Ассемблер. Введение.

Иван Викторович Михайлов

ИТМО, КТ

imihajlow@gmail.com

11.02.2015

Организация курса.

Структура курса

- 16 ± 1 занятие;
- Три домашних работы. Сроки сдачи:
  - 15 марта,
  - 5 апреля,
  - 26 апреля;
- Одна итоговая курсовая. Срок сдачи 1 июня;
- Промежуточные тесты;
- Возможны бонусы.

Оценивание

При сдаче в срок:

- Курсовая работа 55 баллов,
- Каждая домашняя работа 15 баллов.
- Тесты не оцениваются, но должны быть сданы.

При сдаче позже срока оценка домножается на 0,6.

Если вы набрали менее 50 баллов, то это неудовлетворительно.
Баллы оставшихся сортируются по возрастанию, оценки назначаются так:

- Минимум 20% пятерок,
- Максимум 20% троек,
- Остальное четверки.

Сборка программ из исходников.

От исходников к исполняемому файлу

- Препроцессинг.
- Компиляция.
- Линковка.

В качестве примера компилятора возьмем GCC.

Препроцессинг

- Директивы препроцессора начинаются с символа #.
- Препроцессор просто применяет найденные директивы к тексту программы, не обращая внимания на синтаксис.
- Результат препроцессинга можно увидеть с помощью команды `cpp` или `gcc -E`

Компиляция

- Препроцессор уже отработал (заголовочные файлы включены, макроопределения раскрыты).
- Компилятор преобразует файл с текстом программы в *объектный файл*.
- Выполнить препроцессинг и компиляцию файла можно с помощью команды `gcc -c`

Объектные файлы

Объект

- Имеет имя и область видимости – как минимум одну единицу трансляции.
- Имеет содержимое.

Примеры объектов:

- Функция, метод.
- Статическая переменная.
- Таблица виртуальных функций класса.
- Полностью специализированная шаблонная функция.

Примеры не-объектов:

- Любой тип.
- Пространство имен.
- Неспециализированная шаблонная функция.

Объектные файлы

- Содержат машинный код *объектов* и их имена (*symbols*).
- Содержат ссылки на внешние (extern) объекты.
- Содержат заголовок с информацией о целевой аппаратной платформе.

Статические библиотеки (`lib*.a`) – просто архив из нескольких объектных файлов (`*.o`).
Просмотр таблицы символов объектного файла: команда `nm`.

Объектные файлы

Основные форматы:

- ELF – Linux, BSD;
- Mach-O – Mac (OS X, iOS);
- COFF – Windows.

Программные секции (сегменты)

.text

Исполняемый код. Функции.

.data

Статические неконстантные переменные. Содержит данные для инициализации.

.rodata

Статические константы.

.bss

Неинициализированные статические переменные. Содержит только имена переменных и их размер, инициализация нулями происходит при запуске программы или не происходит вообще.

Пример

test.c

```
void bar();

const int step = 6;
static int n;

void foo() {
    static int count = 1;
    count += step;
    bar();
}
```

test.o

```
.text:
    foo
.data:
    foo::count
.rodata:
    step
.bss:
    n
extern:
    bar
```

Стандартные библиотеки

- Стандартная библиотека языка C (libc) – содержит malloc, free, printf и т.д.
- Стандартная библиотека языка C++ (libstdc++) – содержит new, delete, dynamic_cast, ...
- Стандартный рантайм (crt0.o) – содержит точку входа в программу (_start), вызывает main().

Линковка

- Вход – объектные файлы (*.o), статические библиотеки (архивы объектных файлов), неявно – стандартные библиотеки.
- Разрешаются зависимости, устраняются внешние ссылки. Неустраненные внешние ссылки или многократное появление символов с одним именем вызывают ошибку линковки.
- Выход – исполняемый файл.
- Команда – `ld` (без стандартных библиотек), `gcc` (для языка C), `g++` (для языка C++).

Исполняемый файл – тот же объектный, но содержит точку входа, не содержит внешних ссылок, а всем объектам присвоены адреса.

Язык ассемблера.

Особенности языка ассемблера

- Один „оператор” языка = одна процессорная инструкция;
- Никакой оптимизации;
- Работа с процессором напрямую.

Диалекты ассемблера

- Intel
  - MASM
  - TASM
  - FASM
  - NASM
  - YASM
  - Тысячи их!
- AT&T
  - GAS

YASM/NASM

- Существует для всех основных ОС;
- Контекстонезависимый синтаксис;
- Регистрозависимые имена (как в объектных файлах).

Язык ассемблера

Строчка кода

метка: инструкция операнды ; комментарий

Сначала приемник, потом источник.

Препроцессор YASM/NASM

- Похож на препроцессор C/C++;
- Использует % вместо #;
- Изучается вами самостоятельно.

Основные директивы YASM/NASM

section

`section` *`имя_секции`*

Помещает следующий за директивой код в соответствующую секцию.

Пример:

```
section .text
```

Основные директивы YASM/NASM

extern

`extern` *`имя_символа, ...`*

Объявляет символы внешними (определенными вне текущего модуля).

