Инструктивно-справочные материалы по практикуму на MASM 6.14

Во втором семестре первого курса освоение языка Ассемблера может выполняться в среде виртуальной машины **win.prac** , установленной в компьютерных классах факультета. При загрузке виртуальной машины **win.prac** компилятор ассемблера с актуальным набором рекомендованных библиотек автоматически скачивается по адресу http://arch32.cs.mcu.ru/semestr2 и устанавливается в каталоге **C:\MASM 6.14** . На рабочем столе распрложен ярлык (shortcut), позволяющий быстро открыть этот каталог в Проводнике [1] .

Сеанс работы (вариант 1).

1. Войти в подкаталог **_Examples** каталога **C:\MASM 6.14** .

2. Создать рабочую папку, напрмер, **mywork1** [2] как копию каталога **_Шаблон** (**a.asm** и **makeit.bat**).

3. В файл a.asm [3] вставить текст своей программы. Отредактировать комментарий и заголовок консольного окна.

4. Создать резервную копию файла **a.asm** [4] (после каждого редактирования).

5. Запустить командный файл makeit.bat [5] , уже настроенный на обработку файла **a.asm** .

6. Если при компиляции были зафиксированы ошибки (с указанием номеров строк),
рекомендуется переключиться в окно редактора и внести необходимые исправления (п.3) .
Консольное окно следует закрыть перед тем, как компилировать программу (п.5) вновь.

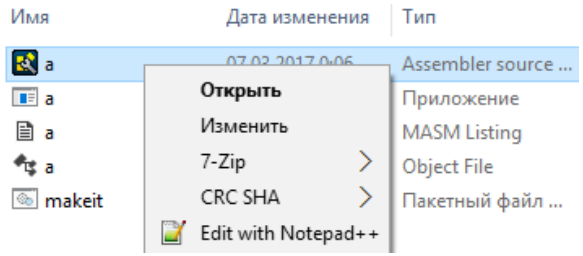
7. Протестировать программу и не забыть сохранить окончательный вариант [4] **.**

---

[1] В каталоге **C:\MASM 6.14**, в часности, размещен подкаталог **_Examples,** в котором находится подкаталог **_Шаблон**. В нем находятся файлы **a.asm** и **makeit.bat** .

[2] цель – внести мнемонику в имена отдельных каталогов для каждой отлаживаемой программы за время сеанса работы в каталоге **C:\MASM 6.14_Examples .**

[3] Для того, чтобы внести текст программы во все необходимые секции (.code, .data ...) можно использовать любой текстовый редактор. Например, нажать правой клавишей мыши на файле **a.asm** и выбрать в контекстном меню пункт Edit with Notepad++ . Важно не забывать **сохранять** изменения .



[4] Сделанную работу можно сохранить на диске **H:** (который ассоциируется в **win.prac** с личным сетевым каталогом студента), на Flash-диске (подключив его, обратившись к меню виртуальной машины комбинацией клавиш Правый Ctrl+Home) или воспользоваться интернетом для того, чтобы послать её по почте или воспользоваться другим сервисом .

Удобно открыть отдельное окно проводника и после каждого изменения обновлять резервную копию (только файл **a.asm** или целиком рабочую папку (например, **mywork1**)) на случай «зависания» компьютера.

[5] Открывается *« консольное »* окно, в котором отображается результат компиляции. Если ошибок компиляции не было, запускается линкер. Если и редактирование связей не вызвало ошибок начинается выполнение программы.

По окончании работы программы (в текущей строке) отображается сообщение Press any key to continue . . . После нажатия любой клавиши консольное окно закрывается.

Такая простая инструкция позволяет быстро начать писать и отлаживать свои программы. Также должны быть тщательно изучены все примеры, использованные в пособии В.Г.Баулы http://arch32.cs.mcu.ru и разбираемые на лекциях . Кроме того ряд интересных примеров находятся в подкаталогах **_Examples** и **prg .**

Команды языка Ассемблера (Макроассемблер 6.14 32-битный)

Ниже приведен список наиболее употребительных команд , который может выполнить роль справочной таблицы.

