Введение в программирование на языке ассемблера MASM для архитектуры IA-32

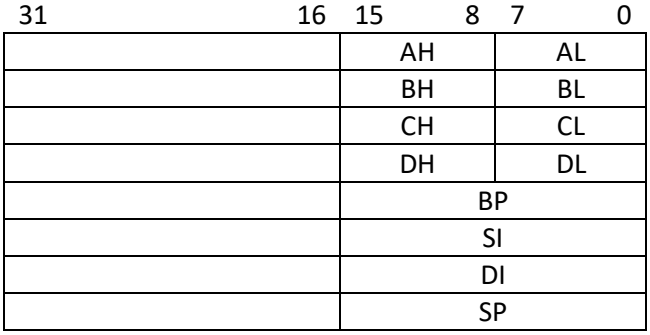
IA-32 (Intel Architecture, 32-bit) — это 32-разрядная версия архитектуры набора команд x86. Также встречается название i386, поскольку впервые этот вариант архитектуры x86 был реализован в микропроцессоре Intel 80386. Последующие расширения этой системы команд, реализованные в микроархитектурах 80486, P5 и P6, называют, соответственно, i486, i586 и i686. Данные расширения дополнили базовую архитектуру возможностями вычислений с плавающей точкой и SIMD-вычислений (использующих параллелизм на уровне данных).

Организация памяти

В архитектуре IA-32 применяется байтовая адресация памяти с помощью 32-разрядных адресов. Порядок байтов — обратный (little-endian). Инструкции работают с операндами длиной 8 (байт), 16 (слово) или 32 (двойное слово) бит. Используются также 64-разрядные операторы (четверное слово) для чисел с плавающей точкой двойной точности и упакованных целых чисел. Архитектура IA-32 имеет различные режимы доступа к памяти. Далее везде подразумевается плоская (flat) модель памяти, при которой программе доступно одно сплошное адресное пространство с адресами от 0 до $2^{32} - 1$.

Регистры

В процессорах IA-32 имеется восемь 32-разрядных регистров общего назначения. Эти регистры допускают обратную совместимость со старыми 8- и 16-разрядными процессорами Intel. На рисунке 1 показана взаимосвязь новых и старых регистров общего назначения, а также некоторые общеупотребительные соглашения по их использованию.



16-bit	32-bit	Назначение
AX	EAX	Аккумулятор
BX	EBX	Указатель базы
CX	ECX	Счетчик
DX	EDX	Целое умножение/деление
	EBP	Указатель кадра стека
	ESI	Индекс источника
	EDI	Индекс приемника
	ESP	Указатель стека

Рисунок 1. Регистры общего назначения

Счетчик команд в процессорах x86 называется EIP (extended instruction pointer). Он содержит адрес следующей инструкции, подлежащей исполнению, и может быть изменен командами ветвлений, безусловных переходов и вызова функций.

Большинство архитектур CISC используют флаги состояния (коды условий) для принятия решений о ветвлениях и отслеживания переносов и арифметических переполнений. В архитектуре IA-32 используется 32-разрядный регистр EFLAGS, в котором хранятся флаги состояния. Назначение некоторых битов из регистра EFLAGS приведено в таблице 1. Остальные биты используются операционной системой.

Таблица 1. Некоторые биты регистра EFLAGS

Название	**Назначение**
CF (Carry Flag)	Бит переноса, сгенерированный последней арифметической операцией. Обозначает переполнение в арифметике без знака.
ZF (Zero Flag)	Результат последней операции был равен нулю.
SF (Sign Flag)	Результат последней операции был отрицательным (старший бит результата равен 1).
OF (Overflow Flag)	Переполнение при вычислениях со знаковыми целыми.

Операнды

Команды IA-32 содержат два операнда: операнд-источник и операнд-источник/приёмник. Следовательно, команда всегда записывает результат на место одного из операндов. Операндами могут быть непосредственные значения, содержимое регистров и значения из памяти. В таблице 2 перечислены допустимые комбинации расположения операндов в командах IA-32 на примере команды сложения (add). Как видно, возможны любые комбинации, кроме «память–память».

Таблица 2. Расположение операндов

Источник/Приёмник	**Источник**	**Пример**	**Выполняемая операция**
Регистр	Регистр	add EAX, EBX	EAX <– EAX + EBX
Регистр	Непосредственный операнд	add EAX, 42	EAX <– EAX + 42
Регистр	Память	add EAX, [20]	EAX <– EAX + Mem[20]
Память	Регистр	add [20], EAX	Mem[20] <- Mem[20] + EAX
Память	Непосредственный операнд	add [20], 42	Mem[20] <- Mem[20] + 42

Режимы адресации памяти

Эффективный адрес операнда, находящегося в памяти, в самом общем виде вычисляется как

$$\text{Effective Address} = \text{BaseReg} + \text{IndexReg} * \text{ScaleFactor} + \text{Disp}.$$

Здесь базовый регистр (BaseReg) может быть любым регистром общего назначения; индексный регистр (IndexReg) может быть любым регистром общего назначения кроме ESP (т.к. он служит указателем процессорного стека); величина смещения (Disp) указывается непосредственно в инструкции; масштабный множитель (ScaleFactor) может принимать значения 1, 2, 4 или 8. Длина эффективного адреса — всегда 32 бит. Для вычисления эффективного адреса не обязательно задавать все перечисленные параметры. В таблице 3 показаны возможные формы адресации операнда в памяти на примере команды пересылки

(mov). Во всех примерах из таблицы значение двойного слова, расположенного в ячейке с соответствующим эффективным адресом, копируется в регистр EAX.

Таблица 3. Режимы адресации памяти

Форма адресации	**Пример**
Disp	mov eax,[MyVal]
BaseReg	mov eax,[ebx]
BaseReg + Disp	mov eax,[ebx+12]
Disp + IndexReg * SF	mov eax,[MyArray+esi*4]
BaseReg + IndexReg	mov eax,[ebx+esi]
BaseReg + IndexReg + Disp	mov eax,[ebx+esi+12]
BaseReg + IndexReg * SF	mov eax,[ebx+esi*4]
BaseReg + IndexReg * SF + Disp	mov eax,[ebx+esi*4+20]

Форма адресации, использующая только смещение обычно используется для доступа к глобальной или статической переменной. Базовый регистр аналогичен указателю в языке C/C++. С помощью базового регистра и смещения можно осуществлять доступ к полям структуры. Формы адресации, использующие индексный регистр, полезны для доступа к элементам массива. Масштабный множитель удобен при доступе к элементам массива стандартных типов (размером 1, 2, 4 или 8 байт). Комбинации базового и индексного регистров полезны при доступе к элементам двумерного массива.

Команды

Архитектура IA-32 имеет большую систему команд. В таблице 4 показаны некоторые из команд общего назначения. Представленные команды могут иметь разное число операндов, например

op

op D

op D, S

Здесь op — символическое обозначение операции (мнемоника). Операнд-приёмник обозначен как D (регистр или ячейка памяти), а операнд-источник обозначен как S (регистр, непосредственный операнд (imm) или ячейка памяти).

