Лабораторная работа 3.
Модули и функции на ассемблере

Цель работы: изучить процесс компиляции программы на C++; научиться включать в проекты на языке C++ ассемблерные модули; научиться описывать функции и вызывать из программы на языке C++.

Л3.1. Задание на лабораторную работу

Задание 1. Разработайте ассемблерную функцию, вычисляющую целое выражение от целого аргумента (в соответствии с вариантом), а также головную программу на языке C++, использующую разработанную функцию.

$y(x) = x + 1$

```
extern "C" int foo(int x);

__asm__
(
    "foo:\n"
    "   movl 8(%esp), %eax\n"
    "   imul $3,%eax\n"
    "   add $1,%eax\n"
    "   ret\n"
);

int main(void)
{
    int x = 1;
    int y = foo(x);
}
```

Задание 2. Разработайте программу, целиком написанную на ассемблере, вычисляющую значение $y(x)$ для $x = 13$ и выводящую полученное значение на стандартный вывод с использованием библиотеки stdlib (в частности, функции printf).

```
.data:
printf_format:
        .string "%d\n"

.globl main
main:
        movl $13, %eax
        imul $3,%eax
        add $1,%eax
        pushl %eax
        pushl $printf_format
        call  printf
        addl  $8, %esp
        movl  $0, %eax
        ret
```

Задание 3. Опишите функцию на произвольном языке высокого уровня (включая C/C++) и вызовите её из ассемблерной функции.

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

using namespace std;

extern "C" int fooASM();
extern "C" int foo();
__asm__
(
   "fooASM:\n"
   "  call foo\n"
   "  ret\n"
);
int foo()
{
   int minX, maxX;
   printf("Enter range(upper and lower bounds): ");
   scanf("%d", &minX);
   scanf("%d", &maxX);
   int x = rand()%(maxX-minX)+minX;
   return x;
}
int main(void)
{
   srand(time(NULL));
   cout << fooASM();
}
```

Задание 4. Бонус (+2 балла). Опишите на ассемблере одну подпрограмму с параметрами $a, b, \ldots$ и результатами x и y и вызовите её из другой ассемблерной программы.

$$\begin{cases} x = a + c * b \\ y = a - c * b \end{cases}$$

```
.data
printf_format:
    .string "%d\n"
a:
    .int 1
b:
    .int 2
c:
    .int 3
x :
    .int 0
y:
    .int 0

calc:

    movl a, %eax
    movl b, %ebx
    movl c, %ecx
    imul %ecx, %ebx
    add %ebx, %eax
    movl %eax, x

    movl a, %eax
    movl b, %ebx
    imul %ecx, %ebx
    sub %ebx, %eax
```

```
        movl %eax, y

        ret
.globl main
main:

        call  calc

        movl x, %eax
        pushl %eax
        pushl $printf_format
        call  printf

        addl  $8, %esp
        movl  $0, %eax

        movl y, %eax
        pushl %eax
        pushl $printf_format
        call  printf

