 Находнов Максим 103 группа

09.02.2017г.

Примеры ассемблерных программ

```
%include "io.inc"

section .text
global CMAIN
CMAIN:
    ; This is a comment
    PRINT_STRING    "Hello, world!"
    NEWLINE
    xor     eax, eax
    ret
```

global CMAIN - позволяет вызывать из других модулей.

конец программы.

```
%include "io.inc"

section .bss
    a   resd    1

section .data
    b   dd      1

section .text
global CMAIN
CMAIN:
    GET_DEC 4, [a]
    GET_DEC 4, [b]
    mov     eax, dword [a]
    add     eax, dword [b]
    PRINT_DEC 4, eax
    NEWLINE
    xor     eax, eax
    ret
```

сколько байтов считать

взятие адреса.

- Сайт курса – http://asmcourse.cs.msu.ru
- Регистры: eax, ebx, ecx, edx, esi, edi, ebp, esp.
- Возвращаемое значение функции – в регистре eax, указатель стека – в регистре esp.
- CMAIN = int main(void)
- Средства ввода: GET_DEC, GET_UDEC, GET_HEX, GET_CHAR, GET_STRING.
- Средства вывода – то же самое, но PRINT_ вместо GET_.
- Размеры обращений к памяти: byte, word, dword.
- Резервирование памяти: resb, resw, resd.
- Объявление переменных с инициализацией: db, dw, dd.

 Находнов Максим, 103 группа, ВМК МГУ

13.02.2017г.

Ассемблерные инструкции

Инструкция	Описание
MOV	пересылка данных
MOVSX	
MOVZX	
ADD	арифметические операции
SUB	
NEG	

Условные переходы

- ZF – результат операции равен 0 (zero flag)
- SF – результат операции отрицателен (sign flag)
- CF – произошло беззнаковое переполнение (carry flag)
- OF – произошло знаковое переполнение (overflow flag)

Инструкция	Флаги	Инструкция	Флаги
JO	OF = 1	JNO	OF = 0
JS	SF = 1	JNS	SF = 0
JE	ZF = 1	JNE	ZF = 0
JZ		JNZ	
JB	CF = 1	JNB	CF = 0
JNAE		JAE	
JC		JNC	
JBE	CF = 1 или ZF = 1	~~JNBE~~ JA	CF = 0 и ZF = 0
~~JBE~~ JNA		JNBE	
JL	SF ≠ OF	JNL	SF = OF
~~JNBA~~ JNGE		JGE	
JLE	ZF = 1 или SF ≠ OF	JNLE	ZF = 0 и SF = OF
JNG		JG	
JECXZ	ECX = 0		

Средства ввода-вывода

PRINT_DEC size, data	GET_DEC size, data
PRINT_UDEC size, data	GET_UDEC size, data
PRINT_HEX size, data	GET_HEX size, data
PRINT_CHAR ch	GET_CHAR sh
PRINT_STRING data	GET_STRING data, maxsize
NEWLINE	

PF - чётность суммы бит (parityflag)

Задачи

1. Пусть в программе объявлены следущие переменные.

```
section .data
        a       db      0x40
                dw      0x81
                db      0xFE
        b       dd      0
```

Чему будет равно занчение переменной b после выполнения следующих команд?

а)
```
mov     eax, dword [a]
mov     dword [b], eax
```

б)
```
movsx   eax, byte [a + 3]
mov     dword [b], eax
```

в)
```
movsx   eax, word [a + 1]
mov     dword [b], eax
```

г)
```
movsx   eax, byte [a + 1]
mov     dword [b], eax
```

д)
```
movzx   eax, byte [a + 1]
mov     dword [b], eax
```

е)
```
xor     eax, eax
movsx   ax, byte [a + 1]
mov     dword [b], eax
```

2. Приведите пример 16-битных значений, при сложении которых:

   а) происходит как знаковое, так и беззнаковое переполнение,

   б) происходит знаковое, но не происходит беззнаковое переполнение,

   в) происходит беззнаковое, но не происходит знаковое переполнение.

3. Вычислите значения флагов после выполнения следующих фрагментов кода.

а)
```
mov  ah, 0ffh
mov  al, 0feh
add  ah, al
```
CF = 1
OF = 0
SF = 1

б)
```
mov  ah, 0ffh
mov  al, 0feh
sub  ah, al
```
CF = OF = 0 = SF = ZF

в)
```
mov  ax, 0ffh
mov  dx, 0feh
add  ax, dx
```
CF = OF = SF = ZF = 0

г)
```
mov  ax, 0feh
mov  dx, 0ffh
sub  ax, dx
```
CF = 1 SF = 1
OF = 0 ZF = 0

 Находнов Максим, 103 группа, ВМК МГУ

16.02.2017г.

