2. ЛАБОРАТОРНАЯ РАБОТА № 2 «ПРИНЦИПЫ ВЫПОЛНЕНИЯ КОМАНД ВЕТВЛЕНИЯ, ОРГАНИЗАЦИЯ ЦИКЛОВ И ПОДПРОГРАММ»

Цель работы: изучение принципов выполнения команд ветвления, организации циклов и подпрограмм микропроцессоров с архитектурой x86.

1. Команды управления переходами

Для того чтобы иметь возможность переходить по фиксированному адресу и создавать ветвящиеся алгоритмы в системе команд x86 имеются специальные команды безусловного (например, `jmp`) и условного перехода (например, `jc`).

Команды безусловного перехода можно разбить на команды относительного (относительно значения `eip` после выполнения команды `jmp`) перехода и абсолютного перехода. Адрес при относительном переходе определяется смещением со знаком, следующим за кодом команды. Среди относительных переходов различают «короткие» (jmp short) переходы (код `0EB`h и смещение в виде байта), «ближние» (jmp near), иногда называемые внутрисегментными переходами, (код `0E9`h и смещение в виде двух байт) и переходы со смещением в виде четырёх байт.

В командах абсолютного перехода адрес может быть задан прямо в виде значения сегмента и смещения («длинный» абсолютный переход (`jmp far`), код `0EA`h) и косвенно, через значения, содержащиеся в регистрах или памяти («длинный» косвенный переход, адрес содержится в регистрах общего назначения или в памяти, код `0FF`h).

При выполнении команд условного перехода сначала проверяется соответствующее условие (состояние какого-либо флага), и, только если это условие истинно, осуществляется переход, иначе выполняется следующая команда. Для организации циклов используются команды условного перехода группы `loop`, выполнение которых зависит от содержимого регистра-счётчика `(e)cx`.

Рассмотрим в качестве примера команды `jmp short Address` (короткий переход) и `jmp far cs:ip`.

Команда короткого перехода применяется, когда нужно осуществить переход в пределах от минус 128 до плюс 127 байт. Адрес задаётся в виде байта смещения,

интерпретируемого как число со знаком в дополнительном коде. Пример команды `jmp short Address:`

```
    ORG          04020h
    MyDataAddress    dw   0
    ORG      4000h
EB 1E        jmp short MyDataAddress
```

Довольно часто используют следующую форму записи короткого перехода:

```
EB 1E        jmp $+10
```

Здесь знак $ означает положение `ip`. Объясните, почему в коде команды стоит число `1Eh`.

Мнемоника длинного перехода ассемблером не воспринимается, поэтому обычно длинный переход задаётся в виде последовательности байт:

```
DB 0EAh, 20h, 00, 00h, 00h,
```

где `0EAh` — код команды длинного абсолютного перехода

Подпрограммы. При разработке программ нужно стараться сделать их как можно короче и компактнее. С этой целью часть программы, которая неоднократно повторяется, или программа, которая часто используется, могут быть оформлены в виде подпрограмм. *Подпрограмма* — последовательность команд, заканчивающаяся командой возврата, выполнение которой может быть вызвано из любого места программы любое количество раз.

Подпрограммы изначально появились как средство оптимизации программ по объёму занимаемой памяти и экономии времени программиста. В настоящее время, кроме этого, подпрограммы выполняют ещё функцию структуризации программы с целью удобства её понимания и сопровождения, что особенно важно в языках высокого уровня.

При вызове подпрограммы, как и в случае выполнения команды перехода, управление передаётся команде, адрес которой содержится в команде вызова подпрограммы. Но, в отличие от команд перехода, вызов подпрограммы, после выполнения собственно подпрограммы, должен обеспечить выполнение команды, следующей за командой вызова подпрограммы. Поэтому любая подпрограмма должна заканчиваться командой возврата из подпрограммы. Для обеспечения механизма возврата в точку вызова используется специальное средство, называемое **стеком (stack)**.

Стек — память, функционирующая по так называемому принципу LIFO (Last

In, First Out — первым вошёл, последним вышел), и первоначально предназначенная для хранения адресов возврата из подпрограмм. Стеки бывают аппаратные и программные, в микропроцессоре i80386 используется программный стек, то есть стек размещается в ОЗУ. Стек адресуется при помощи пары регистров `ss:sp`. Регистр `ss` (stack segment) содержит адрес сегмента стека, `(e)sp` (stack pointer, указатель стека) — смещение относительно начала сегмента стека.