Использованы следующие обозначения.

Предложения**-команды** – это символьная форма записи машинных команд. Общий синтаксис этого типа предложений таков:

[< метка > :] < мнемокод > [< операнды >] [; < комментарий >]

Скобки [ ] ~ необязательному полю. Кроме того, *пробел отделяет* мнемокод команды от его первого операнда; *запятая разделяет* первый и второй операнды. Пусть символ | (вертикальная черта) будет иметь смысл – или одно, или другое; а символы **B**(byte) | **W**(word) | **D**(dword) указывают на возможные типы операндов (либо оба размером байт, либо оба размером слово или оба размером в двойное слово).

Обозначим **r** – любой регистр, кроме сегментного (а именно, AH, AL, BH, BL, CH, CL, DH, DL – **r8** ;
AX, BX, CX, DX, SI, DI, BP, SP – **r16** ;
EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP – **r32**);

SR – сегментный регистр (один из SS, DS, ES) и **CS** – кодовый сегментный регистр;

i (immediate) – непосредственный операнд, задаваемый в самой команде в виде константы (константного выражения) длиной 1, 2, 4 или 8 байт соответственно **i8, i16, i32** ;

m (memory) – адресное выражение **m8, m16, m32**.

Предложения **резервирования памяти** или директивы резервирования памяти (указания компилятору выделить память)

Синтаксис этого типа предложений таков:

[< метка >] < мнемокод > [< операнды >] [; < комментарий >]

Размер (байт)	Название формата	мнемокод	Примеры (адресация, директива **equ**)		
1	короткое целое	db		n **equ** 20	n equ 10
2	длинное	dw	A dw 20 **dup** (?)	A dw n dup (?)	m equ 2*n
4	сверхдлинное	dd	(Y +0|+1|+2|+3 |+4)		; X – матрица 10x20
8	расширенное	dq	Y db 2, -2, ?, '*', 0FFh		X db n dup (m dup(?)) или
			Записи "*" ~ код этого символа		X db n*m dup(?)
Константы byte=1, word=2, dword=4, qword=8			**dup**licate-копировать		

См. Пособие гл. 4.5 и гл. 6.4

1 КОМАНДЫ **ПЕРЕСЫЛОК** (без работы с сегментными ренистрами)

Синтаксис**:** МнемоКод op1, op2 **Действие:** op1 **:=** op2 (основное действие, как правило такое)

MOV	r**,** i|r|m	B|W|D	Пересылка значения операнда op2 в op1
MOV	m**,** i|r	B|W|D	Пересылка значения операнда op2 в op1
XCHG	r**,** r|m	B|W|D	Обмен содержимым между операндами op1 и op2
XCHG	m**,** r	B|W|D	Обмен содержимым между операндами op1 и op2

Примечание. В командах MOV, XCHG операнды должны быть одинакового**!** размера и *не* быть вида память-память (*не* m, m).

Синтаксис**:** МнемоКод

CBW Преобразование значения *знакового байта*, находящегося в AL, в *слово*-результат в регистре AX
CWD Преобразование значения *знакового слова*, находящегося в AX, в *двойное слово* – <DX, AX>
CWDE Преобразование значения *знакового слова*, находящегося в AX, в *двойное слово* – EAX
CDQ для *знакового* расширения регистра EAX до пары регистров <EDX, EAX>

Синтаксис**:** МнемоКод op1, op2 **Действие:** op1 **:=** op2 (основное действие)

(команды, представляющие исключение из правил по синтаксису – имеют операнды разных размеров)