Ассемблерные инструкции

КОП	#1	#2	#3	SF	ZF	OF	CF
MOV	r 8/16/32	r/m 8/16/32					
	m 8/16/32	r 8/16/32					
MOVSX	r 16/32	r/m 8/16	~~обращается~~				
MOVZX	r 16/32	r/m 8/16					
ADD	r 8/16/32	r/m/i 8/16/32		M	M	M	M
	m 8/16/32	r/i 8/16/32		M	M	M	M
SUB	r 8/16/32	r/m/i 8/16/32		M	M	M	M
	m 8/16/32	r/i 8/16/32		M	M	M	M
CMP	r 8/16/32	r/m/i 8/16/32		M	M	M	M
	m 8/16/32	r/i 8/16/32		M	M	M	M
NEG	r/m 8/16/32			M	M	M	M
INC	r/m 8/16/32			M	M	M	
DEC	r/m 8/16/32			M	M	M	
IMUL (знаковое умнож.)	r/m 8/16/32	(ведет также как MUL)		–	–	M	M
	r 16/32	r/m 16/32	(но обрезает старшие байты)	–	–	M	M
	r 16/32	r/m 16/32	i 16/32	–	–	M	M
MUL (беззнаковое)	r/m 8/16/32	AX=AL·arg(8)		–	–	M	M
IDIV	r/m 8/16/32	знаковое.		–	–	–	–
DIV	r/m 8/16/32	беззнаковое		–	–	–	–
CBW	AL → AX						
CWD	AX → DX:AX						
CDQ	EAX → EDX:EAX						
JMP	r/m/i 32						
Jcc	i 32						

знаковое расширение.

КОП	#1	#2	#3	SF	ZF	OF	CF
AND	r 8/16/32	r/m/i 8/16/32		M	M	0	0
	m 8/16/32	r/i 8/16/32		M	M	0	0
OR	r 8/16/32	r/m/i 8/16/32		M	M	0	0
	m 8/16/32	r/i 8/16/32		M	M	0	0
XOR	r 8/16/32	r/m/i 8/16/32		M	M	0	0
	m 8/16/32	r/i 8/16/32		M	M	0	0
NOT	r/m 8/16/32			M	M	0	0
SHL	r/m 8/16/32	CL		M	M	M	M
	r/m 8/16/32	i 8		M	M	M	M
SHR (логич. сдвиг вправо)	r/m 8/16/32	CL		M	M	M	M
	r/m 8/16/32	i 8		M	M	M	M
SAL	r/m 8/16/32	CL		M	M	M	M
	r/m 8/16/32	i 8		M	M	M	M
SAR (арифмет. сдвиг)	r/m 8/16/32	CL		M	M	M	M
	r/m 8/16/32	i 8		M	M	M	M
ROL	r/m 8/16/32	CL				M	M
	r/m 8/16/32	i 8				M	M
ROR (циклический сдвиг вправо)	r/m 8/16/32	CL				M	M
	r/m 8/16/32	i 8				M	M
RCL	r/m 8/16/32	CL				M	M
	r/m 8/16/32	i 8				M	M
RCR (длинный цикл. сдвиг)	r/m 8/16/32	CL				M	M
	r/m 8/16/32	i 8				M	M
CMOVcc	r 16/32	r/m 16/32					

передвинуть если условие выполнено. Обращение к памяти происходит даже если условие false

M - модификация флага
- - порча флага.

DX:AX = AX · arg(16)
EDX:EAX = EAX · arg(32)
AX = AL · arg(8)

AL = AX ÷ Arg ; AH = AX % Arg(8)
EAX = (EDX:EAX) ÷ arg ; EDX = (EDX:EAX) % arg(32)
AX = (DX:AX) ÷ Arg ; DX = (DX:AX) ÷ Arg(16)

LOOP THERE (выполняет из ECX 1 и, если ECX не 0, то переходит по метке (на метку))
метка

CWDE AX → EAX (знаково)

Задачи

1. Пусть в программе на Си объявлены следующие переменные.

   ```
   int a[4]; short b[4]; char p[4];
   unsigned short c[4]; unsigned char q[4];
   ```

   Напишите фрагменты кода на ассемблере, эквивалентные следующим выражениям на Си.

   а) `a[0] = b[0] + b[1] - b[2]`

   б) `p[0] = a[0] + c[4]`

   в) `b[0] = p[2] + q[3]`

   г) `a[0] /= a[1]`

   д) `a[0] = a[3] / c[2]`

   е) `a[1] = b[0] * b[1]`

   ж) `a[0] = a[1] % a[2]`

   з) `a[0] = a[0] << a[1]`

   и) `a[0] = p[0] | (p[1] << 8) | (p[2] << 16)`

   к) `a[0] = (a[0] << q[0]) | (a[0] >> (32 - q[0]))`

   л) `a[0] = a[0] < 0 ? 0xffffffff : 0`

   м) `a[0] = b[1] / p[2]`

2. В регистре EAX записано некоторое число. Необходимо вычислить моудль этого числа

   - с использованием условных переходов,
   - с использованием инструкции условной пересылки данных,
   - без использования каких-либо условных инструкции (используйте арифметический сдвиг вправо!).

```
cdq
xor eax,edx
sub eax,edx
```

3. Пусть в регистре ECX записано количество элементов массива, а в EBX записан адрес первого элемента.

