Введение в архитектуру ЭВМ

Юрий Анатольевич Богоявленский, заведующий кафедрой Информатики и математического обеспечения, к.т.н., доцент: ybgv@cs.petrsu.ru, https://cs.petrsu.ru/

Лекция 1. Введение в дисциплину

План лекции

- Мотивация
- Как организована дисциплина
  - Изучаемая архитектура и инструменты
  - Разделы дисциплины
  - Отчетность.
- Учебник – оглавления, указатель
- Системная среда центрального процессора
  - Архитектура фон Неймана
  - Оперативная память. Единицы доступа и адреса.
- Архитектура IA-32. Базовая среда выполнения
- Компактное представление двоичных чисел
  - Позиционные системы счисления.
  - Представление двоичного числа с помощью 16-х цифр.
  - Порядок байтов в ОП при представлении чисел
- Введение в язык ассемблера
  - Исходный, объектный, исполняемый файлы, формат ELF
  - Базовая идея — символьные имена
  - Форматы команд ЯКЦП

Мотивация

В работе *Peter J. Denning et al. Computing as a Discipline, Final Report of the ACM Task Force on The Core of Computer Science, PP 62, 1988,* дисциплина Информатика определена как содержащая девять предметных областей (subject area):

Algorithms and Data Structures	Numerical and Symbolic Computation
Architecture	Operating Systems
Artificial Intelligence and Robotics	Programming Languages
Database and Information Retrieval	Software Methodology and Engineering
Human - Computer Communication	

В 1991 г. на основе этого определения в работе *Allen B. Tucker (Editor and Co-chair), Bruce H. Barnes (Co-chair) et al. Computing Curricula. Report of the ACM/IEEE-CS Joint Curriculum Task Force, PP 162, 1991* были разработаны рекомендации по разработке учебных планов (curricula) для ВУЗов, которые обновлялись в 2001, 2008, 2013, 2020 годах. В этом процессе количество предметных областей выросло до восемнадцати, однако во всех этих документах предметная область «Архитектура» сохранила свою фундаментальную роль в учебных планах.

Причина такого положения в том, что концепции, понятия и навыки этой предметной области являются важнейшими компонентами квалификации профессионала по ИТ, в том числе, разработчика ПО.

Во первых, они являются базовыми элементами профессионального языка, на котором формулируются постановки задач и изложение других разделов Информатики.

Во вторых, они существенно используются при изучении других предметных областей Информатики, и, в частности, дисциплин, читаемых кафедрой ИМО:

1. **“Операционные оболочки“.**
2. **“Взаимодействующие параллельные системы“.**
3. **“Операционные системы”.**
4. **“Системное программирование”.**
5. **“Компьютерные сети”.**
6. **“Архитектура современных ЭВМ“.**
7. **“Формальные языки и методы трансляции“.**

И в третьих, существует целый ряд актуальных продуктов, которые можно разрабатывать только на архитектурном уровне с

использованием языка ассемблера, либо языка C или других языков системного программирования.

Вот далеко не полный и постоянно расширяющийся перечень таких продуктов:

- **начальный загрузчик, BIOS на ПЗУ;**
- **драйверы устройств и обработчика прерываний;**
- **ПО встраиваемых систем, микроконтроллеров, IoT-устройств, где критичны компактность и производительность;**
- **специфические для процессора команды, не реализованные в компиляторе. Например — команда побитового вращения — для алгоритмов шифрования;**
- **донастройка векторизованных функции для программ на языках более высокого уровня, таких как C, до набора команд SIMD;**
- **пилотажно-навигационные системы обеспечивающие передачу управляющих сигналов от органов управления в кабине экипажа к исполнительным приводам аэродинамических поверхностей (недопустима сборка мусора, подкачки и т. п.);**
- **криптографические алгоритмы, которые всегда должны выполняться за строго одинаковое время для выполнения, предотвращая временные атаки;**

Разработка подобных продуктов требует освоения студентами навыков разработки на уровне архитектуры, формирования у них культуры архитектурного мышления.

В ИМИТ эта культура формируется дисциплиной "Введение в архитектуру ЭВМ", которая читается как базовая на первом или втором курсе. Дисциплина "Архитектура современных ЭВМ" — читается на

третьем курсе, по выбору, опирается на знания полученные ранее, и расширяет навыки разработки на уровне архитектуры за счет углубленного изучение деталей.

При таком подходе необходимо строить схему первой дисциплины так, чтобы, при минимальном количестве деталей, дать студенту возможность освоить базовые архитектурные принципы.

NBNB. Инструмент — язык ассемблера

Как организована дисциплина.