        addl  $8, %esp
        movl  $0, %eax

        ret
```

Вопросы:

1. Какие вы знаете соглашения о вызове?

Тридцатидвухбитные соглашения о вызовах

Таблица 3.1

Соглашение	Параметры в регистрах	Порядок	Очистка стека
cdecl		C	вызывающая программа
pascal		Pascal	функция
winapi (stdcall)		C	функция
gnu		C	this — функция, остальные — вызывающая программа
gnu fastcall	*ecx, edx*	C	функция
gnu regparm (3)	*eax, edx, ecx*	C	функция
Borland fastcall	*ecx, edx*	Pascal	функция
Microsoft fastcall	*ecx, edx*	C	функция

2. Какая команда передаёт управление подпрограмме?

`call`

3. Какая команда возвращает управление вызывающей программе?

`ret`

4. Что такое адрес возврата?

Пусть следующая команда, расположенная по адресу $\mathbf{c}_i$ — `call f.`

Команда `call` помещает в стек адрес следующей по порядку команды c_{i+1} — адрес возврата

5. Какие вы знаете регистры общего назначения?

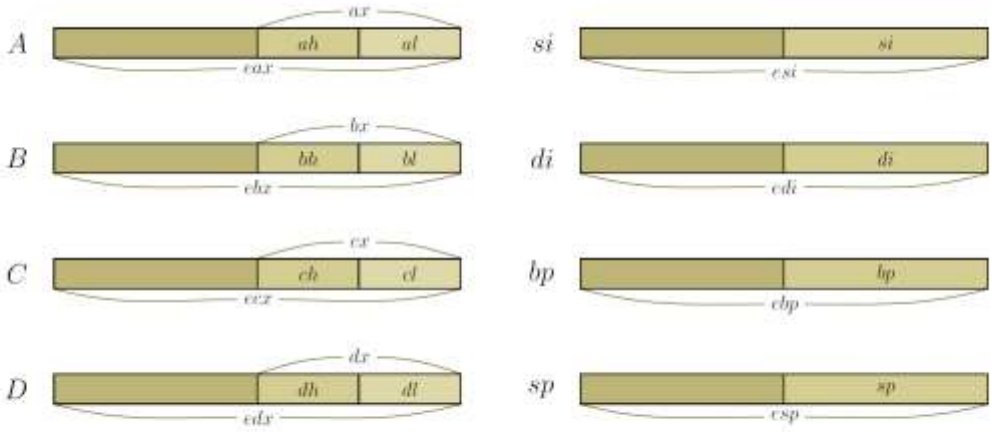

Рис. 1.13. Регистры общего назначения

в тридцатидвухбитном режиме

6. Какие вы знаете команды ассемблера x86?

Основные общие команды

Таблица 4.2

Команда	Действие
`nop` `nop srm`	Ничего не делает *(no operation)*
`mov src, dest`	Присваивание *src* в *dest* *(move)*
`lea smem, dreg`	Вычисление адрес *smem* и записывает его в *dreg* *(load effective address)*
Работа со стеком	
`push src`	Помещение *src* в стек (уменьшает указатель стека)
`pop dest`	Выталкивание значение из стека в *dest* (увеличивает указатель стека)
Вызов и возврат из функций	
`call proc`	Вызов подпрограммы — помещает в стек адрес следующей инструкции (адрес возврата) и переходит по адресу *proc*
`ret [imm]`	Возврат из подпрограммы — снимает со стека адрес возврата и помещает его в указатель команд. Если указан параметр *imm*, снимает со стека ещё *imm* байтов.

Команды целочисленной арифметики

Таблица 4.3

Команда	Действие
Сложение и вычитание	
`inc dest`	Инкремент $++dest$ ($dest = dest + 1$, выполняется быстрее `add`)
`dec dest`	Декремент $--dest$ ($dest = dest - 1$, выполняется быстрее `sub`)
`add src, dest`	Сложение $dest \mathrel{+}= src$ $(dest = dest + src)$
`adc src, dest`	Сложение с переносом из предыдущей части $dest \mathrel{+}= src + CF$ $\Big(dest = dest + (src + CF)\Big)$
`sub src, dest`	Вычитание $dest \mathrel{-}= src$ $(dest = dest - src)$
`cmp src, dest`	Вычитание $dest - src$ без изменения *dest* (только флаги)
`sbb src, dest`	Вычитание с переносом из предыдущей части $dest \mathrel{-}= src + CF$ $\Big(dest = dest - (src + CF)\Big)$
`neg dest`	Изменение знака $dest = -dest$
Расчёт линейной комбинации	
`lea` δ`(r1,r2,`γ`), dreg`	Вычисление эффективного адреса часто используется для расчёта линейной комбинации $dreg = r1 + \gamma \cdot r2 + \delta$ $dreg, r1, r2$ — регистры, γ — 1, 2, 4 или 8, δ — константа `lea` не изменяет флагов

Команды расширения (увеличения разрядности)

Таблица 4.5

Команда	Действие
`movz srm, dreg`	$dreg = srm$ с беззнаковым расширением (размер srm меньше $dreg$)
`movs srm, dreg`	$dreg = srm$ со знаковым расширением (размер srm меньше $dreg$)
`cStD`	Знаковое расширение регистра A (таблица 4.6)

Основные битовые операции

Таблица 4.7

Команда	Действие
	Поразрядные операции
`not dest`	Побитовая инверсия $dest = \sim dest$
`and src, dest`	Побитовое «и» $dest \,\&= src$ $(dest = dest \;\&\; src)$
`test src, dest`	Побитовое «и» $dest \;\&\; src$ без изменения $dest$ (только флаги)
`or src, dest`	Побитовое «или» $dest \mid= src$ $(dest = dest \mid src)$
`xor src, dest`	Побитовое «исключающее или» $dest \hat{}= src$ $(dest = dest \hat{} src)$
	Битовые сдвиги
`shr times, dest`	Беззнаковый (логический) сдвиг вправо $dest = (unsigned)dest >> times$ Освободившиеся старшие разряды заполняются нулями, младшие теряются, кроме последнего, который попадает в CF
`sar times, dest`	Знаковый (арифметический) сдвиг вправо $dest = (signed)dest >> times$ Освободившиеся старшие разряды заполняются знаковым битом, младшие теряются, кроме последнего, который попадает в CF
