Программирование на языке ассемблер (Assembler)

1) Структура команд ассемблера.

Язык программирования наиболее полно учитывающий особенности "родного" микропроцессора и содержащий мнемонические обозначения машинных команд называется Ассемблером. Программа, написанная на Ассемблере называется исходной программой. Далее остановимся на версии, называемой Турбо Ассемблер.

Разработка программы на Ассемблере состоит из следующих этапов:

1) Составление алгоритма в виде блок-схемы или структурного описания,

2) Ввод в ЭВМ текста исходной программы PROG.ASM с помощью редактора текстов. Имя PROG может быть произвольным, а расширение ASM - обязательно,

3) Перевод (**трансляция** или ассемблирование) исходной программы в машинные коды с помощью транслятора TASM.EXE. На этом этапе получается промежуточный продукт PROG.OBJ (объектный код). Выявленные при этом синтаксические и орфографические ошибки исправляются повтором

4) Преобразование с помощью программы TLINK.EXE объектного кода PROG.OBJ в выполнимый код PROG.EXE или PROG.COM.

5) Выполнение программы и ее отладка начиная с п.1, если встретились логические ошибки.

Текст программы на Ассемблере содержит следующие операции:

1. а) **команды или инструкции**,
2. б) **директивы или псевдооператоры**,
3. в) **операторы**,
4. г) **предопределенные имена**.

Действия обусловленные операциями перечисленными в пп.б,в,г выполняются на этапе трансляции, т.е. являются командами Ассемблеру. Операции, называемые командами или инструкциями выполняются во время выполнения программы, т.е. являются командами микропроцессору.Инструкция записывается на отдельной строке и включает до четырех полей, необязательные из которых выделены [ ]:

[метка:]	мнемоника_команды	[операнд(ы)]	[;комментарий]

Метка или символический адрес- условный адрес операции.
Мнемоника - сокращенное обозначение кода операции (КОП) команды, например мнемоника ADD обозначает сложение (addition).
Операндами могут быть явно или неявно задаваемые двоичные наборы, над которыми производятся операции.Операнды приводятся в одной из четырех систем счисления и должны оканчиваться символом b(B), o(O), d(D), h(H) для 2, 8, 10 или 16-ной системы счисления.. К шестнадцатиричному числу добавляется слева ноль, если оно начинается с буквы.

Система команд может быть классифицирована по трем основным признакам -
длина команды или число занимаемых ею байтов,
функциональное назначение и
способ адресации.

Для МП 1810ВМ86 (8086) команда занимает от одного до шести байтов.

Первым байтом команды всегда является код операции, например код команды INT XXh равен CD(HEX).

По функциональному признаку их можно разбить на пять больших групп:

1. **команды пересылки данных,**
2. **арифметические команды,**
3. **логические команды,**
4. **команды переходов и**
5. **команды управления.**

Существует пять основных способов адресации:

регистровая,
непосредственная,
прямая,
косвенная и
стековая.

Большинство остальных способов адресации являются комбинациями или видоизменениями перечисленнных.

В первом случае операнд(ы) **располагаются в регистрах микропроцессора (МП**), например по команде MOV AX,CX пересылается содержимое CX в AX.
При **непосредственной адресации** операнд располагается в памяти непосредственно за КОП, инструкция MOV AL,0f5h записывает число 245(f5) в регистр AL.
В случае **прямой адресации** за КОП следует не сам операнд, а адрес ячейки памяти или внешнего устройства, например команда IN AL,40h вводит байт данных из внешнего устройства с адресом 40h.
Косвенная адресация отличается от регистровой тем, что в регистре хранится адрес операнда, т.е. по команде MOV AL,[BX] в аккумулятор al будет записано число из ячейки памяти с адресом, хранящимся в регистре BX.
Стековая адресация производится к операндам расположенным в области памяти, называемой стек.

2). Структура программы на ассемблере.

Программа, написанная на ассемблере, содержит следующие компоненты: ОПЕРАТОРЫ, ПРЕДОПРЕДЕЛЁННЫЕ ИМЕНА, ДИРЕКТИВЫ И КОМАНДЫ.

1. ПРЕДОПРЕДЕЛЕННЫЕ ИМЕНА

$ - программный счетчик. Отмечает текущий адрес в текущем сегменте.

@data - адрес начала сегмента данных.

mov ax,@data

mov ds,ax;

в сегментном регистре DS теперь адрес сегмента данных.

??date, ??time, ??filename - эти имена во время трансляции заменяются, соответственно на текущие дату, время и имя файла в формате ASCII.

2 ОПЕРАТОРЫ

1. **()** - скобки, определяют порядок вычислений
2. **[]** - например [BX] означает содержимое ячейки памяти с адресом в регистре bx. Признак косвенной адресации.
3. **+, -, *, /** - операторы сложения, вычитания, умножения и деления.

mov ax, (2 * 3 + 8 / 2) - 2; в регистр ax будет помещено число 8.

4. **MOD** - деление по модулю. Даёт остаток.