Таблица 4. Некоторые команды IA-32

Мнемоника	**Назначение**	**Функция**
ADD/SUB	Сложение/вычитание	D = D + S / D = D − S
ADDC	Сложение с переносом	D = D + S + CF
INC/DEC	Инкремент/декремент	D = D + 1 / D = D − 1
CMP	Сравнение	Установить флаги по результатам D – S
NEG	Инверсия	D = −D

AND/OR/XOR	Логическое «И/ИЛИ/ИСКЛЮЧАЮЩЕЕ ИЛИ»	D = D op S
NOT	Логическое НЕ	D = !D
IMUL/MUL	Знаковое/беззнаковое умножение	EDX:EAX = EAX * D
IDIV/DIV	Знаковое/беззнаковое деление	EDX:EAX/D; EAX = частное; EDX = остаток
SAR/SHR	Арифметический/логический сдвиг вправо	D = D >>> S / D = D >> S
SAL/SHL	Арифметический/логический сдвиг влево	D = D <<< S / D = D << S
ROR/ROL	Циклический сдвиг вправо/влево	Циклически сдвинуть D на S разрядов
RCR/RCL	Циклический сдвиг вправо/влево через бит переноса	Циклически сдвинуть CF и D на S разрядов
BT	Проверка бита	CF = D[S] (бит номер S из D)
BTR/BTS	Проверить бит и сбросить/установить его	CF = D[S]; D[S] = 0 / 1
TEST	Установить флаги по результатам проверки бит	Аналогично AND D, S, но без переписывания D
MOV	Скопировать операнд	D = S
PUSH	Поместить на стек	ESP = ESP −4; Mem[ESP] = S
POP	Прочитать из стека	D = Mem[ESP]; ESP = ESP + 4
CLC, STC	Сбросить/установить флаг переноса	CF = 0 / 1
JMP	Безусловный переход	Переход по относительному адресу: EIP = EIP + S. Переход по абсолютному адресу: EIP = S
Jcc	Ветвление (условный переход)	Если установлен флаг, то EIP = EIP + S
LOOP	Проверка условия цикла	ECX = ECX – 1. Если ECX ≠ 0, то EIP = EIP + imm
CALL	Вызов функции	ESP = ESP – 4; Mem[ESP] = EIP; EIP = S
RET	Возврат из функции	EIP = Mem[ESP]; ESP = ESP + 4

Можно заметить, что некоторые команды всегда производят действия только с определенными регистрами. Например, умножение двух 32-битных чисел всегда использует в качестве одного из источников EAX и всегда записывает 64-битный результат в EDX и EAX. Команда LOOP всегда хранит счетчик итераций цикла в ECX, а команды PUSH, POP, CALL и RET используют указатель вершины стека ESP.

В таблице встречается команда Jcc (jump). На самом деле такой команды нет. Так в таблице обозначено целое семейство команд условного перехода, которые проверяют значения флагов кодов условий (condition codes) из регистра EFLAGS и, если выполнено соответствующее условие, осуществляют ветвление. Например, команда JZ осуществляет переход в том случае, когда флаг нуля (ZF) равен 1, а команда JNZ – когда ZF равен 0. Команды перехода обычно

следуют за командами, которые устанавливают флаги, такими как команда сравнения (CMP). В таблице 5 перечислены коды условий, соответствующие мнемонические суффиксы (cc) и проверяемые при этом флаги состояний. Существуют также похожие команды условного перемещения CMOVcc (conditional move) и SETcc (установить байт операнда в 1 при выполнении условия или в 0 в противном случае). Например, пара команд

```
CMP D, S
CMOVLE D1, S1
```

осуществляет присваивание `D1 = S1`, только если `D <= S`. Последовательность

```
CMP D, S
SETNE D1
```

имеет следующий смысл: `(D != S ? D1 = 1 : D1 = 0).` В командах SETcc используется однобайтный операнд-приемник.

Таблица 5. Коды условий, мнемонические суффиксы (cc) и проверяемые условия

Код условия	**Суффикс**	**Проверяемое условие**
Above	A	CF == 0 && ZF == 0
Neither below or equal	NBE	
Above or equal	AE	CF == 0
Not below	NB	
Below	B	CF == 1
Neither above nor equal	NAE	
Below or equal	BE	CF == 1 || ZF == 1
Not above	NA	
Equal	E	ZF == 1
Zero	Z	
Not equal	NE	ZF == 0
Not zero	NZ	
Greater	G	ZF == 0 && SF == OF
Neither less nor equal	NLE	
Greater or equal	GE	SF == OF
Not less	NL	
Less	L	SF != OF
Neither greater nor equal	NGE	
Less or equal	LE	ZF == 1 || SF != OF
Not greater	NG	
Sign	S	SF == 1
Not sign	NS	SF == 0
Carry	C	CF == 1
Not carry	NC	CF == 0
Overflow	O	OF == 1
Not overflow	NO	OF == 0

Parity	P	PF == 1
Parity even	PE	
Not parity	NP	PF == 0
Parity odd	PO	

Система команд IA-32 также включает команды обработки чисел с плавающей точкой и специализированные инструкции для мультимедийных приложений (MMX) и векторной обработки данных (SSE).

Язык ассемблера

В языке ассемблера имеется ряд операторов, которые позволяют управлять процессом ассемблирования. Эти операторы называются директивами. Они действуют только в процессе ассемблирования программы и, в отличие от команд, не генерируют машинных кодов.

Например, директивы ассемблера позволяют задавать размер операнда. Если в качестве операнда используется регистр EAX, то подразумеваемый размер операнда 32 бита. Регистр AL подразумевает размер операнда 8 бит. Однако, в некоторых случаях ассемблеру требуется дополнительная информация о размере операнда. Например, в команде

```
MOV DWORD PTR [EAX + 16], 100
```

используются ключевые слова DWORD PTR, чтобы указать размер операнда. В данном случае значение в квадратных скобках должно интерпретироваться как адрес 32-разрядного операнда. Таким образом данная команда аналогична по смыслу следующему выражению на языке C/C++

```
*(int*)(EAX + 16) = 100
```

Аналогично, ключевые слова WORD PTR и BYTE PTR указывают, что следующие за ними имена следует интерпретировать как 16 и 8-битные операнды соответственно.

Если LOCATION — метка обозначающая какой-то адрес в памяти, то команда

```
MOV EAX, LOCATION
```

перемещает двойное слово, расположенное по этому адресу, в регистр EAX, т.е. данная команда эквивалентна следующей

```
MOV EAX, DWORD PTR LOCATION
```

Если же требуется трактовать LOCATION как непосредственный операнд, то используется ключевое слово OFFSET:

```
MOV EAX, OFFSET LOCATION
```

Здесь адрес LOCATION записывается в регистр EAX.

Структура программы на языке ассемблера MASM

При написании программ на языке ассемблера MASM (Microsoft макроассемблер), можно использовать следующий шаблон:

```
.386
.model flat,stdcall
.stack 4096

ExitProcess PROTO, dwExitCode:DWORD

.data
      ; здесь объявляются переменные

.code
main PROC
      ; сюда помещается код программы

      INVOKE ExitProcess,0
main ENDP
END main
```

Директива `.386` разрешает ассемблирование непривилегированных команд процессора 80386. Таким образом, программа сможет работать с 32-разядными регистрами и адресами. Директива `.model flat,stdcall` задает плоскую модель памяти (flat) и определяет стандартное соглашение о вызове процедур (stdcall). Это соглашение используется в 32-разрядной версии операционной системы Windows. Строка `.stack 4096` задает размер программного стека в 4 КБ (по умолчанию используется 1024 байта). Директивы `.data` и `.code` отмечают сегменты программы, в которых располагаются, соответственно, данные и код. Символом точки с запятой «;» начинается комментарий, действующий до конца строки.

Директива `PROTO` объявляет прототип функции `ExitProcess`, которая является системной функцией Windows. После запятой указывается список входных параметров объявляемой функции. В данном случае имеется один параметр с формальным именем `dwExitCode`, типа `DWORD` (двойное слово). Этот параметр играет роль возвращаемого значения данной ассемблерной программы. Если это значение равно `0`, это свидетельствует об успешном завершении программы.