Богоявленский, Ю.А. Цифровая среда Института математики и информационных технологий. Дисциплина «Введение в архитектуру ЭВМ» — минималистский подход [Текст] / Ю.А. Богоявленский // Цифровые технологии в образовании, науке, обществе : материалы XII всероссийской научно-практической конференции (4-6 декабря 2018 года). - Петрозаводск, 2018. - С.31-34. - Режим доступа: https://it2018.petrsu.ru/doc/it2018.pdf

Изучаемая архитектура и инструменты

- **среда - OpenSuSe Linux;**
- **архитектура — только более «компактная»IA-32, INTEL® 64 — в дисциплине "Архитектура современных ЭВМ";**
- **задачи для процессоров реальных ПЭВМ;**
- **язык семейства индустриальных ассемблеров gas (man as, см. TARGET);**

- **профессиональный синтаксис AT&T, который используется компилятором gcc для записи результата трансляции файла pr.c в файл на языке ассемблера pr.S командой gcc . . .-S pr.c если отменить фазу ассемблирования с помощью ключа – S.**
- **демонстрации – отладчик kdbg.**

Разделы дисциплины.

1. Вводная часть.

Системная среда центрального процессора

Регистры, оперативная память. Язык ассемблера, представление на нем команд, символьные имена, характеризация операндов. Пример программы. Системы счисления, порядки байтов Big/Little endian, машинная арифметика, дополнительный код, Unicode.

2. Простые программы.

Системные вызовы на примере read/write, их взаимодействие со стандартными системными файлами stdin/stdout. Примеры на ввод символов с клавиатуры и вывод на экран. Директивы определения данных.

3. Режимы адресации и архитектурный стек

Структура двоичных команд, режимы адресации, перемещение (OFFSET) и смещение (dicplacement). Пример вычисления адресов элементов матрицы, расположенной в оперативной памяти. Стековый доступ к оперативной памяти.

4. Функции

Параметры функции, ее тело, вызов, возврат. Соглашения о связях по стандарту Application Binary Iinterface (ABI) для i386. Команды call, ret, адрес возврата, коды пролога и эпилога, бесконечная вложенность вызовов, локальные переменные, кадр стека. Пример использования библиотеки glibc.

5. Раздельная трансляция и внешние имена

Виды отладки, объектный файл, связь между раздельно оттранслированными объектными файлами, модулей, внешние и внутренние символьные имена, редактор связей ld. Применение соглашений ABI для взаимных вызовов функций на языках ассемблер и C.

NB. Сложность разделов нарастает в любом разделе используется материал из предыдущих.

Рекомендуется активно задавать вопросы, вести конспект, обращаться за консультациями.

Отчетность

ПМиИ, ИСиТ — зачет, ПрИн — экзамен.

Страница дисциплны:

http://kappa.cs.petrsu.ru/~chistyak/architecture/

Баллы, бонусы. Зачет/экзамен – без письменной работы.

2020 г. новость от 02.06.2022 на

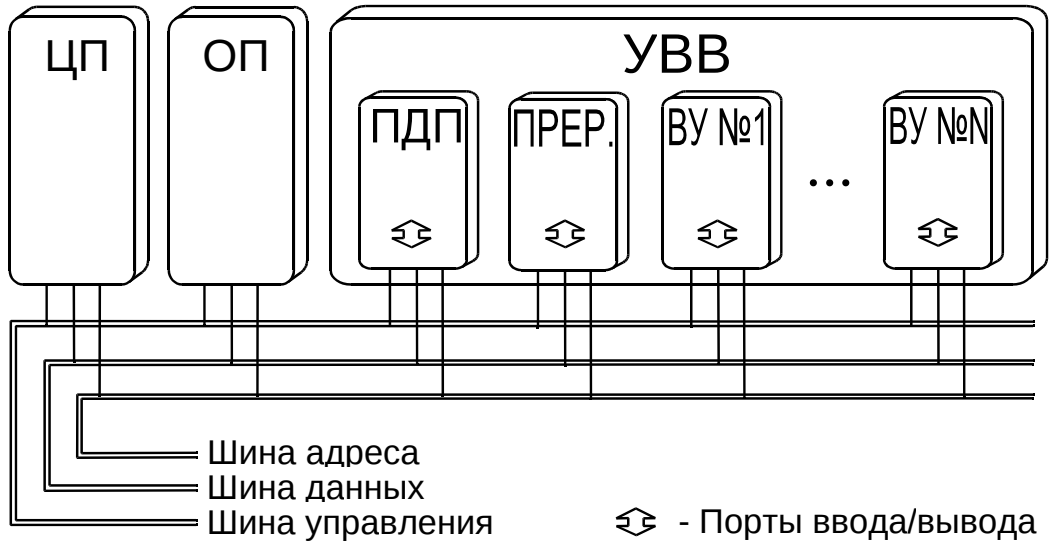
https://cs.petrsu.ru/news/archive.php.ru?q=news2022.xml#prettyPhoto[asm_exam]/0/