Процесс передачи управления подпрограмме, называется вызовом подпрограммы. Для вызова подпрограммы и возврата из нее в языке ассемблера используются команды `call` и `ret`, которые при выполнении используют стек. При необходимости подпрограмме можно передавать аргументы (*входные параметры*), в качестве которых могут выступать либо непосредственно данные, либо их адреса. Результаты работы программы, передаваемые по окончании её работы в основную программу, называются *выходными параметрами*.

Синтаксис команды `call` показан на рисунке 4.7. Необязательные модификаторы `near` и `far` указывают тип вызова: соответственно ближний (внутрисегментный), когда вызываемая подпрограмма находится в том же сегменте, что и вызывающая, или дальний (межсегментный), когда вызываемая подпрограмма находится в другом сегменте. Отсутствие модификатора указывает на ближний вызов. Вместо символического имени подпрограммы в команде можно указать её адрес.

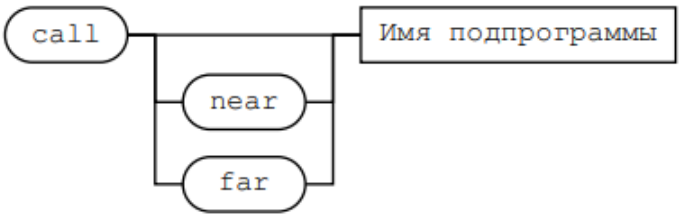

Рисунок 4.7 — Синтаксис команды `call`

В зависимости от типа вызова команда `call` адреса возврата сохраняются несколько по-разному. При ближнем вызове микропроцессор сохраняет в стеке содержимое регистра `ip`, которое, по сути, является адресом возврата (адресом следующей за `call` команды), после чего помещает в регистр `ip` адрес перехода. Что происходит при дальнем вызове, читателю предлагается узнать самостоятельно.

Команда `ret` (ближний возврат) берёт из стека последнее, помещённое туда слово, и помещает его в регистр `ip`, после этого выполнение программы продолжится с этого адреса. Как уже говорилось, любая подпрограмма должна заканчиваться командой возврата (`ret`, `retf`, `iret`, `ret N`). Самостоятельно

выясните, в чем различие этих команд возврата.

Автоматическое сохранение и восстановление адреса основной программы при выполнении подпрограммы позволяет сделать подпрограммы вложенными (nested procedures), т. е. осуществлять вызовы одной подпрограммы из другой. Уровень вложенности определяется лишь размером стека.

Для работы со стеком можно использовать команды `push r` (записать в стек содержимое обозначенного регистра) и `pop r` (записать данные из стека в обозначенный регистр). Эти команды являются однобайтовыми [3]. Начиная с микропроцессора i80386 при работе со стеком можно использовать непосредственную адресацию.

Рассмотрим работу МП со стеком на примере команды `pushf`.

`pushf` — команда сохраняет младшее слово регистра флагов `eflags` в стеке, т. е. сохраняет 16-разрядный регистр флагов `flags`. Если рассматривать процесс выполнения команды, то можно отметить, что при 8-битной шине данных первым в стек помещается старший байт регистра `flags` по адресу `sp - 1`, а потом младший байт по адресу `sp - 2`, где `sp` — адрес ячейки до обращения к стеку (т. е. до выполнения команды `pushf`). Такой формат размещения в стеке называется Little endian (или Intel формат). При работе с 16-битной шиной данных в стек будут сразу помещаться слова. Пример работы со стеком представлен на рисунке 4.8. После выполнения команды `sp` = 21FEh, — указывает на младший байт регистра `flags`.