Первая строчка после директивы `.code` обычно является точкой входа в программу (т.е. это первая команда, которая будет выполняться при запуске программы). Как правило это процедура с именем main. Код процедуры располагается между директивами `main PROC` и main `ENDP`. Последнее, что делает данная процедура, это вызов служебной функции `ExitProcess` с параметром `0`. Этот вызов осуществляется директивой `INVOKE`, которая, в отличие от стандартной команды `CALL`, передает аргументы на стек в соответствии со стандартными соглашениями о передаче параметров. Наконец, директива `END` отмечает конец программы и сообщает ассемблеру точку входа в программу (в данном случае main).

Определение данных

Непосредственно после директивы `.data` определяются данные для программы. Например, команда

```
count DWORD 12345
```

определяет двойное слово с именем `count`, которое должно инициализироваться значением 12345. Имя переменной является меткой, значение которой есть адрес данной переменной (строго говоря, это относительный адрес, т.е. смещение от адреса начала сегмента данных). Тим переменной сообщает ассемблеру размер памяти, необходимый для размещения этой переменной. В таблице 6 приведены некоторые стандартные для MASM типы данных.

Таблица 6. Некоторые встроенные типы данных MASM

Тип	**Описание**
BYTE	8-битное целое без знака
SBYTE	8-битное целое со знаком (S = signed)
WORD	16-битное целое без знака
SWORD	16-битное целое со знаком
DWORD	32-битное целое без знака (D = double)
SDWORD	32-битное целое со знаком (SD = signed double)
QWORD	64- битное целое (Q = quad)

Переменную можно оставить неинициализированной, используя символ `?` вместо инициализирующего значения, например,

```
myVar BYTE ?
```

Инициализирующие значения можно задавать в разных системах счисления. Так `00110010b`, `32h`, и `50d` представляют собой одно и то же десятичное число 50, записанное, соответственно, в двоичном, шестнадцатеричном и десятичном формате.

Тип DWORD можно использовать для объявления переменной, которая содержит 32-битный адрес другой переменной, например,

```
pVal DWORD val3
```

Здесь `pVal` содержит адрес переменной `val3`.

В одной команде объявления данных можно использовать несколько инициализирующих значений, разделенных запятыми. При этом метка будет представлять собой адрес первого инициализатора. Например, команда

```
list BYTE 10,20,30,40
```

определяет массив `list` из четырех однобайтных значений. Этот массив можно продолжить

```
list BYTE 10,20,30,40
     BYTE 50,60,70,80
     BYTE 81,82,83,84
```

Строку символов, завершающуюся нулевым байтом, можно определить аналогичным образом:

```
greeting1 BYTE 'H','e','l','l','o',' ','W','o','r','l','d','!',0
```

Ту же строку можно объявить с помощью более удобного синтаксиса

```
greeting1 BYTE ‘Hello World!’,0
```

Также можно использовать двойные кавычки

```
greeting1 BYTE “Hello World!”,0
```

Объявление строки может занимать несколько строк в тексте программы:

```
greeting1 BYTE “Welcome to demo program ”
     BYTE “created by Somebody Hacker.”,0dh,0ah
     BYTE “If you wish to modify this program, please ”
     BYTE “send me a copy.”,0dh,0ah,0
```

Шестнадцатеричные коды `0dh` и `0ah` соответствуют символам возврата каретки (CR) и перевода строки (LF), соответственно. При выводе в стандартное устройство вывода, эти два символа дают эффект перевода курсора в начало следующей строки.

Выделить память для нескольких элементов данных можно с помощью оператора дупликации `DUP`:

```
BYTE 20 DUP(0)              ; 20 байт, все равны нулю
BYTE 20 DUP(?)              ; 20 байт, не инициализированы
BYTE 4 DUP(“STACK”)         ; 20 байт: “STACKSTACKSTACKSTACK”
```

Ниже приведены еще некоторые примеры определения данных:

```
value1 BYTE ‘A’             ; символьный литерал
value3 SBYTE –128           ; знаковый байт
word1 WORD 65535            ; максимальное значение слова без знака
word2 SWORD -32768          ; минимальное двухбайтное беззнаковое слово
word3 WORD ?                ; неинициализированное беззнаковое слово
val1 DWORD 12345678h        ; беззнаковое двойное слово
val2 SDWORD –2147483648          ; четырехбайтная переменная со знаком
val3 DWORD 20 DUP(?)             ; массив из 20 беззнаковых двойных слов
pVal DWORD val3                  ; указатель на начало массива val3
myList DWORD 1,2,3,4,5           ; массив
quad1 QWORD 1234567812345678h    ; четверное слово
```

Ниже показана программа, демонстрирующая определение и использование переменных.

```
; Программа, складывающая значения трех переменных,
; sum = val1 + val2 + val3

.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD

.data
val1 DWORD 20002000h
val2 DWORD 11111111h
val3 DWORD 22222222h
```

```
sum DWORD 0

.code
main PROC
mov eax, val1
add eax, val2
add eax, val3
mov sum, eax

INVOKE ExitProcess,0
main ENDP
END main
```

Символические константы

Язык ассемблера MASM позволяет определять символические имена для целых значений и выражений, а также для текстовых строк. Символические константы не резервируют место в памяти. Они используются только ассемблером при сканировании текста программы и не могут быть изменены во время выполнения программы.

Директива, обозначаемая символом `=` ассоциирует имя с целочисленным выражением. Например, если после команды

```
N = 100
```

в тексте программы встретится имя `N`, оно будет заменено значением `100` в процессе ассемблирования. Символ, объявленный с помощью директивы `=` может быть переопределен.

В следующем примере

```
list BYTE 10,20,30,40
ListSize = ($ - list)
```

имя `ListSize` означает длину массива байтов `list`. Для вычисления этой длины используется оператор `$`, который возвращает адрес текущего местоположения (в данном примере это адрес байта, следующего за последним элементом массива). При вычислении количества элементов в массиве, нужно учитывать размер элементов, например

```
list WORD 1000h,2000h,3000h,4000h
ListSize = ($ – list) / 2

list DWORD 10000000h,20000000h,30000000h,40000000h
ListSize = ($ – list) / 4
```

Директива, `EQU`, позволяет связать символическое имя не только с целочисленным значением, но и с некоторым произвольным текстом. В последнем случае, текст помещается в угловые скобки (<>). Например, следующий фрагмент программы

```
matrix1 EQU 10 * 10
matrix2 EQU <10 * 10>
```

```
.data
M1 WORD matrix1
M2 WORD matrix2
```

заменится ассемблером на

```
M1 WORD 100
M2 WORD 10 * 10
```

Символ, определенный с помощью `EQU` не может быть переопределен.