Учебник – оглавления, указатель

Богоявленский Ю. А., Дьяконов М. В., Печников А. А. Центральные процессоры персональных ЭВМ

https://edu.petrsu.ru/files/upload/2124_1427288068.pdf

Системная среда центрального процессора, стр. 25



Архитектура фон Неймана

Джона фон Не́ймана — математик с мировым именем (при рождении Я́нош Ла́йош Нейман, при работе в Берлинском университете - Иоганн фон Нейман). Он является ведущим автором работы:

Burks A. W., Goldstine H. H., Neumann J. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. — Institute for Advanced Study, Princeton, N. J., July 1946.

В этой работе были сформулированы основные принципы логический структура ЭВМ, названные позже «архитектура фон Неймана» и реализованные в десятках ЭВМ по всему миру.

1. **Двоичное кодирование информации**
2. **Неразличимость команд и данных**
3. **Адресуемость оперативной памяти**
4. **Последовательное выполнение команд**

Оперативная память. Единицы доступа и адреса.

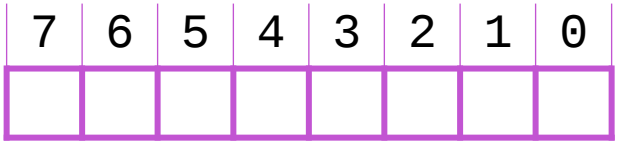
Единица доступа это блок ОП, который может быть обработан за одну команду ЦП. Байт — минимальная единица доступа. Байты ОП и их последовательные номера реализованы аппаратно.

Физический адрес — это аппаратный номер байта, физическое адресное пространство — вся совокупность байтов ОП.

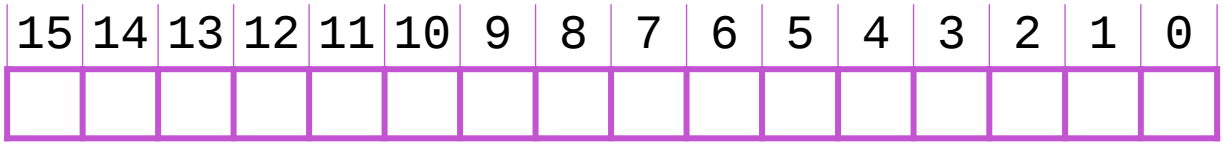
Байты, состоят из физически реализованных двоичных цифр (Binary digit) — bit — 0/1. Биты нумеруются СПРАВА НАЛЕВО в порядке степеней разрядов в представлении двоичного числа:

$$Q = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

Байт. Суффикс кода команды — b, пример — movb



Слово. Суффикс кода команды — w, пример - addw



Двойное слово. Суффикс кода команды — l, пример - cmpl

Архитектура IA-32. Базовая среда выполнения

Материал этого раздела взят из руководства «Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 1: Basic Architecture»

IA-32 (сокращение от "Архитектура Intel, 32-разрядная"), называемая также i386 - это поддерживающая 32-разрядные вычисления архитектура x86, разработанная Intel и впервые реализованная в процессоре 80386 в 1985 году.

Аббревиатура "IA-32" используется как общее обозначения для всех версий x86, обеспечивающих такую поддержку. IA-32 полноценно поддерживается процессорами с архитектурой Intel®64, операционными системами и системами программирования.

Архитектура IA-32 поддерживает три основных режима работы:

I. **Protected mode — защищенный режим с возможностью многозадачного выполнения ПО для процессора Intel 8086;**
II. **Real-address mode — реализует среду программирования процессора Intel 8086. Процессор работает в этом режиме после включения питания или нажатия кнопки Reset.**

III. System management mode — **режим управления системой** (System management mode).

Режим работы определяет, какие команды и архитектурные функции доступны.

Любой программе или задаче, выполняющейся на процессоре IA-32, **предоставляется набор ресурсов для выполнения машинных команд и для хранения кода, данных и информации о состоянии. Эти ресурсы составляют базовую среду выполнения (БСВ) для процессора** IA-32.