Пример:

```
extern printf, scanf
```

global

`global` *`имя_символа`*

Объявляет символы доступными извне модуля. Пример:

```
global _start
_start:  ; some code
```

Основные директивы YASM/NASM

align

`align` n

Выравнивает следующую инструкцию на n байт.

```
align 4
```

Псевдоинструкции YASM/NASM

db, dw, ...

`dX` *`expression`*

Размещает данные в памяти.

- db – байт (8 бит),
- dw – слово (16 бит),
- dd – двойное слово (32 бита),
- dq – 64 бита,
- ddq, do – 128 бит.

Примеры:

```
db 12, 34 ; 0x0C 0x22
dd 0x1234 ; 0x34 0x12 0x00 0x00
db 'abc' ; 0x41 0x42 0x43
```

Псевдоинструкции YASM/NASM

resb, resw, ...

`resX` *N*

Резервирует неинициализированное место для *N* переменных.

Используется в секции `.bss`.

Примеры:

```
buffer:  resb 256 ; 256 байт
vector:  resq 3 ; 3 64-битных числа
```

Псевдоинструкции YASM/NASM

equ

label equ *expression*

Присваивает константное значение символу.

Пример:

```
string:  db 'Hello world', 0 ; строка
length:  equ $ - string ; длина строки
```

Примечание: специальный символ $ принимает значение адреса текущей строчки.

IA-32

Программная модель

- Регистры;
- Флаги;
- Память;
- Стек.

Главные регистры

- EAX – „Accumulator register”;
- EBX – „Base register”;
- ECX – „Count register”;
- EDX – „Data register”.

- ErX – 32 бита;
- rX – младшие 16 бит;
- rL – младшие 8 бит;
- rH – вторые 8 бит (биты 8..15).

Пример

EBX = 0xDEADBEEF
BX = 0xBEEF
BH = 0xBE
BL = 0xEF

Индексные регистры

- ESI – „Source index”;
- EDI – „Destination index”;
- EBP – „Base pointer”;
- ESP – „Stack pointer”.

EIP – указатель текущей инструкции.

- Er – 32 бита;
- r – младшие 16 бит;

Сегментные регистры

- CS – „Code segment”;
- DS – „Data segment”;
- ES – „Extra segment”;
- FS – „F segment”;
- GS – „G segment”;
- SS – „Stack segment”.

16 бит!

Флаги

Содержатся в регистре EFLAGS.
Основные флаги состояния:

- CF – беззнаковый перенос;
- ZF – ноль;
- SF – знак (отрицательный);
- OF – знаковый перенос.

Условное выполнение

1. Выполнить арифметическое действие. Флаги будут установлены согласно результату.
2. Выполнить условный переход по одному из флагов.

Адресация в YASM/NASM

`foo` – значение `foo`.
`[foo]` – значение из памяти по адресу `foo`.

Пример

```
number: dd 0x12345678 ; По метке number находится 4 байта
                      ; со значением 0x12345678

mov eax, number    ; Поместить в eax значение (адрес)
                   ; метки number
mov ebx, [number]  ; Поместить в ebx значение
                   ; из памяти по метке number
mov ecx, eax       ; Поместить в ecx значение из eax
mov edx, [eax]     ; Поместить в edx значение
                   ; из памяти по адресу из eax
```

Стек

Растет вниз.
Указатель стека – ESP.
Поместить в стек: `push`
Вынуть из стека: `pop`

Псевдокод

```
push a:
    esp -= sizeof(a)
    [esp] = a
pop a:
    a = [esp]
    esp += sizeof(a)
```

Первая программа.

Простейшая программа на C

hello.c

```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```

Проблемы ассемблера

- `printf` – часть libc;
- `main` вызывается из crt0.o;
- Как вызвать функцию? Как передать туда строку?
- Как вернуть результат функции?

Вывод: слишком сложно для первого раза.

Требования к первой программе

- Должна корректно завершаться.

Текст

exit.asm

```
global _start

section .text
        _start:
                                        ;
                                        ; Call exit(3) syscall
                                        ;       void exit(int status)
                                        ;
        mov     ebx, 0                  ; Arg one: the status
        mov     eax, 1                  ; Syscall number 1
        int     0x80
```

_start

```
global _start

section .text
        _start:
```

Символ `_start` виден снаружи и находится в секции `.text`.
`_start` – точка входа в программу.

mov

```
mov     ebx, 0              ; Arg one: the status
mov     eax, 1              ; Syscall number 1
```

Первые инструкции!
Первым приемник, вторым источник.
Записываем 0 (аргумент функции) в ebx и 1 (номер системного вызова) в eax.
Почему не наоборот? Почему не в edx и esi? Конвенция (соглашение) вызова функций.

syscall

```
int     0x80
```

Программное прерывание номер 0x80 – способ обратиться к ядру линукса.

Сборка программы

Сборка

```
yasm -o exit.o -f elf32 exit.asm
ld -o exit -melf_i386 exit.o
./exit
```

Что почитать

- Yasm user manual
- man ld
- man nm
- man objdump
- man syscall
- man syscalls

Конец.