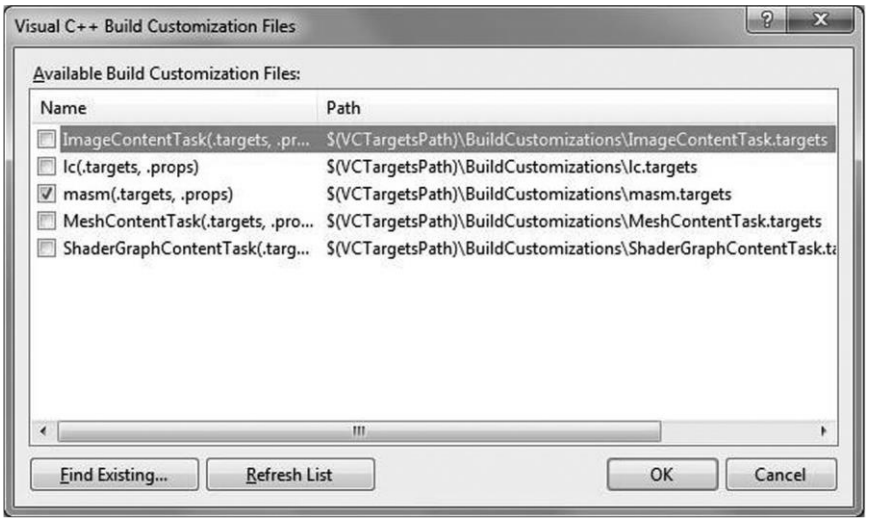
Создание проекта в Microsoft Visual Studio

Макроассемблер MASM поставляется вместе со средой программирования Visual Studio. Чтобы писать и отлаживать программы на языке ассемблера MASM в этой среде нужно создать стандартный проект для консольного приложения Win32 (так как это делается для языка C++). Если программа будет состоять только из ассемблерного кода, лучше выбирать пустой проект. Чтобы включить поддержку MASM нужно:

1. В окне Solution Explorer правой кнопкой мыши нажать на название созданного проекта и выбрать Build Dependencies | Build Customizations.

2. В открывшемся окне Visual C++ Build Customization Files отметить пункт masm(.targets, .props).

3. Нажать OK.



Затем нужно добавить новый файл в проект:

1. В окне Solution Explorer правой кнопкой мыши нажать на название проекта и выбрать Add | New Item.

2. В открывшемся окне Add New Item выбрать Installed | Visual C++ | Code.

3. Выбрать тип файла C++ File (.cpp).

4. В поле Name ввести имя файла явно прописав расширение .asm и выбрать нужную директорию в поле Location.

5. Нажать Add.

Чтобы проверить, что для файла на языке ассемблера установился соответствующий инструмент сборки, нужно правой кнопкой мыши нажать на asm-файл и выбрать Properties. В открывшемся окне выбрать Configuration Properties | General | Item Type. Убедиться, что выбран пункт Microsoft Macro Assembler.

В качестве примера в добавленный файл с расширением .asm можно скопировать следующую простую программу:

```
; Программа складывающая два 32-разрядных числа
.386
.model flat,stdcall
.stack 4096

ExitProcess proto, dwExitCode:dword

.code
main proc
      mov eax,5
      add eax,6

      invoke ExitProcess,0
main endp
end main
```

Чтобы построить и запустить проект на выполнение нужно

1. Выбрать Build | Build Solution.

2. Выбрать Debug | Start Without Debugging.

Для отладки программы можно использовать стандартные средства, такие как выполнение по шагам (повторным нажатием клавиши F10), установка точек останова. В режиме отладки полезно выбрать Debug | Windows | Registers, для отображения содержимого регистров процессора. Нажатие правой кнопки мыши на окне Registers открывает меню, в котором можно выбрать пункт Flags, чтобы включить отображение отдельных флагов состояния из регистра EFLAGS.

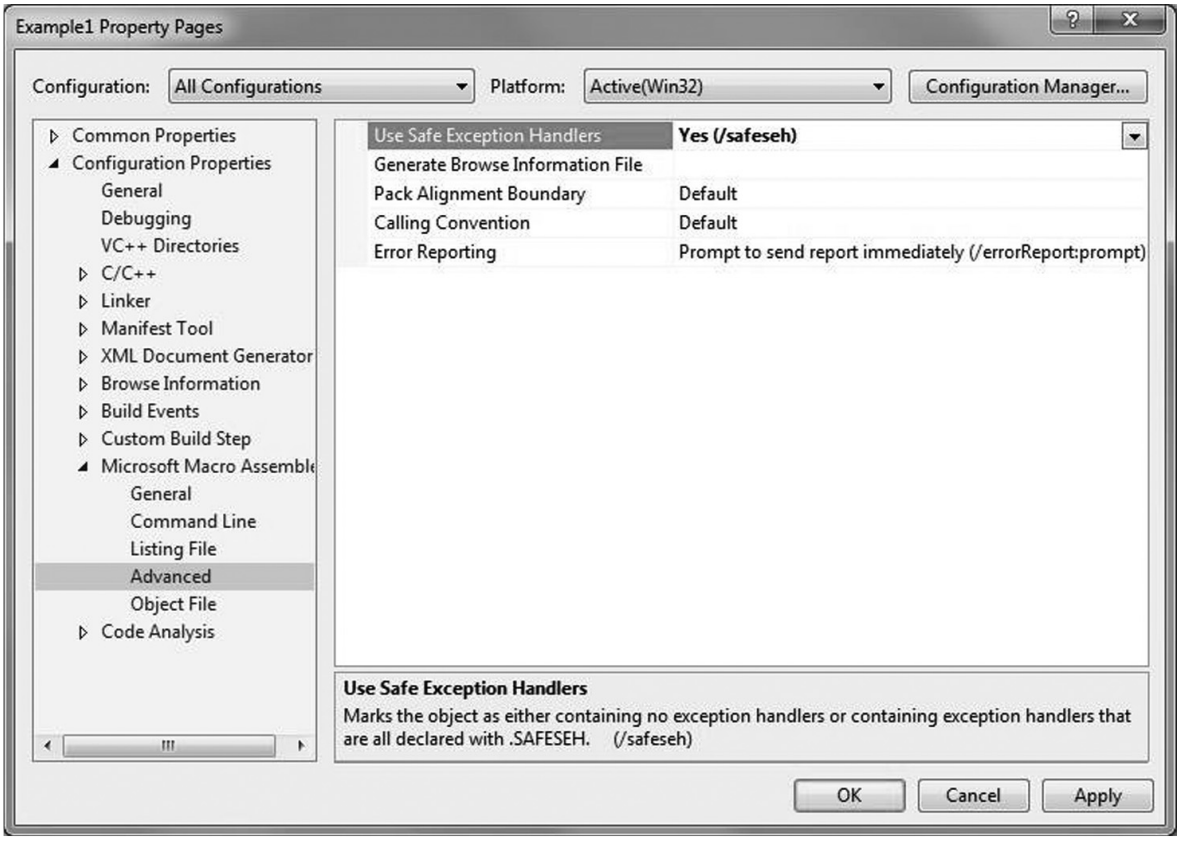
Если при построении проекта возникает ошибка компоновки «LNK1221: a subsystem can't be inferred and must be defined», то нужно в окне Solution Explorer правой кнопкой мыши нажать на название проекта и выбрать Properties | Linker | System. Убедиться, что для опции SubSystem выбрано значение Console (/SUBSYSTEM:CONSOLE).

Если необходимо, чтобы MASM генерировал объектные модули, совместимые с компилятором и компоновщиком C++, то нужно сделать следующее:

1. В окне Solution Explorer правой кнопкой мыши нажать на название проекта и выбрать Properties.

2. В открывшемся окне выбрать Configuration Properties | Microsoft Macro Assembler | Advanced.

3. Нажать на пункт Use Safe Exception Handlers и выбрать Yes (/safeseh).