   а) Напишите цикл, который считывает элементы массива.

   б) Напишите цикл, который находит максимум в массиве и записывает его в регистр EDX.

4. Напишет фрагмент программы, который проверяет число, записанное в ESI на простоту. Если число простое, то в регистр EAX нужно записать 1, иначе – 0.

```
cmp eax, 0
jns .1
neg eax
.1:
```

```
xor edx, edx
sub edx, eax
cmovg eax, edx.
```

Находков Максим 103 группа
03.03.2017г.

Задачи

Напишите код на ассемблере, эквивалентный приведенным фрагментам программ на Си.

1.
```
int **p;
/* ... */
if (*p) **p = 42;
```

```
section .bss
    p resd 1
section .text
    mov eax, dword [p]
    mov eax, dword [eax]
    cmp eax, 0
    je .out
    mov dword [eax], 42
.out:
```

2.
```
int *p, *q;
/* ... */
*p++ = *++q;
*p++ = --*q
```

3.
```
char *s1, *s2;
/* ... */
do {
        *s1++ = *s2;
} while (*s2++);
```

4.
```
char *s1;
int l;
/* ... */
l = 0;
while (*s1++) l++;
```

5.
```
int *a, n;
/* ... */
for (int i = 0; i < n; i++) {
        a[i] = i + 1;
}
```

6.
```
char *s;
/* ... */
for (i = 0; s[i]; i++) {
        s[i] = s[i] + 'A' - 'a';
}
```

7.
```
int *a, n;
/* ... */
do {
        if (a[n] < 0) break;
        if (a[n] == 0) continue;
        a[n]--;
} while (n-- != 0)
```

8.
```
int *a, n;
/* ... */
while (n) {
        n--;
        if (a[n] <= 0) continue;
        a[n]--;
}
```

9.
```
int *a, n, i;
/* ... */
for (i = n - 1; i >= 0 && a[i] > 0; i--) {
        if (a[i] == 1) continue;
        a[i]--;
}
```

10.03.2017 Находнов Максим 103 группа

Задачи

Напишите код на Си, эквивалентный приведенным фрагментам программ на ассемблере.

1.
```
        mov     eax, dword [a]
        cmp     eax, dword [b]
        cmovb   eax, dword [b]
        mov     dword [x], eax
```

unsigned int a,b,x;
x = a<b ? b : a;

2.
```
        mov     eax, dword [a]
        mov     edx, dword [a + 4]
        add     eax, dword [b]
        adc     edx, dword [b + 4]
        sub     eax, dword [c]
        sbb     edx, dword [c + 4]
        mov     dword [a], eax
        mov     dword [a + 4], edx
```

long long a, b, c;
a += (b-c)

3.
```
        mov     ecx, dword [n]
        inc     ecx
        lea     ebx, [a + 4 * ecx - 4]
        xor     eax, eax
L:
        imul    eax, dword [x]
        add     eax, dword [ebx]
        sub     ebx, 4
        dec     ecx
        jne     L
        mov     dword [f], eax
```

int *b = a+n, f=0, x;
for(int c=n+1; c; --c)
  f=x*f + a[c-1];

4.

```
    cmp     dword [a], 0
    je      L1
    mov     eax, dword [b]
    cdq
    idiv    dword [a]
    jmp     L2
L1:
    mov     eax, 0
L2:
    mov     dword [x], eax
```

```
int a, b, x
if(a) x = b/a;
else x = 0;
```

5.

```
    mov     eax, dword [a]
    imul    dword [b]
    idiv    dword [c]
    mov     dword [x], eax
```

```
int x, a, b, c;
x = (long long) a*b/c;
```

Ассемблерные инструкции

КОП	#1	#2	#3	SF	ZF	OF	CF
XCHG	r/m 8/16/32	r/m 8/16/32	~~нельзя~~				
LEA	r 32	m					
ADC	r/m 8/16/32	r/m/i 8/16/32		M	M	M	M
SBB	r/m 8/16/32	r/m/i 8/16/32		M	M	M	M
CWDE							
CPUID[1]							
POPCNT[1]	r 16/32	r/m 16/32		0	M	0	0
LZCNT[1]	r 16/32	r/m 16/32			M		M

[1] Инструкции CPUID, POPCNT и LZCNT не входят в программу курса.