`shl times, dest` `sal times, dest`	Сдвиг влево (`shl` и `sal` — синонимы) $dest = dest << times$ Освободившиеся младшие разряды заполняются нулями, старшие теряются, кроме последнего, который попадает в CF
`ror times, dest`	Циклический сдвиг $dest$ вправо
`rol times, dest`	Циклический сдвиг $dest$ влево
`rcr times, dest`	Циклический сдвиг через флаг переноса $dest \cup CF$ вправо
`rcl times, dest`	Циклический сдвиг через флаг переноса $dest \cup CF$ влево
	Загрузка idx-го бита числа во флаг CF
`bt idx, dest`	$CF = dest[idx]$
`btc idx, dest`	$CF = dest[idx]$ с последующей инверсией бита $dest[idx] = \neg dest[idx]$
`btr idx, dest`	$CF = dest[idx]$ с последующим сбросом бита $dest[idx] = 0$
`bts idx, dest`	$CF = dest[idx]$ с последующей установкой бита $dest[idx] = 1$

7. Какие вы знаете флаги?

Флаги состояния

Флаги состояния отображают результаты целочисленных арифметических операций (сложения, вычитания, умножения, поразрядных логических операций и пр.); этими флагами являются биты 0, 2, 4, 6, 7 и 11 регистра *flags*.

CF (бит 0)

Флаг переноса (Carry Flag = CF). Устанавливается, если происходит перенос из старшего разряда результата за пределы разрядной сетки при сложении или заём в старший разряд из несуществующего (выходящего за пределы операнда, воображаемого) разряда при вычитании, таким образом, этот флаг показывает переполнение при выполнении беззнаковых арифметических операций.

Флаг CF часто используется и для других целей, тогда его значение не связано с беззнаковым переполнением. Так, этот бит используется командами сдвига — именно в него выдвигается «лишний» бит, командами извлечения бита — для хранения извлечённого значения и многими другими.

PF (бит 2)

Флаг чётности (Parity Flag = PF). Устанавливается, если младший байт результата команды содержит чётное число единиц, иначе — сбрасывается.

Флаг чётности использовался для подсчёта контрольных сумм.

AF (бит 4)

Флаг коррекции (Adjust Flag = AF). Устанавливается, если арифметическая операция производит перенос (заём) из младшей тетрады младшего байта, т. е. из бита 3 в старшую тетраду при сложении (вычитании). Используется только для двоично-десятичной (BCD — Binary-Coded Decimal) арифметики, которая оперирует исключительно младшими байтами.

ZF (бит 6)

Флаг нуля (Zero Flag = ZF). Устанавливается, если результат операции нуль, иначе сбрасывается.

SF (бит 7)

Флаг знака (Sign Flag = SF). Всегда равен значению старшего бита результата. Этот бит интерпретируется как знаковый в некоторых арифметических операциях (0/1 число положительное/отрицательное).

OF (бит 11)

Флаг переполнения (Overflow Flag = OF). Устанавливается, если при знаковой интерпретации результат операции не помещается в операнд (слишком большое положительное или слишком маленькое для отрицательных знаковых чисел); иначе сбрасывается. При сложении этот флаг устанавливается в 1, если происходит перенос в старший бит и нет переноса из старшего бита (то есть сумма положительных чисел даёт результат, интерпретируемый как отрицательный), или имеется перенос из старшего бита, но отсутствует перенос в него (сумма отрицательных чисел положительна); в противном случае, флаг OF устанавливается в 0. При вычитании он устанавливается в 1, когда возникает заём из старшего бита, но заём в старший бит отсутствует, либо имеется заём в старший бит, но отсутствует заём из него.

Флаг переполнения сигнализирует о потере старшего бита результата в связи с переполнением разрядной сетки при работе со знаковыми числами.

Этот флаг используется командами знаковой целочисленной арифметики.

Флаги состояния используются командами целочисленной арифметики трёх типов знаковой, беззнаковой и BCD, а также командами условного перехода (ветвления) и условного присваивания. При выполнении арифметических операций устанавливаются все три набора флагов.

При переполнении индикатором является:

для знаковой арифметики флаг OF,

для беззнаковой арифметики флаг CF,

для BCD-арифметики флаг AF.