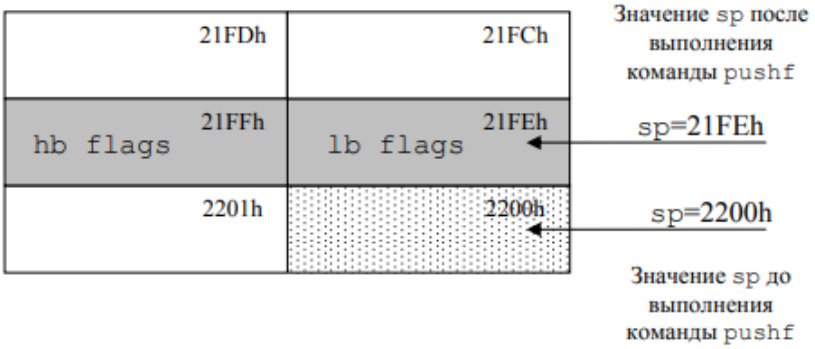

Рисунок 4.8 — Работа МП со стеком на примере команды `pushf`

При извлечении значения из стека (по команде `popf`), первым извлекается младший байт слова, а затем старший, а `sp` увеличивается на 2.

Таким образом, при записи данных в стек, значение `sp` уменьшается, т. е. стек растёт по направлению к младшим адресам. Если используется представление памяти с младшими адресами, расположенными вверху, то говорят, что стек растёт вверх.

Как следует из приведённых выше сведений, значение указателя стека `sp` всегда должно быть чётным.

Передача параметров. Часто основная программа должна передать вызываемой подпрограмме определённую информацию, данные, с которыми подпрограмма должна работать. И часто подпрограмма после завершения работы должна передать результат своего выполнения в вызвавшую её программу.

Передаваемые подпрограмме данные называют параметрами или, иначе, аргументами. Существует три общепринятых способа передачи параметров: через регистры, через общую область памяти и через стек [14].

Передача параметров в регистрах очень проста. Для этого нужно просто поместить значения-параметры в соответствующие регистры и вызвать подпрограмму. Каждая подпрограмма может иметь свои собственные потребности в параметрах, и чтобы избежать путаницы, лучше выработать некоторые соглашения передачи параметров и придерживаться их. Например, можно следовать правилу, согласно которому первый параметр-указатель всегда передается в регистре `bx`, второй — в `si` и т.д. Следует взять за правило особенно аккуратно комментировать каждую подпрограмму — какие параметры она получает, в каких регистрах они находятся.

В большинстве языков высокого уровня, включая Паскаль и Си, и в подпрограммах на Ассемблере, вызываемых из этих языков [14], передача параметров обычно осуществляется через стек. Передача параметров через стек несколько более сложна и отличается значительно меньшей гибкостью, чем передача через регистры. Передача параметров заключается в том, что вызывающая программа предварительно помещает в стек в определённом порядке параметры, требуемые для выполнения подпрограммы, а затем вызывает эту подпрограмму. Пример передачи параметров через стек представлен на рисунке 4.9 [10].

Вызванная подпрограмма непосредственно выполняет операции над данными, расположенными в стеке. Поскольку при работе со стеком микропроцессор автоматически использует регистры `ss` и `sp`, то при передаче параметров не рекомендуется записывать в эти регистры сведения о количестве передаваемых параметров. Обычно для этих целей используется регистр `(e)bp`, который по умолчанию также настроен для работы со стеком и в который записывается текущее значение указателя стека `(e)sp`. Выглядит это следующим образом:

```
push bp
mov bp, sp
```