5. **SHL,SHR** - сдвиг операнда влево, вправо.

mov si, 01010101b SHR 3; в регистр SI будет загружено число 0Ah (00001010).

6. **NOT** - побитовая инверсия.

7. **AND,OR,XOR** – операции

mov dl, (10d OR 5d) XOR 7d; (dl) будет равно 8.

8. **:** - переназначение сегмента.

mov dl,[es:bx]; поместить в dl байт данных из сегмента es и отстоящий от его начала на (bx) байтов (смещение).

9. **OFFSET** - оператор получения смещения адреса относительно начала сегмента (то есть количества байтов от начала сегмента до идентификатора адреса).

mov bx, OFFSET table

ДИРЕКТИВЫ (ПСЕВДООПЕРАТОРЫ)

1. **:** - определяет близкую метку (в пределах сегмента).

2 . **=** - присваивает символическому имени значение выражения.

3. **.CODE** - определяет начало кодового сегмента, то есть сегмента, где располагаются коды программы.

4. **.DATA** - определяет начало сегмента данных.

5. **DB,DW** - директивы резервирующие один или несколько байтов: DB, или одно или несколько слов: DW.

6. **END** - обозначает конец программы.
7. **ENDM** - окончание блока или макроопределения
8. **ENDP** - обозначает конец подпрограммы.
9. **EQU** - присваивает символическому имени или строке значение выражения.
10. **LABEL** - определяет метку соответствующего типа.
11. **LOCAL** - определяет метки внутри макроопределений как локальные и в каждом макрорасширении вместо них ассемблер вставляет уникальные метки: ??XXXX, где XXXX = (0000...FFFF)h. Почему ??XXXX ? Да потому что никому не должно прийти в голову начинать символическое имя с ??, и транслятор смело может генерировать метки не боясь совпадений.
12. **MACRO** - задает макроопределение.

Swap MACRO a,b; a,b - параметры макро (ячейки памяти)
mov ax,b;данное макрооопределение позволяет делать
mov bx,a;обмен данными между ячейками памяти, в
mov a,ax;отличие от команды xchg ;
mov b,bx;нельзя mov a,b;
ENDM

Вызов этого макроса производится командой: Swap m,n

13. **.MODEL** - определяет размер памяти под данные и код программы.
.MODEL tiny;под программу,данные и стек отводится один общий сегмент (64 Kb).
14. **PROC** - определяет начало подрограммы.

Print PROC NEAR

;здесь команды подпрограммы

Print ENDP

....

call Print;вызов подпргаммы.

15. **.STACK** - определяет размер стека.
.STACK 200h; выделяет 512 байтов для стека.
16. **.RADIX base** - определяет систему счисления по умолчанию, где base - основание системы счисления: 2, 8, 10, 16.

.RADIX 8

oct   = 77; oct равно 63d.

17. **;** - начало комментария.

Рассмотрим классификацию команд.

1. КОМАНДЫ ПЕРЕСЫЛКИ

1. **MOV** DST,SRC; переслать (SRC) в (DST). Здесь и далее содержимое регистра, например регистра AL будет обозначаться - (AL) или (al), а пересылка в комментарии будет обозначаться знаком <-- .
2. **PUSH** RP; поместить на вершину стека содержимое пары регистров RP (например push bx).
3. **POP** RP; снять с вершины стека два байта и поместить в пару RP (например pop ax).
4. **XCHG** DST, SRC; поменять местами содержимое (DST) и (SRC). Оба операнда не могут быть одновременно содержимым ячеек памяти.
5. **XLAT** SRC; извлечь из таблицы с начальным адресом SRC байт данных имеющий номер от начала таблицы = (AL), и поместить его в AL. Адрес SRC должен находиться в регистре BX. Другой вариант: XLATB.
6. **IN** ACCUM, PORT; поместить в аккумулятор AL или AX байт или слово из порта с адресом PORT. Если адрес порта <= FF то адрес порта может указываться непосредственно, если адрес порта > FF, то адрес порта указывается косвенно, через содержимое регистра DX (специальная функция регистра общего назначения).

7. **OUT** PORT, ACCUM; переслать из аккумулятора AL или AX байт или слово в ВУ с символическим адресом PORT.
8. **LEA** RP,M; загрузить в регистр RP эффективный адрес (смещение) ячейки памяти с символическим адресом M.

2.АРИФМЕТИЧЕСКИЕ КОМАНДЫ

1. **ADD** DST, SRC; сложить содержимое SRC и DST и результат переслать в DST.

add al, [mem_byte]; mem_byte однобайтовая ячейка памяти
add [mem_word], dx; mem_word двухбайтовая ячейка памяти
add ch,10001010b;

2. **INC** DST; увеличить (DST) на 1 (инкремент (DST)).
3. **SUB** DST, SRC; вычесть (SRC) из (DST) и результат поместить в DST.
4. **DEC** DST; декремент (DST).
5. **CMP** DST, SRC; сравнить содержимое DST и SRC. Эта команда выполняет вычитание (SRC) из (DST) но разность не помещает в DST и по результату операции воздействует на флаги.