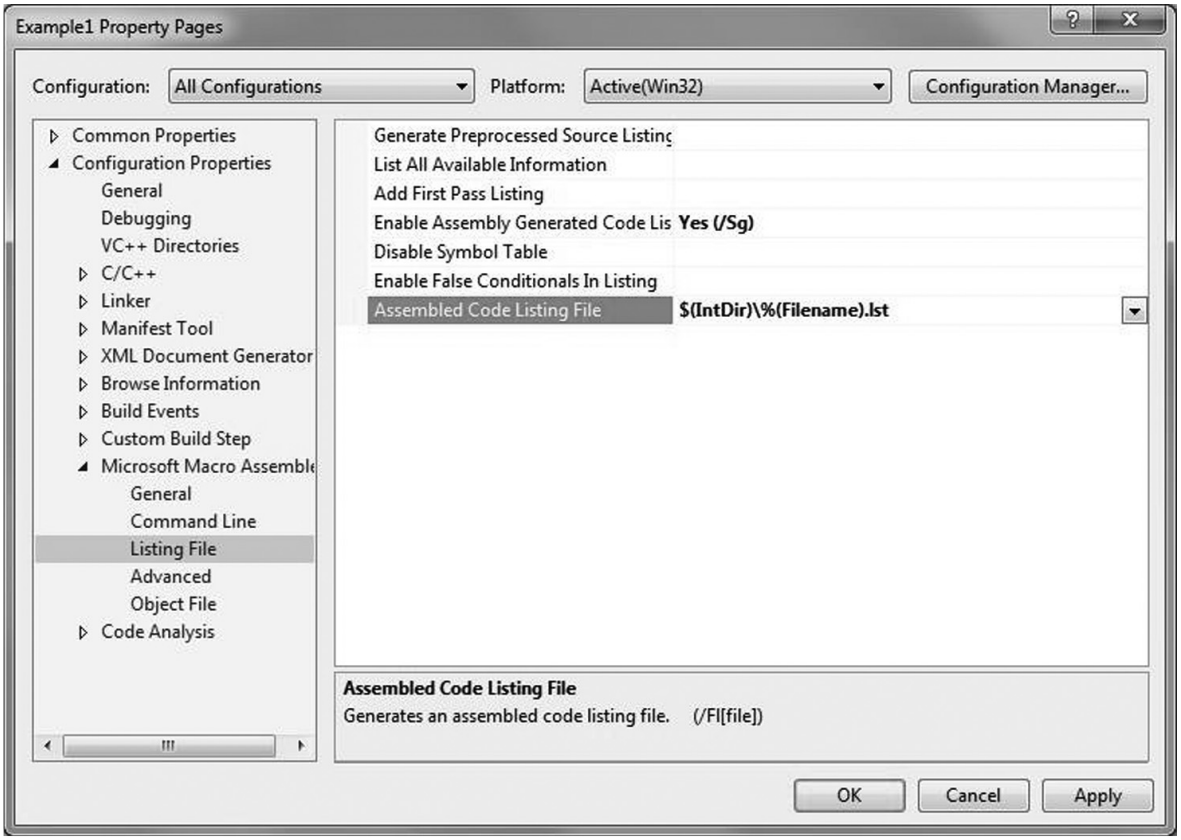


4. В этом же окне можно заставить MASM генерировать листинг-файл (очень полезен при анализе программы):

a. выбрать Configuration Properties | Microsoft Macro Assembler | Listing File

b. в пункте Enable Assembly Generated Code Listing выбрать Yes (/sg)

c. в пункте Assembled Code Listing File ввести текст $(IntDir)\%(Filename).lst



5. Нажать OK.

Вызов ассемблерных функций из C++

Для знакомства с системой команд IA-32 и основами программирования на языке ассемблера очень удобно использовать язык высокого уровня в качестве среды, поглощающей специфические детали взаимодействия с операционной системой. Тогда в программе, написанной на языке высокого уровня, можно легко организовать ввод-вывод данных для функций, написанных на языке ассемблера.

Допустим, требуется написать функцию на языке ассемблера, которая делает то же самое, что и следующая функция на C++:

```
int sum(int a, int b)
{
      return a + b;
}
```

Соответствующая функция для MASM может выглядеть так

```
.model flat, c

.code
sum proc

push ebp
mov ebp, esp
```

```
mov eax, [ebp + 8]
add eax, [ebp + 12]

pop ebp
ret

sum endp
end
```

При таком подходе функция на языке высокого уровня является спецификацией для соответствующей ассемблерной функции.

Ассемблерную процедуру sum нужно поместить в asm-файл. В этот же проект нужно добавить cpp-файл, например, с таким кодом:

```
#include <iostream>
using namespace std;

extern “C” int sum(int a, int b);
int main()
{
      int a = 5, b =3;
      int s = sum(5, 3);
      cout << “a = ” << a << endl;
      cout << “b = ” << b << endl;
      cout << “sum = ” << s << endl;
      return 0;
}
```

Примеры

1.1. Арифметические выражения

```
; C++: variable = 47;
mov variable, 47


; C++: var1 = var2;
mov eax, var2
mov var1 eax

; альтернативный способ (медленнее)
push var2
pop var1


; C++: x = y + z;
mov eax, y
add eax, z
```

```
mov x, eax


; C++: x = y - z;
mov eax, y
sub eax, z
mov x, eax


; C++: x = y * z; // unsigned
mov eax, y
mul z           ; результат в EDX:EAX
mov x, eax


; C++: x = y / z; // unsigned int
mov eax, y
mov edx, 0      ; расширение EAX путем заполнения нулями EDX
div z           ; деление EDX:EAX на z
mov x, eax


; C++: x = y / z; // signed int
mov eax, y
cdq             ; знаковое расширение EAX в EDX
idiv z          ; деление EDX:EAX на z
mov x, eax


; C++: x = y % z; // unsigned
mov eax, y
mov edx, 0      ; расширение EAX путем заполнения нулями EDX
div z           ; деление EDX:EAX на z
mov x, edx      ; забираем остаток из EDX


; C++: x = y % z; // signed
mov eax, y
cdq             ; знаковое расширение EAX в EDX
idiv z
mov x, edx      ; забираем остаток из EDX


; C++: var1 = -var2;
mov eax, var2
neg eax
mov var1, eax


; C++: w = (a + b) * (y + z);
;
; разбить на простые части
```

```
; temp1 = a + b;
; temp2 = y + z;
; w = temp1 * temp2;
; по возможности сохранять промежуточные значения в регистрах
;
mov eax, a
add eax, b
mov ebx, y
add ebx, z
imul ebx
mov w, eax


; C++: x = (y+z) * (a-b) / 10;
;
; разбить на простые части
; temp1 = (y+z)
; temp2 = (a-b)
; temp1 = temp1 * temp2
; x = temp1 / 10
; по возможности сохранять промежуточные значения в регистрах
;
mov eax, y      ; eax = y + z
add eax, z
mov ebx, a      ; ebx = a - b
sub ebx, b
imul ebx        ; edx:eax = eax * ebx
mov ecx, 10
idiv ecx        ; eax = edx:eax / 10
mov x, eax
```

1.2. Логические выражения

Несмотря на то, что для представления логических значений достаточно одного бита, в языке ассемблера для этих целей обычно используют один из стандартных типов (например, байт или слово). Для представления логического значения false можно использовать число 0, а для true — все ненулевые числа. Другой способ — представлять true и false числами 1 и 0 соответственно. Пусть логические переменные *a* и *b* представляются вторым способом, тогда:

```
; C++: y = a && b;
mov al, a
and al, b
mov y, al

; C++: y = a || b;
mov al, a
or al, b
mov y, al

; C++: b = !a;
mov al, a       ; Команда NOT сама по себе не даст
```

```
not al          ; правильный ответ для al = !al, т.к.
and al, 1       ; (not 0) не равно 1. Команда AND
mov b, al       ; исправляет проблему.

mov al, a       ; Другой способ вычислить b = !a;
xor al, 1       ; Инвертирует нулевой бит.
mov b, al


; C++: b = ((x == y) && (a <= d)) || ((z - a) != 5);
mov eax, x
cmp eax, y
sete al         ; AL = x == y
mov ebx, a
cmp ebx, d
setle bl        ; BL = a <= d
and bl, al      ; BL = (x == y) && (a <= d)
mov eax, z
sub eax, a
cmp eax, 5
setne al
or al, bl       ; AL = ((x == y) && (a <= d)) || ((z - a) != 5);
mov b, al
```