Процессоры Intel®64 **поддерживают БСВ архитектуры** IA-32 **и аналогичную среду в режиме** IA-32e**, которая может выполнять 64-разрядные программы (64-разрядный подрежим) и 32-разрядные программы (подрежим совместимости).**

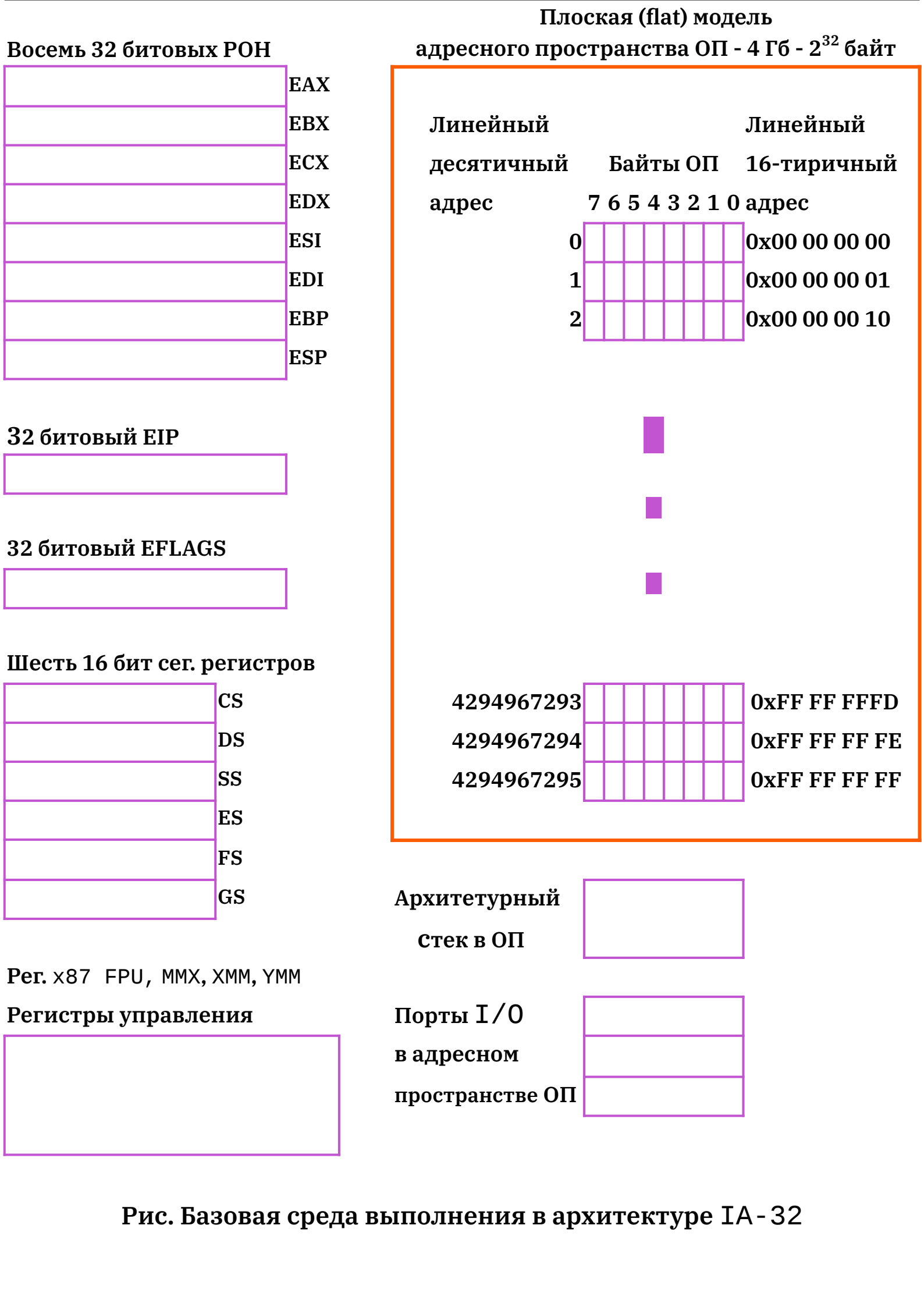


Рис. Базовая среда выполнения в архитектуре IA-32

БСВ используется прикладными программами и операционной системой и содержит следующие элементы.