6.
```
    mov     eax, 1
L:
    xchg    eax, dword [s]
    cmp     eax, 1
    je      L
```

volitale int a, s;
do {
a = s;
s = 1;
} -2 инструкции
{while(a == 1);

1 инструкция

7.
```
    mov     ebx, dword [b]
    test    ebx, ebx
    je      L
    cmp     dword [ebx], 0
    jne     L
    mov     dword [ebx], 1
L:
```

8.
```
    movsx   eax, word [a]
    movsx   ebx, word [b]
    movsx   ecx, word [c]
    cmp     eax, ebx
    jl      L1
    cmp     eax, ecx
    jge     L2
L1:
    add     word [a], 10h
    jmp     L3
L2:
    sub     word [a], 10h
L3:
```

short a,b,c;
if((a<b)||(a<c))
a += 16;
}
else a -= 16;

9.

L0

L1

L2:

L3:

```
mov     eax, dword [a]
mov     ebx, dword [b]
mov     ecx, dword [c]
cmp     eax, ebx
jg      L0
cmp     ebx, ecx
jle     L1

cmp     eax, ebx
jl      L2
cmp     ebx, ecx
jl      L2

mov     dword [res], 1
jmp     L3

mov     dword [res], 0
```

```
int a, b, c, res;

res = !((a>b||b>c)&&
        (a<b||b<c));
```

 17.03.2017г

Нахаков Максим 103 группа.

```
%include "io.inc"

section .text
global CMAIN
plus:
    push    ebp
    mov     ebp, esp
    mov     eax, dword [ebp + 8]
    add     eax, dword [ebp + 12]
    mov     esp, ebp
    pop     ebp
    ret

CMAIN:
    push    ebp
    mov     ebp, esp
    sub     esp, 8
    GET_DEC 4, [esp]
    GET_DEC 4, [esp + 4]
    call    plus
    PRINT_DEC   4, eax
    leave
    ret
```

- EAX, ECX, EDX – можно изменять в функции,
- EBX, ESI, EDI, EBP – нужно сохранять значения,
- ESP – служебный регистр.

```
%include "io.inc"

section .data
    msg     db  'Hello, world!\n', 0

section .text
global CMAIN
CMAIN:
    push    ebp
    mov     ebp, esp
    sub     esp, 8
    mov     dword [esp], msg
    call    printf
    mov     esp, ebp
    pop     ebp
    ret
```

Ассемблерные инструкции

КОП	#1	#2	#3	SF	ZF	OF	CF
CALL	r/m/i 32						
RET							
PUSH	r/m 32						
POP	r/m 32						
LEAVE							

Задачи

1. Напишите функцию, которая ищет максимум в массиве.
2. Напишите функцию, которая ищет среднее арифметическое трех чисел.
3. Напишите функцию, которая сравнивает две строки.
4. Напишите функцию, которая обменивает местами свои аргументы, переданные по указателю.
5. Напишите функцию, которая получает на вход два массива одинаковой длины `a` и `b` и указатель на функцию `f`. Необходимо вычислить значение массива `b` по формуле `b[i]=f(a[i])`.
6. Напишите эквивалентную функцию на языке Си.

```
foo:
        push    ebp
        mov     ebp, esp
        mov     eax, dword [ebp + 8]
        mul     dword [ebp + 12]
        leave
        ret
```

7. Напишите эквивалентную функцию на языке ассемблера.

```
int foo(int a) {
    int t;
    if (bar(a, &t))
        return t;
    return 0;
}
```

8. Напишите эквивалентную функцию на языке Си.

```
foo:
        push    ebp
        mov     ebp, esp
        push    dword [ebp + 12]
        push    dword [ebp + 8]
        call    [ebp + 16]
        leave
        ret
```

9. Напишите эквивалентную функцию на языке ассемблера.

```
long long foo(short a, char b) {
    return a + b;
}
```

10. Напишите эквивалентную функцию на языке Си, если известно, что возвращаемое значение имеет тип `unsigned int`.

```
foo:
        push    ebp
        mov     ebp, esp
        push    ebx
        mov     ebx, dword [ebp + 12]
        mov     eax, dword [ebp + 8]
        mov     ebx, dword [ebx + 4 * eax]
        sub     esp, 4
        mov     dword [esp], 0
        call    ebx
        add     esp, 4
        pop     ebx
        leave
        ret
```

 Находнов Максим
24.03.2017 группа 103

Задачи

1. Напишите функцию, которая рекурсивно вычисляет число сочетаний используя формулу

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}.$$

2. Напишите эквивалентные функции на языке ассемблера.

```
int g(int n) { return f(n, 1); }
int f(int n, int res) {
    return n ? f(n - 1, res * n) : 1;
}
```

3. Пусть структура `struct point {int x, int y}` описывает точку на плоскости. Напишите функцию, которая подсчитывает в переданном ей массиве точек, сколько из них принадлежат прямой $y = \frac{x}{2}$.

4. Напишите функцию `struct point nearest(struct point, struct point)`, которая возвращает ту из двух переданных структур, которая находится ближе к началу координат.