2.1. Ветвления

```
; C++:
; if(a == b) x = y;
; else b = b + 1;
            mov eax, a
            cmp eax, b
            jne ElsePart
            mov eax, y
            mov x, eax
            jmp EndOfIf
ElsePart:
            inc b
EndOfIf:


; C++:
; if( (( x > y ) && ( z < t )) || ( a != b ) )
;     f = d;
;
; лучше представить в виде
; if(a != b) f = d;
; else if(x > y)
;           if(z < t)
;                 f = d;
;
            mov eax, a
```

```
            cmp eax, b
            jne DoIf
            mov eax, x
            cmp eax, y
            jng EndOfIf
            mov eax, z
            cmp eax, t
            jnl EndOfIf
DoIf:
            mov eax, d
            mov f, eax
EndOfIf:
```

2.2. Циклы

```
; C++:
; while( val1 < val2 )
; {
;     val1++;
;     val2--;
; }
;
            mov eax, val1
BeginWhile:
            cmp eax, val2     ; if not (val1 < val2)
            jnl EndWhile      ; выйти из цикла
            inc eax
            dec val2
            jmp BeginWhile    ; повторить
EndWhile:
            mov val1, eax
```

3. Массивы. Строки

```
; Суммирование элементов массива
.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.data
myArray DWORD 1,2,3,4,5       ; разместить в памяти массив из 5 чисел

.code
main PROC
      mov edi, OFFSET myArray       ; EDI = адрес начала массива myArray
      mov ecx, LENGTHOF myArray     ; инициализировть счетчик длиной массива
      mov eax,0                     ; обнулить сумму
L1:                                 ; метка начала цикла
```

```
      add eax, [edi]                ; добавить к сумме значение элемента массива
      add edi, TYPE myArray         ; перейти к следующему элементу
      loop L1                       ; повторять пока ECX не станет равен 0
; команда loop по умолчанию использует ECX как счетчик, который автоматически
; уменьшается на единицу на каждом шаге цикла и сравнивается с нулем
      invoke ExitProcess,0
main ENDP
END main
```

; Пузырьковая сортировка массива

```
.386
.model flat,stdcall
.stack 4096

ExitProcess PROTO,dwExitCode:DWORD

.data
arr   DWORD 50h, 40h, 30h, 20h, 10h
Count = ($ - arr) / TYPE arr

.code
main PROC
      mov ecx, Count
      dec ecx
L1:   push ecx                ; сохранить счетчик внешнего цикла
      mov esi, OFFSET arr     ; ESI указывает на первый элемент
L2:   mov eax, [esi]          ; взять число из массива
      cmp [esi+4], eax        ; сравнить со следующим
      jg L3                   ; if ([ESI+4] > [ESI]) не обменивать
      xchg eax,[esi+4]        ; обменять значения двух элементов массива
      mov [esi], eax
L3:   add esi,4               ; продвинуть указатель на следующий элемент
      loop L2                 ; внутренний цикл
      pop ecx                 ; восстановить счетчик внешнего цикла
      loop L1                 ; повторить внешний цикл
L4:
      INVOKE ExitProcess,0
main ENDP
END main
```

; Копирование строки

```
.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.data
source BYTE "This is the source string",0 ; строка, завершающаяся нулевым байтом
```

```
target BYTE SIZEOF source DUP(0)               ; выделить место для копирования
строки

.code
main PROC
      mov esi,0               ; обнулить индексный регистр
      mov ecx, SIZEOF source  ; инициализировать счетчик цикла
L1:
      mov al, source[esi]     ; взять символ исходной строки
      mov target[esi], al     ; сохранить его во второй строке
      inc esi                 ; перейти к следующему символу
      loop L1                 ; повторить для всей строки

      invoke ExitProcess,0
main ENDP
END main
```

; Инверсия строки

```
.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO,dwExitCode:DWORD
.data
myString BYTE "Never odd or even",0
strSize = ($ - myString) - 1
.code
main PROC
; Занести строку в стек.
      mov ecx, strSize
      mov esi, 0
L1:   movzx eax, myString[esi]      ; взять символ строки (“zx” – zero extend)
      push eax                ; сохранить в стеке
      inc esi
      loop L1
; Извлечь строку из стека (в обратном порядке)
; и сохранить в исходный массив myString.
      mov ecx, strSize
      mov esi, 0
L2:   pop eax                       ; достать символ из стека
      mov myString[esi], al   ; сохранить в строке
      inc esi
      loop L2
      INVOKE ExitProcess,0
main ENDP
END main
```

; Обращение к элементу двумерного массива

```
.386
```

```
.model flat,stdcall
.stack 4096

ExitProcess PROTO,dwExitCode:DWORD

.data
; Пусть имеется таблица чисел из 3 строк и 5 столбцов.
; Расположим элементы таблицы в памяти построчно.
tableD      DWORD 10h, 20h, 30h, 40h, 50h
RowSize = ($ - tableD)  ; размер строки (в байтах)
            DWORD 60h, 70h, 80h, 90h, 0A0h
            DWORD 0B0h, 0C0h, 0D0h, 0E0h, 0F0h

; Допустим, нужно обратиться к элементу tableD[1][2]
row_index = 1
column_index = 2

.code
main PROC
      mov ebx, OFFSET tableD              ; адрес начала таблицы
      add ebx, RowSize * row_index        ; адрес начала нужной строки
      mov esi, column_index
      mov eax, [ebx + esi * TYPE tableD]  ; EAX = 80h

      INVOKE ExitProcess,0
main ENDP
END main
```

4. Функции

```
;-----------------------------------------------------
;int div(int a, int b, int* q, int* r)
;{
;     if (b == 0) return 0; // error code: divizion by zero
;     *q = a / b;
;     *r = a % b;
;     return 1;             // success code
;}
;-----------------------------------------------------

div proc

push ebp
mov ebp, esp

xor eax, eax                  ; eax = 0, error exit code
mov ecx, [ebp + 12]           ; ecx = b
or ecx, ecx                   ; обновляет флаги условий, не изменяя ecx
jz InvalidDiv

mov eax, [ebp + 8]            ; eax = a
cdq                           ; “convert doubleword to quadword”: после этого
```

```
                                ; edx:eax содержит делимое “a”
idiv ecx                        ; деление со знаком, edx:eax / ecx =>
                                ; eax - частное, edx - остаток
mov ecx, [ebp + 16]
mov [ecx], eax                  ; *q = eax
mov ecx, [ebp + 20]
mov [ecx], edx                  ; *r = edx
mov eax, 1                      ; success code

InvalidDiv:
pop ebp
ret

div endp



;-------------------------------------------------------
;int fact(int n)
;{
;     int f = 1;
;     while (n > 0)
;     {
;           f *= n;
;           n--;
;     }
;     return f;
;}
;-------------------------------------------------------
fact proc

push ebp
mov ebp, esp

mov eax, 1
mov ecx, [ebp + 8]

cmp ecx, 0
jle Done

Loopstart:
imul eax, ecx
; следующие две команды можно заменить на "loop Loopstart"
dec ecx
jg Loopstart

Done:
pop ebp
ret

fact endp
```

```
;-----------------------------------------------------
;int GCD(int a, int b)
;{
;     if (a < 0) a = -a;
;     if (b < 0) b = -b;
;     while (b > 0)
;     {
;           int r = a % b;
;           a = b;
;           b = r;
;     }
;     return a;
;}
;-----------------------------------------------------
GCD proc

push ebp
mov ebp, esp

mov eax, [ebp + 8]
mov ecx, [ebp + 12]

; вычисление модуля eax: abs(x) = (x XOR (x >>> 31)) - (x >>> 31)
mov edx, eax
sar edx, 31
xor eax, edx
sub eax, edx
; модуль ecx
mov edx, ecx
sar edx, 31
xor ecx, edx
sub ecx, edx

BeginLoop :
xor edx, edx
div ecx
mov eax, ecx
mov ecx, edx
cmp ecx, 0
jg BeginLoop

pop ebp
ret

GCD endp
```