I. Адресное пространство ОП

- **Физическое адресное пространство объемом до 64 Гбайт (2^{36} байт).**
  - **Модели памяти IA-32**
    - **При использовании средства управления памятью процессора, программы напрямую не обращаются к физической памяти а используют одну из трех моделей памяти: плоскую (`Flat memory model`), сегментированную (`Segmented memory model`) или реального адреса (`Real-address mode memory model`), которая соответствует модели памяти процессора `8086`.**
  - **Мы будем работать только в модели плоской памяти — едином непрерывном т. н. линейном адресном пространстве объемом 4 Гб, содержащем код, данные и стеки. Адреса байтов называются линейными адресами и задаются в диапазоне 0 — (2^{32}-1). Т. о. мы НЕ РАССМАРИВАЕМ использование сегментных ьрегистров.**

II. Базовые регистры выполнения программ

- **восемь 32 битовых регистров общего назначения (РОН);**
- **шесть 16 битовых сегментных регистров;**
- **32 битовые регистры EFLAGS и EIP (указатель команд).**

составляют среду в которой выполняется команды общего назначения — целочисленная арифметика, управление порядком выполнения, операции с битовыми и байтовыми строками, адресация памяти. Эти регистры (кроме сегментных) буду рассмотрены на следующей лекции.

- **В БСВ IA-32 входят также не рассматриваемые нами регистры x87 FPU (операции над числами с плавающей точкой), регистры MMX, XMM, YMM для поддержки операций SIMD (single-instruction, multiple-data) с различными данными**
- **Различные регистры управления.**

III. Архитектурный стек (ОС выделяет блок в ОП).

IV. Порты ввода/вывода (I/O) в адресном пространстве ОП.

Компактное представление двоичных чисел

- **Веб-разработка — цветов в CSS, #FF5733 — оранжевый.**
- **Криптография — представления хеш-значений.**
- **Сетевые технологии — IP-адреса, 2001:0db8:85a3 — в протоколе IPv6.**
- **Системное программирование — ошибки и состояния — 0×0000FFFF указывает на тип сбоя.**
- **Адреса памяти и команды, например в файле листинга**
  - `57 0000 29090000  n:  .long 2345  # int n = 2345;` число
  - `73 0006 A1000000  movl n, %eax  # готовим к команде` деления
- **Языки C, C++, Python и т.п. — работа с битами.**

Позиционные системы счисления.

Вес цифры зависит от позиции $525 = 5 \cdot 10^2 + 2 \cdot 10^1 + 5 \cdot 10^0$

Общая формула позиционной системы счисления.

$$Q = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \ldots + a_1 \cdot p^1 + a_0 \cdot p^0$$

a_i **— цифры системы,** p **— ее основание.**

Рассмотрим системы с основаниями 2, 8, 16.

p	цифры														
2	0	1													
8	0	1	2	3	4	5	6	7							
16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
											10	11	12	13	14	15

Представление двоичного числа с помощью 16-х цифр.

Это особенно удобно для восьми битовых байтов (октетов) в которых каждой двоичной тетраде — полубайту (nibble, nybble) взаимно однозначно соответствуют шестнадцатеричная цифра. Так, 32 разрядное двоичное число — с весьма длинной и непонятной записью:

`10010011111010100000110100111111`

`(тетрады — 1001 0011 1110 1010 0000 1101 0011 1111)`

будет представлено в четырех байтах (восьми полубайтах) в естественном порядке как `0x93EA0D3F` (`0x` указывает, что число представлено в шестнадцатеричной системе).

Порядок байтов в ОП при представлении чисел

В том случае, если число не может быть представлено одним байтом, нужно зафиксировать в архитектуре системы, в каком порядке байты будут храниться в ОП или передаваться по линиям связи. Принято два таких порядка.

От младшего к старшему - `Little endian` (остроконечный) - байты младших цифр имеет меньший адрес. Число 93EA0D3F в этом порядке будет представлено так:

Адрес ОП	a+0	a+1	a+2	a+3
Байты числа	3F	0D	EA	93

Преимущество. Адрес двойного слова, младшего слова и младшего байта совпадают и равны, для этого примера - a+0. Для порядке Big endian это не так.

От старшего к младшему - Big endian (тупоконечный) - байты младших цифр имеет больший адрес. Число 93EA0D3F в этом порядке будет представлено так:

Адрес ОП	a+0	a+1	a+2	a+3
Байты числа	93	EA	0D	3F

Порядок байтов выбирает конструктор архитектуры ЭВМ.

Порядок Little endian принят а архитектурах Intel, Big endian - в сетях TCP/IP (network byte order), архитектурах Motorola, соответствует порядку десятичной системы счисления.

ЯКЦП – набор двоичных команд, основной цикл ЦП, стр. 29.

До начала 50-х годов программировали в двоичных кодах используя 8 или 16 представление двоичных команд и данных.

Введение в язык ассемблера.