MOVZX	r16**,** r8|m8	Пересылка с расширением *беззнакового* значения операнда op2 в op1
MOVZX	r32**,** r8|m8| r16|m16	Пересылка с расширением *беззнакового* значения операнда op2 в op1
MOVSX	r16**,** r8|m8	Пересылка с расширением *знакового* значения операнда op2 в op1
MOVSX	r32**,** r8|m8| r16|m16	Пересылка с расширением *знакового* значения операнда op2 в op1
LEA	r32**,** m8 |m16|m32	Загрузка в регистр вычисленного исполнительного адреса

Синтаксис**:** МнемоКод op

PUSH	r32|m32|i32	D	Запись содержимого операнда op двойного слова в стек ESP:=(ESP – 4) mod 2^{32} ; <ESP>:= op (push ESP в стек кладется до своего изменения)
POP	r32|m32	D	Пересылка двойного слова из стека в регистр или по исполнительному адресу op := <ESP> ; ESP:=(ESP + 4) mod 2^{32}

Команды пересылок флаги не меняют.

2 **АРИФМЕТИЧЕСКИЕ** команды.

Синтаксис**:** МнемоКод op1, op2 **Действие:** op1 **:=** op2 (основное действие)

ADD r**,** i|r|m B|W|D Сложение op1:= op1 + op2
ADD m**,** i|r B|W|D Сложение op1:= op1 + op2
SUB r**,** i|r|m B|W|D Вычитание op1:= op1 – op2
SUB m**,** i|r B|W|D Вычитание op1:= op1 – op2
ADC r**,** i|r|m B|W|D Сложение *с учетом предыдущего CF* op1:= op1 + op2 + CF
ADC m**,** i|r B|W|D Сложение *с учетом предыдущего CF* op1:= op1 + op2 + CF
SBB r**,** i|r|m B|W|D Вычитание *с учетом предыдущего CF* op1:= op1 – op2 – CF
SBB m**,** i|r B|W|D Вычитание *с учетом предыдущего CF* op1:= op1 – op2 – CF
CMP r**,** i|r|m B|W|D Сравнение операндов, аналог SUB без сохранения результата в op1
CMP m**,** i|r B|W|D Сравнение операндов, аналог SUB без сохранения результата в op1

В командах выше операнды должны быть одинакового! размера и *не* быть вида память-память (*не* m, m), устанавливают (меняют) флаги CF, OF, SF, ZF.

Синтаксис**:** МнемоКод op1
INC r|m B|W|D Увеличение значения операнда op на 1
DEC r|m B|W|D Уменьшение значения операнда op на 1
Команды INC и DEC не меняют CF флаг.
NEG r|m B|W|D Изменение значения операнда op на противоположное

MUL r|m B|W|D Умножение *беззнаковое* значения регистра AL (или AX | EAX) на значение op в команде соответствующего размера; результат в регистре AX (или в регистрах <DX, AX> | <EDX, EAX>)
IMUL r|m B|W|D Умножение *знаковое* значения регистра AL (или AX | EAX) на значение op в команде; результат в регистре AX (или в регистрах <DX, AX> | <EDX, EAX>)
DIV r|m B|W|D Деление *беззнаковое* значения в регистре AX (или в регистрах <DX, AX> | <EDX, EAX>) на значение op, указанного в команде; результаты в регистрах AH:= *mod* (или в DX:= *mod* | EDX= *mod*) и AL:= *div* (или AX:= *div* | EAX= *div*)
IDIV r|m B|W|D Деление *знаковое* значения в регистре AX (или в регистрах <DX, AX> | <EDX, EAX>) на значение op, указанного в команде; результаты в регистрах AH:= *mod* (или в DX:= *mod* | EDX= *mod*) и AL:= *div* (или AX:= *div* | EAX= *div*)

OP2 не может быть непосредственным операндом: 2, 1000, n заданное по equ. При умножении CF и OF устанавливаются по правилу CF=OF=1 , если результат превосходит размер операндов, иначе CF=OF=0. Флаги SF и ZF портятся.

4 КОМАНДЫ **УПРАВЛЕНИЯ** (основные, наиболее употребительные)

Синтаксис: МнемоКод op (не меняют флаги!).