Задания

1. Арифметические и логические выражения

1.1. Вычислить значение выражения

1. $y = (b^2 – (a + 1) * c) / b$
2. $y = a / c – b + (d + 1) * 5$
3. $y = a * b – b^2 / (c + 2)$
4. $y = a * (a + b / 4) / (c – 1)$
5. $y = 3 * a * c / (5 * (b – 5))$
6. $y = a * c – 3 * (b + 3 / c)$
7. $y = a^3 / 3 – c * (b + 3)$
8. $y = (b – 5)^2 / 4 + 2 * b$
9. $y = a * c / 2 – (a + b) / 2$
10. $y = (b^2 – 2 * b) / (3 * a + b)$
11. $y = (a^2 – b^2) / 2 + a * (c + 1)$
12. $y = (a – c)^2 + 2 * a * c / b$
13. $y = (a^3 – 1) / (b – 4) – 5$
14. $y = b^2 * (a + c) + (d – 1) / c$
15. $y = b^3 – 2 * a * b + a^2 / b$
16. $y = b^2 / 3 – a * c + 5$
17. $y = a * c^2 – b * a / c + a / b$
18. $y = a * c * (b – a) / 4 + a^2 - 2$
19. $y = a * c^2 – b * c / a + d / (b + a)$
20. $y = (l – a)^2 / c + b – l + c / 2$
21. $y = (a – b^2) / (c – a) + a^2 – c$
22. $y = (b – 5) * (b + 2) + b + a / 2$
23. $y = (a + b) / c – c^2 * a – b$
24. $y = b * (c – a) – c / (d – 1)$

1.2. Вычислить результат логического выражения T = S для заданных значений логических переменных a, b, c.

+ — логическое сложение (логическое «или»)

· — логическое умножение (логическое «и»)

¯ —логическое отрицание (логическое «не»)

1. $T = a \cdot \overline{b \cdot \bar{c}}$ $S = a \cdot \bar{b} + a \cdot c$
2. $T = a + \bar{b} \cdot \bar{c}$ $S = a + \overline{b + c}$
3. $T = (a + b) \cdot \bar{c} \cdot \bar{d}$ $S = a \cdot \overline{(c + d)} + b \cdot \overline{(c + d)}$
4. $T = a \cdot \bar{b} + a \cdot \bar{c}$ $S = a \cdot \overline{b \cdot c}$
5. $T = \overline{(b + c)} \cdot d$ $S = \bar{b} \cdot \bar{c} \cdot d$
6. $T = (\bar{b} + \bar{c}) \cdot \bar{d}$ $S = \overline{(b + d)} + \overline{(c + d)}$
7. $T = (a + b) \cdot (\bar{c} + \bar{d})$ $S = a \cdot \overline{(c \cdot d)} + b \cdot \overline{(c \cdot d)}$
8. $T = \overline{(a + b) \cdot (\bar{c} + \bar{d})}$ $S = \overline{(a + b)} + c \cdot d$
9. $T = \overline{(\bar{a} \cdot b) + (c \cdot \bar{d})}$ $S = (a + \bar{b}) \cdot (\bar{c} + d)$
10. $T = \overline{(\bar{a} + b + c) \cdot \bar{d})}$ $S = a \cdot \overline{(b + c)} + d$
11. $T = \overline{(a + b + \bar{c}) \cdot d)}$ $S = \overline{(a + b)} \cdot c + \bar{d}$
12. $T = \overline{(a + \bar{b} + c) \cdot d)}$ $S = \overline{(a + c)} \cdot b + \bar{d}$

1.3. Побитовые операции

1. Задано число n в формате char. Сбросить биты 1, 3.
2. Задано число n в формате unsigned char. Установить биты 2, 4.
3. Задано число n в формате signed char. Инвертировать биты 3, 6.
4. Задано число n в формате short int. Сбросить биты 1, 7.
5. Задано число n в формате unsigned short int. Установить биты 7, 8.
6. Задано число n в формате signed short int. Инвертировать биты 1, 15.
7. Задано число n в формате int. Сбросить биты 7, 15.
8. Задано число n в формате unsigned int. Установить биты 7, 15.
9. Задано число n в формате signed int. Инвертировать биты 1, 7.
10. Задано число n в формате unsigned long. Сбросить биты 1, 10.
11. Задано число n в формате signed long. Установить биты 0, 1.

12. Задано число n формате long long. Инвертировать биты 2, 15.

2. Ветвления. Циклы

2.1. Ветвления

1. Обменять значения целочисленных переменных x, y, z так, чтобы $x \geq y \geq z$.

2. Обменять значения целочисленных переменных x, y, z так, чтобы $x < y < z$.

3. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 < x2 < x3 < x4$.

4. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 > x2 > x3 > x4$.

5. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 \geq x2 < x3 \geq x4$.

6. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 < x2 > x3 < x4$.

7. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 \geq x2 \geq x3 \geq x4$.

8. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 < x2 > x3 > x4$.

9. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 > x2 > x3 < x4$.

10. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 < x2 > x3 > x4$.

11. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 \geq x2 < x3 \geq x4$.

12. Обменять значения целочисленных переменных $x1, x2, x3, x4$ так, чтобы $x1 < x2 > x3 < x4$.

2.2. Поле шахматной доски определяется парой натуральных чисел, каждое из которых не превосходит 8: первое — номер вертикали, второе — номер горизонтали. Заданы натуральные числа k, l, m, n.

1. Определить одного ли цвета поля (k, l) и (m, n).

2. На поле (k, l) расположен слон. Угрожает ли он полю (m, n)?

3. На поле (k, l) расположен ферзь. Угрожает ли он полю (m, n)?

4. На поле (k, l) расположен конь. Угрожает ли он полю (m, n)?

5. Можно ли с поля (k, l) одним ходом ладьи попасть на поле (m, n)?

6. На поле (k, l) стоит ладья, на поле (m, n) — слон. Определить, бьет ли ладья слона, слон — ладью или фигуры не угрожают друг другу.

7. На поле (k, l) стоит ладья, на поле (m, n) — ферзь. Определить, бьет ли ладья ферзя, ферзь — ладью или фигуры не угрожают друг другу.

8. На поле (k, l) стоит ладья, на поле (m, n) — конь. Определить, бьет ли ладья коня, конь — ладью или фигуры не угрожают друг другу.

9. На поле (k, l) стоит ладья, на поле (m, n) — пешка. Определить, бьет ли ладья пешку, пешка — ладью или фигуры не угрожают друг другу.

10. На поле (k, l) стоит ферзь, на поле (m, n) — слон. Определить, бьет ли ферзь слона, слон — ферзя или фигуры не угрожают друг другу.

11. На поле (k, l) стоит слон, на поле (m, n) — конь. Определить, бьет ли слон коня, конь — слона или фигуры не угрожают друг другу.

12. На поле (k, l) стоит ферзь, на поле (m, n) — конь. Определить, бьет ли ферзь коня, конь — ферзя или фигуры не угрожают друг другу.

2.3. Циклы

1. Найдите сумму первых n натуральных чисел, которые являются степенью числа 5.