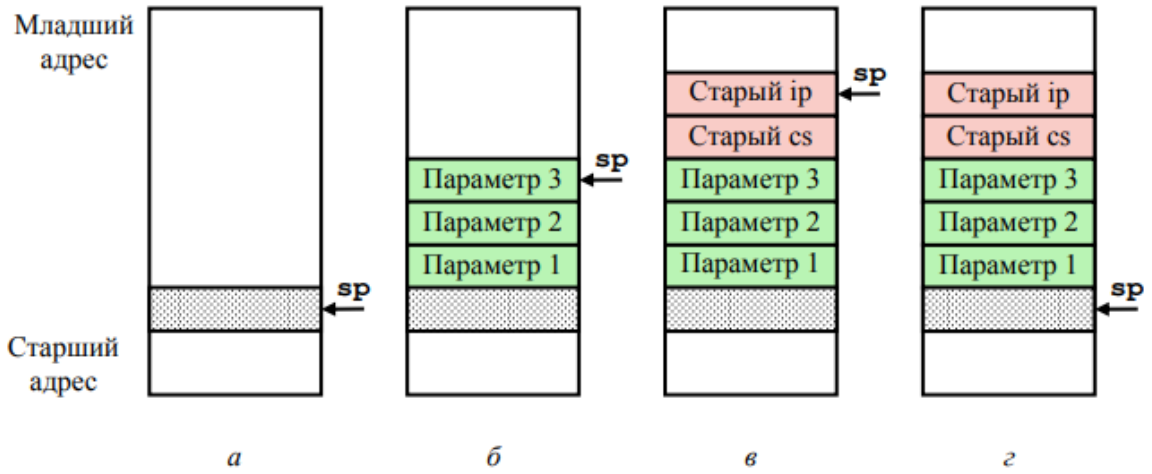

Рисунок 4.9 — Передача параметров через стек: *а* — стек до вызова подпрограммы; *б* — в стек поместили три параметра; *в* — стек после выполнения команды `call`; *г* — стек после возврата из подпрограммы командой `ret 3`

Теперь доступ к стеку можно осуществлять с помощью регистра `bp`.

В конце подпрограммы обычно используют команду `ret N` (`ret` со смещением). При выполнении команда `ret N` дополнительно добавляет значение смещения N к указателю стека `sp`. Это позволяет игнорировать параметры, помещаемые в стек перед вызовом подпрограммы. Рассмотрим на примере, представленном на рисунке 4.9, выполнение команды возврата без смещения и со смещением. Если при выходе из подпрограммы мы будем использовать команду `ret`, то после выполнения этой команды из стека будет восстановлено значение `ip` и `cs`, а `sp` будет указывать на ячейку памяти содержащую параметр 3. Если данная подпрограмма вызывается неоднократно, то рано или поздно наступит переполнение стека. Чтобы этого избежать, необходимо использовать команду `ret N`. При выполнении этой команды из стека будет также восстановлено значение `ip` и

cs, к этому моменту sp будет указывать на параметр 3; затем к значению sp будет прибавлено значение смещения, указанного в команде. В нашем случае, если параметры однобайтные, то значение смещения должно равняться трём. В результате sp будет иметь исходное значение, которое было до загрузки параметров и вызова подпрограммы.

Для того чтобы использовать подпрограммы на Ассемблере в программах, написанных на языках высокого уровня, необходимо знать и корректно использовать правила (соглашения) передачи параметров через стек, принятые в соответствующем языке высокого уровня.

При передаче параметров, как через регистры, так и через стек, передаваться могут как непосредственно сами значения параметров, так и указатели на них. Обычно в роли указателя выступает адрес ячейки памяти, где и расположен сам параметр. При работе с указателями для получения доступа к значению параметра часто используются следующие методы адресации: косвенно-регистровая (indirect), относительная (base-displacement addressing), индексная (indexed addressing) и относительно-индексная (based indexed addressing).

2. Индивидуальное задание

Создайте новый проект в Visual Studio (См. мет. указания к лабораторной работе №1).

Напишите программу на ассемблере, реализующую вычисления в соответствие с вариантом. В переходах по адресу необходимо использовать подпрограммы.

Подобрать входные данные таким образом, чтобы проверить правильность выполнения программы при всех вариантах ветвлений и переходов.

<table>
<tr><td colspan="2">Дано: X=87A3 Y=5322 Z=07F1
(размещены в памяти один за другим)</td><td>ЗАДАНИЕ № 1</td></tr>
<tr><td>В цикле сдвинуть циклически каждое число X, Y, Z на 3 разряда вправо
(результат X',Y',Z')</td><td>Вычислить M=X'-Z'+Y'
M>0 переход к п/п 1 (R=M+5)
M≤0 переход к п/п 2 (R=-M)</td><td>Если R>007D, то переход к АДР1 (R/2), иначе переход к АДР2 (R or 17D1)
{R число со знаком}</td></tr>
</table>