Безусловные– заменяется значение регистра EIP на новое.

JMP	<метка>		Переход прямой
JMP	r32 | m32	D	Переход косвенный (длинный) по адресу, содержащемуся в операнде
CALL	<метка>		Переход по метке с запоминанием адреса следующей за ней команды в стеке..
CALL	<имя близкой процедуры >		Вызов процедуры (переход с запоминанием точки возврата**:** запись в стек EIP – адреса следующей команды и передача управления на метку процедуры)
	< i32 | r32 | m32 **>**		(см.пособие гл.6).
RET			Возврат из процедуры (считывание из вершины стека адреса и переход по нему)
RET	i16		Возврат из подпрограммы (после извлечения из стека адреса возврата, увеличивается значение регистра ESP на число байтов, равное беззнаковому значению i16, затем передается управление по адресу возврата).

Условные прямые (могут быть как длинными, так и короткими автоматически)

Jxx	<метка>		Условный переход см. далее (на следующем листе уточнения)

Условные короткие(не длинные) прямые(не косвенные) относительные(не абсолютные)

LOOP	<метка>	i8	ECX := ECX–1; короткий переход на метку, если ECX <> 0
LOOPE/Z	<метка>	i8	ECX := ECX–1; переход, если (ECX<>0) and (ZF=1)
LOOPNE/NZ	<метка>	i8	ECX := ECX–1; переход, если (ECX<>0) and (ZF=0)
JECXZ	<метка>	i8	Переход по метке, если ECX=0

```
        mov ecx, <нач.знач>
        JECXZ  L
@@:     тело цикла
        dec ecx
        jnz  @B
L:
```

Команды управления флаги не меняют. Регистр ECX – неявный операнд.

EIP (Extanded **I**nstruction **P**ointer ~ расширенный указатель команд **)** – регистр центрального процессора, при выполнении текущкй команды указывает на начало следующей команды. При относительном переходе значение регистра EIP *изменяется* на знаковую константу: i8 – короткий, i32 – длинный переход. При абсолютном переходе *заменяется* на новое значение i32. Прямые переходы отличаются от косвенных тем, что адрес перехода не явно указан в команде, а находится в “слове” памяти или в регистре, указанном в команде. Символ /(слэш) используется для синонимов в дальнейшем.

Если используется анонимная метка @@**,** то переход jxx @F – вперед на ближайшую, а jxx @B – назад на ближайшую

4а **УСЛОВИЯ** перехода в командах вида **Jxx** <метка>

Мнемокод		Переход, если выполнено условие	Значения флагов
Переходы после команды сравнения для знаковых чисел.			
JL/JNGE	<	меньше / не больше и не равно	SF<>OF
JGE/JNL	≥	больше или равно / не меньше	SF = OF
JLE/JNG	≤	меньше или равно / не больше	(SF<>OF) or (ZF=1)
JG/JNLE	>	больше / не меньше и не равно	(SF=OF) and (ZF=0)
Переходы после команды сравнения для беззнаковых чисел.			
JB/JNAE	<	меньше / не больше и не равно	CF=1
JAE/JNB	≥	больше или равно / не меньше	CF=0
JBE/JNA	≤	меньше или равно / не больше	(CF=1) or (ZF=1)
JA/JNBE	>	больше / не меньше и не равно	(CF=0) and (ZF=0)
Переходы после команд, устанавливающих тот или иной флаг			Значение флага
JE/JZ	=	результат равен нулю	ZF=1
JNE/JNZ	≠	результат не равен нулю	ZF=0
JC|JNC		есть|нет перенос	CF=1 | 0 соответственно
JO|JNO		есть|нет переполнение	OF=1 | 0 соответственно
JS|JNS		отрицатательный|положительный результат	SF=1 | 0 соответственно

Для всех этих команд реализован один формат –прямой относительный переход.
Переход прямой , т.к. в качестве операнда указывается метка команды, на которую надо передать управление.
Переход относительный потому, что в машинной команде указывается не сам адрес, а разность между командой перехода и адресом перехода (разность вместо метки подставит сам ассемблер).
JB/JNAE/ JC – одна и та же машинная команда, аналогично **JAE/JNB/ JNC** – одна и та же машинная команда.