3. ЛОГИЧЕСКИЕ КОМАНДЫ И КОМАНДЫ СДВИГА

1. **AND** DST, SRC; поразрядное логическое "И".
2. **OR** DST, SRC; поразрядное логическое "ИЛИ".
4. **NOT** DST; инверсия всех битов приемника.
5. **TEST** DST, SRC; выполняет операцию AND над операндами, но воздействует только на флаги и не изменяет самих операндов.
6. **SHR** DST, CNT; логический сдвиг вправо, освобождающиеся слева биты заполняются нулем, крайний правый бит выталкивается во флаг CF. Операнд DST может быть ячейкой
7. **SHL** DST, CNT; логический сдвиг влево.
8. **RLC** DST, CNT; циклический сдвиг влево через перенос
9. **RRC** DST, CNT;циклический сдвиг вправо через перенос
10. **ROR** DST, CNT;циклический сдвиг влево
11. **ROL** DST, CNT;циклический сдвиг вправо

4. КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

1. **CALL** SUBR; вызов подпрограммы с адресом SUBR;
2. **RET**; возврат из подпрограммы к оператору следующему непосредственно за CALL, то есть в приведенном выше примере к MOV ..
3. **JMP** NAME; безусловный переход к команде с символическим адресом NAME.
4. **JA** NAME или **JNBE** NAME; условный переход, если, например, в результате сравнения CMP DST, SRC приемник по **абсолютной величине больше** источника, то перейти к метке name.
5. **JB** NAME или **JNAE** NAME; условный переход, если, например, в результате сравнения CMP DST, SRC приемник по **абсолютной величине меньше** источника, то перейти к метке name (команды п4 и п5 выполняются по результатам выполнения операций над **беззнаковыми** числами).
6. **JZ** NAME или **JE** NAME; перейти, если результат операции влияющей на флаг нуля - нулевой (переход по "нулю").
7. **JNZ** NAME или **JNE** NAME; переход по "не нулю". (команды п6 и п7 выполняются по результатам выполнения операций над числами со **знаком**).

5. КОМАНДЫ УПРАВЛЕНИЯ ЦИКЛАМИ

1. **LOOP** NAME; эта команда неявно уменьшает (CX) на 1 и осуществляет переход к ближней метке, если (CX) не равно 0.
2. **LOOPZ** NAME или **LOOPE** NAME кроме того осуществляет проверку ZF флага. Поэтому цикл заканчивается по условию, когда (CX) = 0 или (ZF) = 0 или и то и другое вместе. Т.о. эта команда служит для обнаружения первого ненулевого результата.
3. **LOOPNZ, LOOPNE** - выход из цикла по "нулю".

6.КОМАНДЫ ОБРАБОТКИ СТРОК (ЦЕПОЧЕК БАЙТОВ)

1. **LODSB**; команда lodsb загружает байт адресованный регистром SI из сегмента данных, и увеличивает SI на 1, если перед этим была введена команда CLD (очистить флаг направления DF) и уменьшает SI на 1, если была использована команда STD (установить флаг направления).
2. **MOVSB**; эта команда перемещает один байт из ячейки памяти с адресом в регистре SI в ячейку памяти с адресом в регистре DI и увеличивает (SI) и (DI) на 1. Значение SI может находиться, как в сегменте данных DS, так и в дополнительном сегменте ES. Значение DI может находиться только в дополнительном сегменте ES.
3. **REP** ;префикс повторения команды
4. **CMPSB**; осуществляет сравнение байта строки источника с адресом (SI) и байта строки приемника с адресом (DI)

7. КОМАНДЫ УПРАВЛЕНИЯ МИКРОПРОЦЕССОРОМ

1. **CLC**; сбросить флаг переноса (CF) = 0.
2. **STC**; установить флаг переноса (CF) = 1.
3. **CMC**; инвертировать флаг пнреноса.
4. **CLD**; очистить флаг направления (DF) = 0, в этом случае операции над строками (цепочками байтов) будут производиться от младшего адреса к старшему.
5. **STD**; установить флаг направления (DF) = 1,обработка цепочек байтов производится от старшего адреса к младшему.
6. **STI**; установить флаг прерываний (IF) = 1, разрешить прерывания от внешних устройств.
7. **CLI**; очистить флаг прерываний.
8. **NOP**; холостая операция.

8. КОМАНДЫ ПРЕРЫВАНИЙ

1. **INT** INUM; эта команда вызывает программное прерывание, то есть переход к ячейке памяти с адресом хранящимся в четырех байтах, начиная с адреса INUM * 4, где INUM = (0...255). Это 4-х байтовое число является указателем подпрограммы обработчика данного прерывания, и иначе называется вектором

3). ШАБЛОНЫ ДЛЯ ПРОГРАММ

Текст: C:\MASM.611\BIN\IGN2.ASM
Ст. 0 131 байт 100%

```
code segment
main:
mov ax, 12 ;(pribavity)
mov bx,- 4
idiv bx
mov ax, 4c00h
int 21h
code ends
end main
end
```