M**

**OUT PORTA** ;вывод младшего байта в порт А

**INX H** ;перейти на адрес старшего байта (9-12й бит) данных

**MOV A, M**

**OUT PORTB** ;вывод старшего байта в порт В

**MVI A, 00000001**

**OUT PORTC** ;вывод сигнала «Запись» на порт С(0)

**MVI A, 00000000**

**OUT PORTC** ;обнуление значения порта С

**X1: IN PORTC** ;приём сигнала портом С

**RAL** ;перемещение старшего бита пришедшего сигнала

с порта С(7) в ТС

**JNC X1** ;если ТС=0 (сигнал не пришёл) - продолжить ожидание

**POP PSW** ;восстановление аккумулятора и флагов

**POP H** ;восстановление пары HL

**POP D** ;восстановление пары DE

**POP B** ;восстановление пары BC

**RET** ;выход из драйвера