Дано: X=D5A9 Y=31FF Z=5555		ЗАДАНИЕ № 2
Уменьшить X на 1 В цикле 3 раза к Y прибавить новое X' (результатом будет L)	Вычислить M=X' or Z M>10E8 переход к п/п 1 (R=M-211F) M≤10E8 переход к п/п 2 (R=M+01D0) {M - число без знака}	Если в мл. R нечетное количество единиц,то переход к АДР1 (R/2), иначе переход к АДР2 (R xor OF91)
Дано: X=6DA9 Y=11FA		ЗАДАНИЕ № 3
В цикле 4 раза из X вычесть Y (результатом будет L)	Вычислить M=L xor Y M>0 переход к п/п 1 (R=M & OFOF) M≤0 переход к п/п 2 (R=-M)	Если R=0, то переход к АДР1 (R=27E1 xor L), иначе переход к АДР2 (R+67A1 - L)
Дано: X=13DD Y=715F Z=02FE		ЗАДАНИЕ № 4
Увеличить X на 007D В цикле 3 раза из Y вычесть Z (результатом будет X')	Вычислить M=X'+ X M>OF99 переход к п/п 1 (R=M/2+(Z & 09AB)) M≤OF99 переход к п/п 2 (R=M-019B-cf) {M - число со знаком}	Если R положительный, то переход к АДР1 (R+1) если отрицательный, то переход к АДР2 (R-1)
Дано: X=E773 Y=717F Z=26AA (расположены в памяти один за другим)		ЗАДАНИЕ № 5
X увеличить на "1" В цикле сложить X, X', Y и Z (результат M)	M>0 переход к п/п 1 (R=M*2 xor Y) M≤0 переход к п/п 2 (R=M-(Z & OOFF))	Если произошло переполнение,то переход к АДР1 (R+OF91), иначе переход к АДР2 (R xor D1F1)
Дано: X=EE21 Y=4299 Z=9544 Q=545A (расположены в памяти один за другим)		ЗАДАНИЕ № 6
В цикле из каждого X, Y, Z, Q вычесть единицу (результат X', Y', Z', Q')	Вычислить M=(X'& Z')or(Y'xor Q') M≥0 переход к п/п 1 (R=M/2+1344) M<0 переход к п/п 2 (R=M*2-092C)	Если R≥31B8, то переход к АДР1 (R+221B), иначе переход к АДР2 (R xor 913C) {R число со знаком}
Дано: X=5429 Y=7844 Z=AD43 Q=5622 (расположены в памяти один за другим без знака)		ЗАДАНИЕ № 7
В цикле каждое из X, Y, Z, Q увеличить на единицу и все сложить (результат X', Y', Z', Q' и L)	Вычислить M=(L & X')-(L & Y') M≥921B переход к п/п 1 (R=M/2-12B9) M<921B переход к п/п 2 (R=M-Q'/2)	Если R четное, то переход к АДР1 (R or 009F) иначе переход к АДР2 (R-1)

Дано: X=2231 Y=48B3 Z=6BB8 (размещены в памяти один за другим)		ЗАДАНИЕ № 8
В цикле из F291 вычесть X, Y, Z (результат L)	Вычислить M=L/2+(X & Y) M<99F переход к п/п 1 (R=ст.M<=>мл.M) M≥99F переход к п/п 2 (R=M+10BA) {M - число без знака}	Если в мл. R четное количество единиц, то переход к АДР1 (R & F0F0), иначе переход к АДР2 (R xor 0F0F)
Дано: X=99C5 Y=2A6C Z=80CF (размещены в памяти один за другим)		ЗАДАНИЕ № 9
В цикле у каждого из X, Y, Z обменять местами байты (результатом будет X', Y', Z')	Вычислить M=X'+Y'-Z' M≥012B переход к п/п 1 (R=M/2-0012) M<012B переход к п/п 2 (R=M+388A) {M - число со знаком}	Если R положительное, то переход к АДР1 (R or 0FF0), если - отрицательное, то переход к АДР2 (-R xor 5555)
Дано: X=714A Y=6B15 Z=431B (расположены в памяти один за другим)		ЗАДАНИЕ № 10
В цикле сложить по "модулю 2" числа X, Y, Z (результатом будет L)	Вычислить M=L*2-(X & 00F0) M>3111 переход к п/п 1 (R=-M) M≤3111 переход к п/п 2 (R=~M) {M - число без знака}	Если R четное, то переход к АДР1 (R+0999), иначе переход к АДР2 (R or 5A5A)
Дано: X=45B2 Y=1C2D Z=203E (расположены в памяти один за другим)		ЗАДАНИЕ № 11
В цикле проинвертировать младшие байты X, Y, Z (результатом будет X', Y', Z')	Вычислить M=Z'*8+(X' & Y') Если есть перенос, то переход к п/п 1 (R=мл.M<=>ст.M), иначе переход к п/п 2 (R=M-9911E)	Если R≤0991, то переход к АДР1 (R or 2381), иначе переход к АДР2 (R xor 0080) {R - число со знаком}
Дано: X=8A64 Y=B3D1 Q=88BD (расположены в памяти один за другим)		ЗАДАНИЕ № 12
В цикле каждое из X, Y, Q умножить на 4 (результат X',Y',Q')	Вычислить M=X'+Y'-Q' M>21 переход к п/п 1 (R=M or 0FF0) M≤21 переход кп/п 2 (R=M+1028) {M - число со знаком}	Если R положительно, то переход к АДР1 (R <3c<), если R отрицательно, то переход к АДР2 (R >2>)