Исходный, объектный, исполняемый файлы, формат ELF

Assembly Language **– символьное представление команд, адресов и данных.**

NBNB. Assembly language **may also be called symbolic machine code**

Ассемблер – **транслятор с языка ассемблера на ЯКЦП.**

p.S – **исходный символьный код на языке ассемблера;**
p.o – **объектный файл – команды ЯКЦП, к исполнения не готов;**
p – **исполняемый файл, готов к запуску процесса загрузчиком;**
p.lst – **файл листинга, код ЯКЦП, таблица символьных имен.**

Получение выполняемого файла

Входной файл	Имя программы	Выходной файл
p.S	**Ассемблер** as	p.o p.lst **(по ключам)**
p.o	**Редактор связей (Компоновщик)** ld	p

Файлы p.o **и** p **в нашей среде имеют формат** ELF - (Executable and Linkable Format - Исполняемый и связываемый формат) **— общий стандартный формат для файлов:**

- **объектных;**
- **исполняемых;**
- **общих библиотек;**
- **ядер UNIX-подобных ОС;**
- **основных дампов.**

ELF — **кроссплатформенный, поддерживает 32 и 64 разрядные режимы, применяется в нескольких десятках ОС и аппаратных**

платформ с соответствующими ABI (Application binary interface — Двоичный интерфейс приложений), который рассмотрим в разделе «Функции». Описание ELFдано на ресурсе: Linux Foundation. Referenced Specifications, URL: https://refspecs.linuxbase.org/ в разделе ELF and ABI Standards.

Базовая идея — символьные имена

Всем объектам, используемые при разработке программ на языке ассемблера даются символьные имена, напоминающие смысл этих объектов в естественном языке. Существуют следующие основные группы объектов.

Объект	Кто формирует символьное имя	Пример (для gas)
Двоичный код команды ЯКЦП	Разработчик архитектуры процессора	`addb` — сложить `movw` — переслать `cmpl` — сравнить

Регистр процессора	Разработчик архитектуры процессора	`%eax (32 бита)`
		`%ax (16 бит)`
		`%ah (8 бит)`
		`%eip (32 бита)`

Директива ассемблера	Авторы языка	`.byte`
		`.word`
		`.data`
		`.text`
		`.align`

Разнообразные артефакты, создаваемые программистом, например:
• Двоичный адрес (метка) в ОП:
- данного;
- перехода;
	Программист	`length:`
		`Next_digit:`
		`n:`
		`Finish`

- функции. • Константа. • Макрокоманда. • Секция. • Библиотка.		

Символьные имена используются редактором связей для объединения раздельно транслируемых и библиотечных модулей, а также отладчиком.

Структура программы.

Исходный модуль состоит, как правило, из трех секций :

- данных — содержит директивы определения данных, следующие после каждого вхождения директивы `.data;`
- неинициализированных данных не помещаемых в объектный фвйл, содержит директивы, следующие после каждого вхождения директивы `.bss`;
- команд процессора — содержит команды, следующие после каждого вхождения директивы `.text`.

Хотя бы одна секция должна присутствовать. При ассемблировании исходного файла, содержащего тексты секций они транслируются в машинные представления данных и команд и объединяются в один объектный файл.

NBNB. Секция `.bss` в объектном файле представлена только ее заголовком и не занимает места в файле. Память для нее выделяется загрузчиком при запуске на выполнение.

Символьные имена получают при ассемблировании значение и тип, некторые из них указывает на секцию, где определено имя — `t, d, b` или `a` (возможны также `T, D, B, A`, которые рассмотрим позже.). Значения символьных имен могут быть адресами ОП (адресными) или абсолютными (рассмотрим позже).

В конце файла листнг выводится таблица определенных в исходном файле символьных имен. Пример для задачи 1.

```
DEFINED SYMBOLS
 task1.S:57     .data:0000000000000000 n
 task1.S:58     .data:0000000000000004 length
 task1.S:59     .data:0000000000000008 ten
 task1.S:66     .text:0000000000000000 _start
 task1.S:75     .text:000000000000000b nextdigit
```

Адресные значения вычисляются в процессе перевода с языка ассемблера команд процессора и директив в команды ЯКЦП и данные и размещения их в объектный файл. При этом двоичные образы данных в секции `.data` и команд в секции `.text` размещаются последовательно начиная с адреса `0x00000000`. Затем редактор связей `ld` перемещает эти значения для работ в выделенном участке ОП.