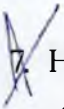
5. Напишите эквивалентную функцию на языке ассемблера.

```
int foo(int a) {
    int t;
    if (bar(a, &t))
        return t;
    return 0;
}
```

6. Напишите эквивалентную функцию на языке ассемблера.

```
struct numbers {
    char size;
    char is_signed;
    char is_last;
    union {
        signed char sc;
        unsigned char uc;
        signed short sh;
        unsigned short uh;
    } u;
};

int sum(struct numbers *a)
{
    int res = 0;
    do {
        switch (a->is_signed * 2 + a->size - 1) {
            case 0: res += a->u.uc; break;
            case 1: res += a->u.uh; break;
            case 2: res += a->u.sc; break;
            case 3: res += a->u.sh; break;
        }
        a++;
    } while (!a[-1].is_last);
    return res;
}
```



Напишите эквивалентные функции на языке Си.

```
g:
        push    ebp
        mov     ebp, esp
        mov     eax, dword [ebp + 8]
        imul    eax, eax
        mov     dword [ebp + 8], eax
        shr     dword [ebp + 12], 1
        leave
        jmp     f

f:
        push    ebp
        mov     ebp, esp
        mov     edx, dword [ebp + 8]
        mov     eax, dword [ebp + 12]
        and     eax, 1
        jz      h
        imul    edx, dword [ebp + 16]
        mov     dword [ebp + 16], edx

h:
        sub     dword [ebp + 12], eax
        mov     eax, dword [ebp + 16]
        leave
        jnz     g
        ret
```

Вызов функции выглядит так.

```
CMAIN:
    ; ....
    sub       esp, 24
    GET_DEC 4, [esp]
    GET_DEC 4, [esp + 4]
    mov       dword [esp + 8], 1
    call      f
    ; ....
```

 Находнов Максим 103 группа
31.05
2017г.

```
%include "io.inc"

section .rodata
    msg     db  'Hello, world!\n', 0

section .text
global CMAIN
CEXTERN printf
CMAIN:
    push    ebp
    mov     ebp, esp
    and     esp, ~15
    sub     esp, 16
    mov     dword [esp], msg
    call    printf
    xor     eax, eax
    leave
    ret
```

Справочный материал

```
size_t fread(void *ptr, size_t size, size_t nmemb,
             FILE *stream);
size_t fwrite(const void *ptr, size_t size,
              size_t nmemb, FILE *stream);
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
char *fgets(char *s, int size, FILE *stream);
```

Задачи

1. Напишите программу, которая считывает два числа числа и выводит их сумму.
2. Напишите программу, которая считывает из выходного файла `input.txt` последовательность 32-битных знаковых чисел и выводит их сумму в выходной файл `output.txt`. Обратите внимание, что сумма нескольких 32-битных чисел уже не обязательно 32-битная.
3. Напишите программу, которая считывает два 32-битных числа из бинарного файла `input.bin`, и выводит их сумму на экран. Можете считать, что сумма введенных чисел также 32-битная.
4. Напишите функцию `int count(FILE *f, int *a, int ch)`, которая считает, сколько раз в каждой непустой строке текстового файла `f` встречается символ `ch` и записывает это в соответствующий элемент массива `a`. Функция долна возвращать количество непустых строк в файле. Длины строк не превосходят 256 байт.
5. В файле `data.bin` записана последовательность структур `struct data { short tag; int value; }`. Напишите программу, которая вычисляет сумму полей `value` всех структур, значение поля `tag` которых равно -273.
6. Пусть `struct string { int l; char s[]; }` описывает строку, где `l` – длина строки, а `s` – сама строка. Напишите функцию `struct string *readstring(FILE *f)`, которая читает из бинарного файла данную структуру. Память под результирующую структуру должна выделяться динамически.
7. Напишите фрагмент программы, который сортирует по неубыванию массив целых чисел с помощью библиотечной функции qsort. Функцию сравнения также нужно написать.

Другие соглашения о вызовах

- fastcall (два аргумента на регистрах ECX и EDX)

```
__attribute__((fastcall))
int f(int x, int y, int z)
{
    return x + y + z;
}
```

```
f:
    lea     eax, [ecx + edx]
    add     eax, dword [esp + 4]
    ret     4
```

- fastcall (один аргумент на регистре ECX)

```
__attribute__((fastcall))
int f(int x, long long y, int z)
{
    return x + y + z;
}
```

```
f:
    mov     eax, ecx
    add     eax, dword [esp + 4]
    add     eax, dword [esp + 12]
    ret     12
```

Ассемблерные инструкции

КОП	#1	#2	#3	SF	ZF	OF	CF
RET	i16						

- fastcall (все аргументы на стеке)

```
__attribute__((fastcall))
int f(long long x, int y, int z)
{
    return x + y + z;
}
```

```
f:
    mov     eax, dword [esp + 4]
    add     eax, dword [esp + 12]
    add     eax, dword [esp + 16]
    ret     16
```

- stdcall

```
__attribute__((stdcall))
int f(int x, int y, int z)
{
    return x + y + z;
}
```

```
f:
    mov     eax, dword [esp + 4]
    add     eax, dword [esp + 8]
    add     eax, dword [esp + 12]
    ret     12
```