2. Найдите сумму первых n натуральных чисел, которые делятся на 3.

3. Найти сумму первых n членов геометрической прогрессии с первым членом 1 и знаменателем -2.

4. Найдите сумму первых n натуральных чисел, которые являются полными квадратами.

5. Найдите сумму первых n натуральных чисел, которые являются степенью числа 3.

6. Найдите сумму первых n натуральных чисел, которые делятся на 5.

7. Найдите сумму первых n натуральных чисел, которые делятся на 6.

8. Найдите сумму первых n натуральных чисел, которые делятся на 9.

9. Найдите сумму первых n натуральных чисел, которые делятся на 3 и 5.

10. Найдите сумму первых n натуральных чисел, которые делятся на 3 и 10.

11. Найдите сумму первых n натуральных чисел, которые делятся на 2 или 5.

12. Найдите сумму первых n натуральных чисел, которые делятся на 3 или 5.

2.4. Циклы.

1. Дано натуральное n. Верно ли, что это число содержит только две одинаковые цифры?

2. Дано натуральное n. Верно ли, что это число содержит ровно три одинаковые цифры?

3. Дано натуральное n. Определить, является ли это число палиндромом.

4. Дано натуральное n. Верно ли, что все цифры числа различны?

5. Дано натуральное n. Верно ли, что это число содержит ровно k одинаковых цифр?

6. Дано натуральное n. Верно ли, что все цифры числа различны?

7. Дано натуральное *n*. Верно ли, что это число содержит ровно *2* одинаковые цифры?

8. Дано натуральное *n*. Верно ли, что это число содержит более *k* одинаковых цифр?

9. Определить, равна ли сумма *k* первых цифр заданного натурального числа, сумме *k* его последних цифр.

10. Дано натуральное *n*. Верно ли, что это число содержит более *k* одинаковых цифр?

11. Дано натуральное *n*. Верно ли, что это число содержит *k* цифр, значения которых меньше заданного *m*?

12. Дано натуральное *n*. Верно ли, что это число содержит более *k* цифр больших суммы первых *2* цифр.

3. Массивы

3.1. Одномерные массивы

1. Найти количество тех элементов массива, которые встречаются в нем ровно два раза.
2. Найти сумму элементов массива, расположенных до минимального.
3. Найти сумму тех элементов массива, которые являются степенью числа *5*.
4. Найти сумму тех элементов массива, которые встречаются по одному разу.
5. Найти сумму элементов массива, расположенных до максимального.
6. Найти сумму тех элементов массива, индексы которых делятся на *2* или 3.
7. Найти сумму элементов массива, расположенных между минимальным и максимальным значениями.
8. Найти сумму тех элементов массива, которые делятся на *3* и *5*.
9. Найти сумму 3 наибольших элементов массива.
10. Найти сумму тех элементов массива, которые делятся на *3*.
11. Найти сумму 5 наименьших элементов массива.
12. Найти сумму элементов массива, расположенных после максимального.
13. Найти сумму тех элементов массива, которые делятся на *3* или *5*.
14. Найти сумму элементов массива, расположенных после минимального.
15. Найти сумму тех элементов массива, которые являются полными квадратами.
16. Найти сумму каждого третьего элемента массива.
17. Найти сумму тех элементов массива, которые делятся на *5*.

18. Найти сумму элементов массива, расположенных до максимального из отрицательных элементов.

19. Найти количество тех элементов массива, которые встречаются в массиве более одного раза.

20. Найти сумму элементов массива, расположенных после минимального из положительных элементов.

21. Найти наибольшее из чисел массива, встречающихся в нем ровно один раз.

22. Найти сумму чисел, расположенных между минимальным положительным и максимальным отрицательным элементами.

23. Найти сумму тех элементов массива, которые делятся на *3* и *10*.

24. Найти наименьшее из чисел массива, встречающихся в нем более одного раза.

3.2. Одномерные массивы

1. Получить за один просмотр массив $C(K)$, упорядоченный по возрастанию, путем слияния массивов $A(N)$ и B(M), упорядоченных по возрастанию ($K = N + M$).

2. Вычислить скалярное произведение заданных векторов a и b длины n.

3. Дан массив целых чисел, содержащий n элементов. Получить массив, в котором записаны сначала все отрицательные числа и нули, затем все положительные числа, сохраняя порядок следования.

4. Отсортировать массив методом выбора.

5. Заданы два одномерных массива $X(n)$, $Y(m)$, причем $0 \le Y_i < n$; $m \le n$, $Y_i \ne Y_j$. Вычислить сумму тех элементов массива X, индексы которых совпадают со значениями элементов массива Y.

6. Отсортировать массив методом вставки.

7. Из двух массивов $A(N)$ и $B(M)$, упорядоченных по возрастанию, получить за один просмотр массив $C(K)$, также упорядоченный по возрастанию, в который включить пересечение элементов двух исходных массивов.

8. Найти наименьшее среди чисел первой последовательности, не входящих во вторую.

9. Дан массив целых чисел, содержащий n элементов. Получить массив, в котором записаны сначала все положительные числа, затем все отрицательные числа и нули, сохраняя порядок следования.

10. Из двух массивов $A(N)$ и $B(M)$, упорядоченных по возрастанию, получить за один просмотр массив $C(K)$, также упорядоченный по возрастанию, в который включить элементы первого массива, исключив из них элементы второго массива.

11. Найти наибольшее среди чисел первой последовательности, входящих во вторую.

12. Заданы два одномерных массива $X(n)$, $Y(m)$. Причем $0 \le Y_i < n$; $m \le n$, $Y_i \ne Y_j$. Вычислить сумму тех элементов массива X, индексы которых не совпадают со значениями элементов массива Y.

3.3. Двумерные массивы

1. Транспонировать заданную матрицу.

2. Вычислить сумму произведений элементов строк заданной матрицы.

3. Для заданной матрицы A получить матрицу $A\ A^T$.

4. Упорядочить строки матрицы по убыванию их первых элементов.

5. Поменять местами строку, содержащую элемент с наибольшим значением в матрице, со строкой, содержащей элемент с наименьшим значением.

6. Вычислить сумму произведений элементов столбцов заданной матрицы.

7. Для заданных матриц A, B получить матрицу $C = A\ B - B\ A$.

8. Создать массив из минимальных значений сумм столбцов заданной матрицы.

9. Упорядочить строки матрицы по возрастанию их последних элементов.

10. Найти максимальный и минимальный элементы среди стоящих на главной и побочной диагоналях матрицы и обменять их местами.

11. Вычислить произведение сумм элементов строк заданной матрицы.

12. Дана матрица A размерности $n \times n$. Получить матрицу $B = E - A + A^2 / 2$.

13. Создать массив из максимальных значений сумм строк заданной матрицы.

14. Упорядочить столбцы матрицы по убыванию их первых элементов.

15. Среди строк заданной матрицы, содержащих только нечетные элементы, найти строку с максимальной суммой элементов.

16. Вычислить произведение сумм элементов столбцов заданной матрицы.

17. Дана матрица A размерности $n \times n$. Получить матрицу $B = E - A^2 / 2$.

18. Упорядочить столбцы матрицы по возрастанию их последних элементов.

19. Найти сумму элементов того столбца, в котором находится наименьший элемент матрицы.

20. Создать массив из минимальных значений сумм строк заданной матрицы.

21. Дана матрица A размерности $n \times n$. Получить матрицу $B = A - A^3 / 6$.

22. Создать массив из максимальных значений сумм столбцов заданной матрицы.

23. Для заданных матрицы A и вектора b, получить вектор $c = A^2\ b$.

24. Для заданных матрицы A и вектора x получить число $x^T A x$.