6 **ЛОГИЧЕСКИЕ** команды и команды **СДВИГОВ** (выполняются **поразрядно**).

Синтаксис**:** МнемоКод op1, op2 Действие: op1 **:=** op2

AND	r**,** i|r|m B|W|D	Логическое умножение (конъюнкция) поразрядное op1**:=** op1 and op2
AND	m**,** i|r B|W|D	
TEST	r**,** i|r|m B|W|D	Логическое умножение *но! без сохранения результата* в op1
TEST	m**,** i|r B|W|D	
OR	r**,** i|r|m B|W|D	Логическое сложение (дизъюнкция) поразрядное op1**:=** op1 or op2
OR	m**,** i|r B|W|D	
XOR	r**,** i|r|m B|W|D	*Исключающее или* (сложение по модулю 2)op1**:=** op1 xor op2
XOR	m**,** i|r B|W|D	

Пояснение к команде xor: i-й бит результата равен 0(нулю), если i-е биты операндов *совпдают,* иначе
i-й бит результата равен 1(единице), если i-е биты операндов *различны*.

Синтаксис**:** МнемоКод op

NOT	i|r|m B|W|D	Поразрядное инвертирование каждого бита операнда op

Напоминание. Логические команды меняют все флаги; но после них обычно используется флаг нуля ZF;
ZF= 1, если результат нулевой во всех разрядах и ZF= 0, если в результате есть хоть одна двоичная единица.

Таблица

Продолжение 6 Команды сдвигов.

Синтаксис**:** МнемоКод op1, i8 (или CL)
второй операнд рассматривается как целое без знака и определяет на сколько разрядов сдвигать.
Действие: сдвиги осуществляются специальным способом на “сдвиговых регистрах” с учетом перевернутого хранения в памяти, результат сдвига записывается на место первого операнда.
Команды сдвига устанавливают все флаги; но как правило, используются два: CF и ZF.

В командах далее в CF заносится значение "уходящего" бита, а с другого конца добавляется 0.

SHR/**SHL**	r|m, i8	*Логический* сдвиг op1 размером B|W|D вправо/влево на k=i8 разрядов
SHR/**SHL**	r|m, CL	*Логический* сдвиг op1 размером B|W|D вправо/влево на содержимое байтового регистра CL
SAR	r|m, i8	*Арифметический* сдвиг op1 размером B|W|D вправо на i8 разрядов и знаковый разряд восстанавливается
SAR	r|m, CL	*Арифметический* сдвиг op1 размером B|W|D вправо на CL разрядов и знаковый разряд восстанавливается
SAL		*Арифметический* сдвиг влево выполняется одинаково с командой логического сдвига SHL

В командах далее попрежнему в CF заносится значение "уходящего" бита и в освобождающийся бит добавляется *он же*! , не 0

ROR/**ROL**	r|m, i8	*Циклический* сдвиг op1 размером B|W|D вправо/влево на i8 разрядов
ROR/**ROL**	r|m, CL	*Циклический* сдвиг op1 размером B|W|D вправо/влево на CL разрядов

В командах далее в CF заносится значение "уходящего" бита, но *прежнее* значение CF заносится в освобождающийся бит.

RCR/**RCL**	r|m, i8	*Циклический* сдвиг через перенос op1 вправо/влево на i8 разрядов (op2=0)
RCR/**RCL**	r|m, CL	*Циклический* сдвиг через перенос op1 вправо/влево на CL разрядов

Применение.

Быстрое умножение, деление и остаток от деления. Особенности деления отрицательных чисел. Правильность результата.