Находнов Максим, 103 группа, ВМК МГУ
07.04.2017г.

calling convention	cdecl	stdcall	fastcall
parameters in registers			ECX, EDX
registers for return	EAX, EDX		
stack cleanup by	caller	callee	callee
caller-save registers	EAX, ECX, EDX		
callee-save registers	EBX, ESI, EDI		

 Находнов Максим

103. группа 14.042017г

Числа с плавающей точкой

Тип	S	E	M	Размер
float	1	8	23	32
double	1	11	52	64

Пусть s, e и m – значения полей знака, экспоненты и мантиссы соответственно, E и M – размеры полей экспоненты и мантиссы в битах соответственно. Обозначим смещение экспоненты через $off = 2^{E-1} - 1$.

- Нормализованные числа: $(-1)^s \times 2^{e-off} \times (1 + m \times 2^{-M})$.
- Денормализованные числа: $(-1)^s \times 2^{1-off} \times m \times 2^{-M}$. $= (-1)^S \times 2^{-off} \times m$
- Специальные значения: $+\infty$, $-\infty$, NaN.

Задача

Пусть $E = 3$, $M = 4$. Выпишите в двоичном виде следующие числа:

а) $1\frac{1}{4}$

б) $-\frac{3}{8}$

в) $-\frac{2}{3}$

г) Минимальное положительное нормализованное число.

д) Минимальное нормализованное число.

е) Максимальное денормализованное число.

ж) Минимальное положительное денормализованное число.

з) $-\infty$

КОП	#1	#2	Примечания
FINIT			инициализация
FFREE	STi		
FLD	m 32/64/80		загрузка на стек
FLD	STi		выгрузка со стека
FST	m 32/64/80		
FST	STi		
FSTP	m 32/64/80		-//- + убирает со стека
FSTP	STi		
FLD1			загрузка на стек: 1
FLDZ			0
FLDL2E			$\ln e$
FLDL2T			$\ln 10$
FLDLG2			$\log_{10} 2$
FLDLN2			$\ln 2$
FLDPI			π
FILD	m 16/32/64		загрузка целого числа
FIST	m 16/32/64		double → int
FISTP	m 16/32/64		
FADD	ST0	STi	
	STi	ST0	
FADDP	STi	ST0	
FSUB	ST0	STi	
	STi	ST0	
FSUBP	STi	ST0	
FMUL	ST0	STi	
	STi	ST0	
FMULP	STi	ST0	
FDIV	ST0	STi	
	STi	ST0	
FDIVP	STi	ST0	
FADDP			берёт 2 значения со стека, сни-мает их со стека, результат кладёт на стек
FSUBP	(STi − ST0) → ST0'		
FMULP			
FDIVP	(STi \ ST0) → ST0'		

Задачи

1. Напишите программу, которая решает линейное уравнение $ax = b$, где $a \neq 0$.
2. Напишите программу, которая находит площадь треугольника, заданного координатами вершин.
3. Напишите программу, которая скалярно умножает два n-мерных вектора.
4. Напишите программу, которая считывает из файла `data.bin` числа двойной точности и находит их среднее геометрическое.
5. Напишите функция `double avg(int n, ...)`, которая вычисляет среднее арифметическое `n` чисел с плавающей точкой.
6. Напишите программу, которая решает уравнение $ax^2+bx+c = 0$.
7. Напишите функцию `double pow(double x, double y)`, которая вычисляет x^y.

КОП	#1	#2	ZF	SF	CF	Примечания
FUCOMI	ST0	STi	M	M	M	флаги ставятся как будто сравнивались беззнаковые числа
FUCOMIP	ST0	STi	M	M	M	
FSQRT						
FSIN						$\texttt{ST0} \in [-2^{63}, +2^{63}]$
FCOS						$\texttt{ST0} \in [-2^{63}, +2^{63}]$
FABS						
FPTAN						Кладет 1.0 на стек
FPATAN						$\operatorname{arctg} \frac{\texttt{ST1}}{\texttt{ST0}}$ (ну, почти)
FSCALE						$\texttt{ST0} \times 2^{\texttt{ST1}}$, ST1 округлено к
FYL2X						$\texttt{ST1} \times \log_2 \texttt{ST0}$
F2XM1						$2^{\texttt{ST0}} - 1, \texttt{ST0} \in [-1, 1]$
FXCH						ST0 ⇄ STi

 Находнов Максим 103 группа

21.04.2017г.

Задача 1

main.c:

```
void hello(void);

const char *msg = "Hello!";

int main(void) {
    hello();
    return 0;
}
```

hello.c:

```
#include <stdio.h>

extern const char *msg;

void hello(void) {
    printf("%s\n", msg);
}
```

В ходе компоновки

- секция `.data` оказалась по адресу 0x80496f4,
- секция `.rodata` по адресу 0x80484c8,
- секция `.plt` по адресу 0x80482c0,
- секция `.text` по адресу 0x8048300.