Дано: X=16E6 Y=D45B Z=C9C9 (расположены в памяти один за другим)		ЗАДАНИЕ № 13
В цикле увеличить на три старшие байты X, Y, Z (результат X',Y',Z')	Вычислить M=X'-Y'+Z' Если биты 2, 9, 10, 6 = 0, то переход к п/п 1 (R=(M[8,10]=1)), иначе переход к п/п 2 (R=M*2)	Если R ≠ 4341, то переход к АДР1 (R or ~Z') иначе переход к АДР2 (R + 66) {R - число со знаком}
Дано: X=B8F7 Y=3DA6 Z=911C Q=6633 (числа со знаком расположены в памяти один за другим)		ЗАДАНИЕ № 14
В цикле изменить знак у младших байтов X, Y, Z, Q (результат X', Y', Z', Q')	Вычислить M=(Y' & X')+(Z' or Q') мл.M<4 переход к п/п 1 (R=M+Y') мл.M≥4 переход к п/п 2 (R=M-999-cf)	Если R отрицательное, то переход к АДР1 (R xor X'), Если R положительное, то переход к АДР2 (R xor 1011)
Дано: X=AB7C Y=C58E Z=ABCD (размещены в памяти один за другим)		ЗАДАНИЕ № 15
В цикле обнулить в X, Y, Z биты 3, 5, 6, 10 (результат X',Y',Z')	Вычислить M=(X' & Y')- ~Z' Если у мл.M четное количество единиц, то переход к п/п 1 (R=M(>6>)), иначе переход к п/п 2 (R=M & F1F1)	Если был перенос, то переход к АДР1 (R+1), иначе переход к АДР2 (R or 1021)
Дано: X=FC6A Y=F639 Z=6132		ЗАДАНИЕ № 16
В цикле у X установить в 1 биты 3, 7, 11 у Y сбросить биты 1, 6, 11 (результат X' и Y')	Вычислить M=(X'-Z) & (Y'-Z) Если M отрицательное, то п/п 1 (R=M+123), Если M положительное, то п/п 2 (R=M-999)	Если R≤09, то переход к АДР1 (R(<4c<)), при R>09 переход к АДР2 (R or 1001) {R - число со знаком}
Дано: X=CD5B Y=E742 Z=9138 (расположены в памяти один за другим)		ЗАДАНИЕ № 17
В цикле у X, Y, Z проинвертировать биты 0, 5, 10, 9, 14,15 (результатом будет X', Y', Z')	Вычислить M=X'-Y'+Z' M<19A8 переход к п/п 1 (R=M(<5c<)) M≥19A8 переход к п/п 2 (R=M and 9238) {M - число без знака}	Если R отрицательное, то переход к АДР1 (мл.R<=>ст.R), Если R положительное, то переход к АДР2 (R-1)