Форматы команд ЯКЦП

Формат ассемблера `gas` в синтаксисе `AT&T`

В этом представлении команда ЯКЦП может иметь три формы: (синтаксис `AT&T` см. в http://kappa.cs.petrsu.ru/~chistyak/architecture/docs/gas/gas-8.html#ss8.1 **)**

- **коп`{b|w|l}` операнд1 (источник), операнд2 (приемник — результат операции);**

- **коп{b|w|l} операнд;**
- **коп.**

Обозначим местоположение и способ задания операндов:

- R **— в регистре, символьным именем регистра;**
- M **— в ОП, символьным именем данного в ОП с адресным значением;**
- I **— в команде (непосредственный операнд), значением или символьным именем.**
- **NBNB. По синтаксису** AT&T **непосредственноу операнду должен предшествовать символ «$»**

Если операндов два, то возможны любые их комбинации кроме M,M. **Также нельзя использовать операнд типа** I **как приемник .**

Примеры (часть из исходного файла task1.S**)**

- movl $0, %ebx **— переслать значение 0 длиной 32 бита (непосредственный операнд) в регистр** ebx**;**
- movl n, %eax **— переслать значение из области ОП, распределенной для символьного имени** n **в регистр** eax**;**
- movl %ebx, length **— переслать значение из регистра** ebx **диной 32 бита в область ОП, распределенную для символьного имени** length**;**
- movw %bx, %dx **— переслать значение из регистра** bx **диной 16 бит в регистр** dx**;**
- movb %bh, %cl **— переслать значение из старшего байта регистра** ebx **длиной 8 бит в младший байт регистра** ecx**;**
- incl %ebx **— увеличить на 1 значение в 32 битовом регистре;**
- sti **— присвоить значение 1 флагу прерываний.**

NBNB. Команды и операнды (стр. 74 книги) заданы в синтаксис Intel:

коп операнд1 (приемник — результат), операнд2 (источник), в синтаксисе AT&T – наоборот, не так, как в книге.

Примечание.

Видно, что команда обычно имеет 0, 1 или 2 операнда. Команда умножения imul может иметь три операнда. При этом в синтаксисе AT&T ее нужно задавать в виде:

imul i,r1,r2 — r2 = i*r1 или imul i,m,r — r = i*m,

где i — непосредственный операнд, r, r1,r2 — имена регистров, m — символьное имя операнда в памяти.

Формат процессора, его соответствие формату ассемблера

В формате процессора команда имеет структуру:

Префикс команды	Префикс разрядности адресации	Префикс разрядности операнда	Префикс сегмента	Код операции	Байт ModR/M	Байт SIB	Смещение	Операнд в команде
0/1 байт	0/1 байт	0/1 байт	0/1 байт	1/2 байт	0/1 байт	0/1 байт	0/1/2/4 байт	0/1/2/4 байт

Рис. 14 в книге «Центральные процессоры персональных ЭВМ»

NB Не использовать рис. на стр. 75.

NBNB. Несколько упрощая зафиксируем, что при формировании двоичной команды из ассемблерной указанные в первом столбце таблицы элементы последней заменяются соответствующими двоичными кодами, указанными во втором столбце и размещаются в указанных в третьем столбце полях двоичной команды.

Элемент ассемблерной команды	Двоичный код	Поле команды процессорного формата
коп{b|w|l}	Двоичный КОП	Код операции
R операнд - символьное имя регистра	Код регистра	Элемент поля ModR/M
M операнд - символьное имя операнда в памяти	Адресное значение символьного имени	Смещение
I операнд - непосредственный операнд в команде	Двоичное значение операнда	Операнд в команде

Остальные поля двоичной команды формируются автоматически в зависимости от суффикса кода ассемблерной команды и других ее параметров.

NBNB. Уже на этом уровне проявляются преимущества языка ассемблера.

Символьным именам команд соответствует, как правило, несколько команд ЯКЦП с разными двоичными кодами, например команда mov (стр. 85) имеет 20 различных кодов в зависимости от типа (R, M или I), размера {b|w|l} операндов и комбинации приемник/источник.

Иллюстрации команд и данных из файлов задачи 1

1. Секция .data — фрагмент файла task1.lst

```
55     .data     # секция данных, распределение памяти
56               # соотв. конструкция языка C и коммент.
57 0000 29090000     n:      .long 2345       # int n = 2345; число
58 0004 00000000     length: .long 0          # int length =0; результат
59 0008 0A000000     ten:    .long 10         # определяем константу ЯВНО
60                                             # нет аналога в C
```

2. Таблица символов объектного файла

```
ybgv@ybgv-home:~/GnuAS/Labs/1> objdump -t -j .data task1-exe-S.o

task1-exe-S.o:     формат файла elf32-i386

SYMBOL TABLE:
00000000 l    d  .data  00000000 .data
00000000 l       .data  00000000 n
00000004 l       .data  00000000 length
00000008 l       .data  00000000 ten
```