Символ `msg` имеет смещение 0x8 от начала секции, символ `hello` имеет смещение 0x11f, а символ `puts@plt` имеет смещение 0x10.

Определите значения всех релокаций.

objdump -dr -M intel main.o:

```
00000000 <main>:
   0:    8d 4c 24 04          lea    ecx, [esp+0x4]
   4:    83 e4 f0             and    esp, 0xfffffff0
   7:    ff 71 fc             push   DWORD PTR [ecx-0x4]
   a:    55                   push   ebp
   b:    89 e5                mov    ebp, esp
   d:    51                   push   ecx
   e:    83 ec 04             sub    esp, 0x4
  11:    e8 fc ff ff ff       call   12 <main+0x12>
             12: R_386_PC32   hello

  16:    b8 00 00 00 00       mov    eax, 0x0
  1b:    83 c4 04             add    esp, 0x4
  1e:    59                   pop    ecx
  1f:    5d                   pop    ebp
  20:    8d 61 fc             lea    esp, [ecx-0x4]
  23:    c3                   ret
```

objdump -dr -M intel hello.o:

```
00000000 <hello>:
   0:    83 ec 18             sub    esp, 0x18
   3:    ff 35 00 00 00 00    push   DWORD PTR ds:0x0
             5: R_386_32      msg

   9:    e8 fc ff ff ff       call   a <hello+0xa>
             a: R_386_PC32    puts

   e:    83 c4 1c             add    esp, 0x1c
  11:    c3                   ret
```

Задача 2

Та же самая программа, но скомпилированная с флагом `-fPIC`.
Адреса символов:

- `hello: 0x8048426`
- `main: 0x804840a`
- `__x86.get_pc_thunk.bx: 0x8048426`
- `_GLOBAL_OFFSET_TABLE_: 0x80496fc`
- `msg: 0x804971c`
- `puts@plt: 0x80482d0`

Вычислите значения всех релокаций.

objdump -d -M intel hello.o:

```
Disassembly of section .text:

00000000 <hello>:
   0:   53                      push   ebx
   1:   83 ec 14                sub    esp, 0x14
   4:   e8 fc ff ff ff          call   5 <hello+0x5>

                  ____________________________________

   9:   81 c3 02 00 00 00       add    ebx, 0x2

                  ____________________________________

   f:   8b 83 00 00 00 00       mov    eax,DWORD PTR [ebx+0x0]

                  ____________________________________

  15:   ff 30                   push   DWORD PTR [eax]
  17:   e8 fc ff ff ff          call   18 <hello+0x18>

                  ____________________________________

  1c:   83 c4 18                add    esp, 0x18
  1f:   5b                      pop    ebx
  20:   c3                      ret
```

readelf -r hello.o:

```
Relocation section '.rel.text' at offset 0x280
                                        contains 4 entries:
 Offset   Type            Sym.Value  Sym. Name
00000005 R_386_PC32      00000000   __x86.get_pc_thunk.bx
0000000b R_386_GOTPC     00000000   _GLOBAL_OFFSET_TABLE_
00000011 R_386_GOT32     00000000   msg
00000018 R_386_PLT32     00000000   puts
```

(часть столбцов была опущена для краткости)

 Находнов Максим 103 группа
21.04
2017г.

Компоновка (исполняемый файл: тип: ELF 12)

Адресация

- Абсолютная
- Относительная (относительно текущего значения EIP)

Статическая компоновка

Ссылка	Формула
R_386_32	S+A
R_386_PC32	S+A-P

- абсолютная адресация.
- относительная

Динамическая компоновка

Ссылка	Формула
R_386_GOT32	G+A
R_386_PLT32	L+A-P
R_386_GOTOFF	S+A-GOT
R_386_GOTPC	GOT+A-P

.so - shared object

Обозначения

S	Адрес символа
A	Слагаемое, записанное в ссылке
P	Адрес ссылки
GOT	Адрес Global Offset Table (это не обязательно начало GOT)
G	Смещение в Global Offset Table
L	Адрес элемента Procedure Linkage Table

Makefile

- `Makefile` или `makefile`.
- `# comment`

  Комментарий.
- VAR=`value`

  Определение переменной.
- ```
  V=1
  VAR=2
  RES=$VAR ${VAR} $(VAR)
  # RES=1AR 2 2
  ```

  Использование переменной.
- ```
  target: dependencies
      actions
  ```

  Правила.
- `all` и `clean` – стандартные имена правил для сборки проекта и очистки директории от промежуточных файлов.
- `$@` – имя цели правила.
- `$<` – имя первой зависимости.
- `$^` – имена всех зависимостей через пробел.