Дано: X=1231 Y=5691 Z=A893 Q=15EE (расположены в памяти один за другим)		ЗАДАНИЕ № 18
Сложить проинвертированные данные в цикле (результатом будет L)	Вычислить M=L & Z' M>4220 переход к п/п 1 (R=M/2) M≤4220 переход к п/п 2 (R=~M) {M - число со знаком}	Если в мл. R четное количество единиц, то переход к АДР1 (R xor F0F0), иначе переход к АДР2 (R+1)
Дано: X=9189 Y=714B Z=CD12 (расположены в памяти один за другим)		ЗАДАНИЕ № 19
В цикле произвести обнуление старших байтов данных X,Y,Z (результат X', Y', Z')	Вычислить M=X'+ Y'& Z' мл.M>0 переход к п/п 1 (R=M-4101) мл.M≤0 переход к п/п 2 (R=M xor E130)	Если в мл.R четное количество единиц, то переход к АДР1 (R or 1024), иначе переход к АДР2 (R/2)
Дано: X=381E Y=F120 Z=3F9B Q=3981 (расположены в памяти один за другим)		ЗАДАНИЕ № 20
В цикле из младших байтов X, Y, Z, Q вычесть 43 (результат X', Y', Z', Q')	Вычислить M=X'& Z' or Y' M≠1324 переход к п/п 1 (R=M+061B) M=1324 переход к п/п 2 (R=0028)	Если ст.R≥31, то переход к АДР1 (R-11F1), иначе переход к АДР2 (R xor 1F08) {R - число без знака}

Условные обозначения в таблице:

~ - логическое отрицание;
& - логическое умножение;
or - логическое сложение;
xor - сумма по "модулю 2";
>N> - циклический сдвиг вправо на N-разрядов;
<N< - циклический сдвиг влево на N-разрядов;
>Nc> - циклический сдвиг вправо на N-разрядов с cf;
<Nc< - циклический сдвиг влево на N-разрядов с cf;
<-> - обмен байтами в слове;
мл. - младший байт в слове;
ст. - старший байт в слове;
M[N] - выбирается бит N; { M[N1]=1, M[N1,N2]=0 }
cf - флаг переноса.

3. Содержание отчета:

1. Текст программы с комментариями

2. Верификация программы: результаты расчета заданных выражений, скриншоты, показывающие содержимое регистров и значения переменных после каждого действия программы, входные данные и скриншоты регистров

и переменных в ключевых точках программы для проверки остальных ветвлений.

4. Литература

1 Intel. Номера процессоров Intel. — [Web-документ], н. д. / 2008. — URL: [http://www.intel.com/products/processor_number/rus/index.htm] (12.11.2008).

2 Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture November 2006 Order Number: 253668-022US.

3 Intel386 EX Embedded Microprocessor User's Manual. — [Электронный документ], 1996 Order Number: 272485-002. — На диске: \Microprocessor\Docs\27248502.pdf].

4 Брамм П., Брамм Д. Микропроцессор 80386 и его программирование: пер. с англ. — Москва: Мир, 1990. — 448 с.

5 Википедия. Intel 8086. — [Web-документ], 10 августа 2009. — URL: [http://ru.wikipedia.org/wiki/8086].

6 Паппас К., Марри У. Микропроцессор 80386. Справочник: пер. с англ. / под ред. В. В. Василькова. — Москва: Радио и связь, 1993. — 318 с.

7 Рафикузаман М. Микропроцессоры и машинное проектирование микропроцессорных систем: пер. с англ. В 2 кн. Кн. 2. — М.: Мир, 1988. — 288 с.

8 Смит Б. Э., Джонсон М. Т. Архитектура и программирование микропроцессора Intel 80386: пер. с англ./ Под ред.А. С. Карнаухова. — Москва: Конкорд, 1992. — 334 с.

9 Л. Дао. Программирование микропроцессора 8088: пер. с англ./ Под ред. М. М. Гельмана. — Москва: Мир, 1988. — 357 с.

10 Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, http://www.intel.com/Assets/PDF/manual/253666.pdf (05.12.2010).

11 Лю Ю-Чжен, Гибсон Г. Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем. Пер. с англ. — М.: Радио и связь, 1987. — 512.; илл.

12 Микропроцессорный комплект К1810: Структура, программирование, применение: Справочная книга/ Ю.М. Казаринов, В.Н. Номоконов, Г.С. Подклетнов, В.Ф. Филиппов; Под ред. Ю.М. Казаринова. – М.: Высш. Шк., 1990. – 269 с.

13 К. Касперски. Cod:net. Техника и философия хакерских атак. — [web-документ]. — URL: [http://www.codenet.ru/progr/other/hack_solon8.php] (2.10.2008).

14 Cod:net. Справочник по системе программирования Турбо Ассемблер. — [web-документ]. — URL: [http://www.codenet.ru/progr/asm/tasm/41.php] (28.10.2008).