3. Таблица символов исполняемого файла

```
ybgv@ybgv-home:~/GnuAS/Labs/1> objdump -t -j .data task1-exe-S

task1-exe-S:     формат файла elf32-i386

SYMBOL TABLE:
080490a2 l       .data  00000000 n
080490a6 l       .data  00000000 length
080490aa l       .data  00000000 ten
080490ae g       .data  00000000 __bss_start
080490ae g       .data  00000000 _edata
080490b0 g       .data  00000000 _end
```

3. Соответствие ассемблерных и двоичных команд ЯКЦП

```
ybgv@ybgv-home:~/GnuAS/Labs/1> objdump -dS task1-exe-S

task1-exe-S:     формат файла elf32-i386


Дизассемблирование раздела .text:

08048074 <_start>:
.text # секция команд процесора

.global _start     # точка входа - глобальная метка

_start:
     nop            # пустая операция - no operation
 8048074:       90                      nop

#   нужна, чтобы задать отладчику контр. точку останова на следующей,
первой
#   содержательной команде программы

     movl $0, %ebx   # counter = 0; счетчик делений
```

```
 8048075:       bb 00 00 00 00          mov    $0x0,%ebx
    movl n, %eax    # готовим к команде деления idivl
 804807a:       a1 a2 90 04 08          mov    0x80490a2,%eax

0804807f <nextdigit>:
                # нет аналога в C
nextdigit:
        movl  $0, %edx          # еще готовим, уже в цикле
 804807f:       ba 00 00 00 00          mov    $0x0,%edx
#   0.  После выполнения эта команда
#   помещает в EDX - остаток, в EAX - частное, что
#   НАРУШАЕТ ИНТЕРПРЕТАЦИЮ значений пары EDX,EAX. Перед следующим делением
эту
#   интерпретацию надо восстановить, присвоив EDX значение 0.

        idivl ten       # делим объединенные регистры edx:eax на 10
 8048084:       f7 3d aa 90 04 08       idivl  0x80490aa
                  # частное в  eax, остаток в edx

        incl %ebx       # ++counter; счетчик делений + 1
 804808a:       43                      inc    %ebx

#    две следующие команды соответствуют условному оператору
#    if (quotient) goto nextdigit;

        cmpl $0, %eax    # частное > 0 ?
 804808b:       83 f8 00                cmp    $0x0,%eax
        jg   nextdigit   # ДА, продолжаем
 804808e:       7f ef                   jg     804807f <nextdigit>
команда cmpl установила в регистре флагов биты, указывающие, что значение в
регистре %eax,т.е. частное после деления, СТРОГО  БОЛЬШЕ нуля.

*/

        movl %ebx, length # length = counter; НЕТ,сохраняем результат
 8048090:       89 1d a6 90 04 08       mov    %ebx,0x80490a6

    Finish              # конец работы,
 8048096:       bb 00 00 00 00          mov    $0x0,%ebx
 804809b:       b8 01 00 00 00          mov    $0x1,%eax
 80480a0:       cd 80                   int    $0x80
```

Выводы

1. Необходимо владеть первичными навыками разработки на уровне архитектуры, иметь культуру архитектурного мышления.

2. Используется среда - OpenSuSe Linux, архитектура IA-32, семейств ассемблеров gas, синтаксис AT&T, отладчик kdbg.

3. Процессор предоставляет программе базовую среду выполнения IA-32, содержащую регистры и оперативную память, имеющую единицы доступа.

4. Двоичные числа удобно представлять в 16-ной системе счисления, в ОП в архитектурах Intel принят порядок байтов «от младшего к старшему» - Little endian.

5. Исходный файл p.S ассемблером as ассемблериуется в объектный файл p.o из которого редактор связей (компоновщик) ld получает исполняемый файл p. Файлы p.o и p в нашей среде имеют формат ELF.

6. Всем объектам, используемые при разработке программ на языке ассемблера даются символьные имена, имеющие значение и тип. Адресные значения вычисляются в процессе перевода с языка ассемблера команд процессора и директив в команды ЯКЦП и данные и размещения их в объектный файл.

7. Исходный модуль состоит из трех секций: .data, .text, .bss.

Ассемблерный и процессорный форматы команд ЯКЦП имеют соответствие полей.

8. В оперативной памяти может находиться только один операнд команды.

9. При ассемблировании поле «Смещение» команды процессорного формата заменяется двоичным адресным значение символьного имени из ассемблерного формата.