Makefile:

```
CC=gcc
NASM=nasm
CFLAGS+=-W -Wall -g -O2 -m32
ASMFLAGS+=-g -f elf32
.PHONY: all clean - определяет all и clean не как
                    файлы, а как директивы.

all: hello

hello: main.o hello.o
        $(CC) $(CFLAGS) -o $@ $^

main.o: main.c
        $(CC) $(CFLAGS) -c $<

hello.o: hello.asm
        $(NASM) $(ASMFLAGS) $< -o $@

clean:
        rm -rf *.o
```

hello.asm:

```
section .rodata
    msg db  'My arguments:\n', 0
section .text
extern printf
global hello
hello:
    sub     esp, 16
    mov     dword [esp], msg
    call    printf
    add     esp, 16
    ret
```

main.c:

```
void hello(void);

int main(int argc, char *argv[]) {
    hello();
    for (int i = 0; i < argc; i++)
        printf("%s\n", argv[i]);
    return 0;
}
```

Задачи

1. Напишите программу, состоящую из двух модулей. Модуль на ассемблере занимается выводом сообщения. Модуль на Си содержит функцию `main`. Программа должна выводить «yes» или «no» в зависимости от

   а) наличия или отсутствия ключа компиляции `-DNO`,

   б) наличия или отсутствия ключа командной строки `-no`.

Компиляция программы

- `nasm -g -f elf32 file1.asm -o file1.o`
  - На ОС Windows вместо `elf32` нужно писать `win32`.
- `gcc -DSIZE=100 -g file2.c -c -o file2.o -m32`
- `gcc file1.o file2.o -o file -m32`

 Находнов Максим 103 группа
05.05.2017г.

Работа виртуальной памяти

1. Память модельного компьютера состоит из 256 адресуемых ячеек размером 1 байт. Выполняется страничная трансляция адресов при обращении к физической памяти. Размер страницы – 16 байт. Транслированные адреса сохраняются в TLB, представляющем собой 2-канальный множественно-ассоциативный кэш. Обращение к физической памяти предваряется проверкой кэша данных, имеющего следующее устройство: кэш прямого отображения, 4 байта в строке, 8 наборов. Описать, что произойдет при чтении по виртуальному линейному адресу `0xcd`.

$0xcd_{16} = 11001101_{2} \Rightarrow$
tag N° offset

$\Rightarrow$ 000 | 11 | 01$_{2}$ – физич. адрес.
tag N° offset

Таблица страниц

VPN	PPN	p
0	–	0
1	f	1
2	2	1
3	–	0

TLB

Набор	tag	v	PPN	p	tag	v	PPN	p
0	2	0	–	0	6	1	0	1
1	0	1	f	1	3	0	–	0

Кэш данных

Набор	tag	v
0	7	1
1	7	0
2	1	0
3	0	1
4	2	1
5	3	1
6	0	0
7	0	0

2. Память модельного компьютера состоит из 512 адресуемых ячеек размером 1 байт. Выполняется страничная трансляция адресов при обращении к физической памяти. Размер страницы – 32 байта. Транслированные адреса сохраняются в TLB, представляющем собой полностью ассоциативный кэш. Обращение к физической памяти предваряется проверкой кэша данных, имеющего следующее устройство: 4-х канальный множественно ассоциативный кэш, 8 байт в строке, 2 набора. Описать, что произойдет при чтении по виртуальному линейному адресу `0x5d`.

$0x5d_{16} = \underbrace{00}_{tag}\underbrace{1011101}_{offset}{}_2$

Таблица страниц

VPN	PPN	p
0	–	0
1	b	1
2	5	1
3	–	0

В TLB – промах;

$\Rightarrow$ PPN = $5_{10} = 0101_2 \Rightarrow$

TLB

tag	v	PPN	p
f	1	0	1
2	0	–	0
5	1	–	0

$\Rightarrow$ $\underbrace{01011}_{tag}\underbrace{101}_{N^o offset}{}_2$

$b_{16} \Rightarrow$ в кэше – промах.

Кэш данных

Набор	tag	v
0	a	1
	3	0
	2	1
	d	0
1	d	1
	5	1
	a	1
	9	1

3. Модельный менеджер памяти управляет кучей из 24 четырехбайтных машинных слов. Для отслеживания свободных блоков используется неявный список. Начальное состояние кучи показано на рисунке ниже. Белый блок свободен, занятые блоки закрашены, неиспользуемая из-за выраавнивания память память перечеркнута. Начальный и последний блоки – служебные, для пользователя они недоступны. Байтовый размер блока хранится в заголовке и граничном теге. В выделенных блоках, за исключением начального блока, граничный тег не используется. Предоставляемая пользователю память выравнивается по 8 байт. Поиск свободного блока начинается с текущей позиции в списке, выбирается первый подходящий. При расщеплении используется вторая часть блока, первая часть – текущая позиция. Слияние проводится незамедлительно.

Рассчитайте пиковое использование памяти и запишите значения тегов после выполнения следующих обращений к менеджеру памяти.

```
p1 = malloc(42);
p2 = malloc(7);
free(p1);
p3 = malloc(36);
p4 = malloc(8);
free(p2);
```