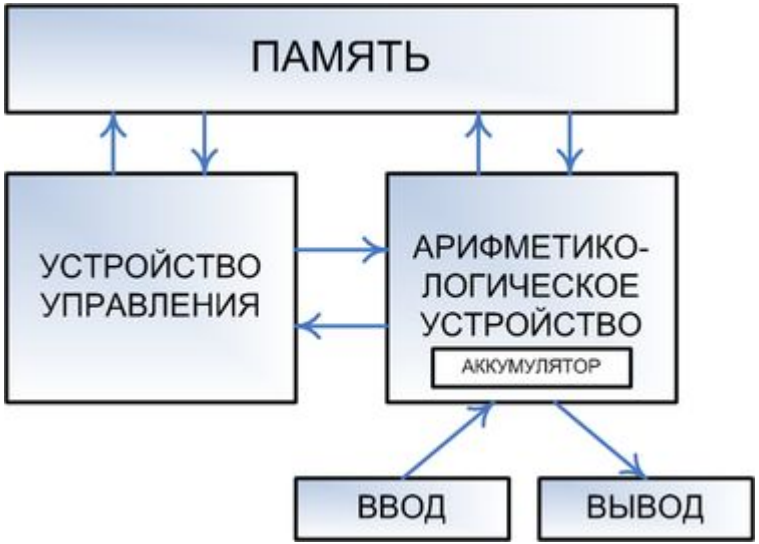
Семинар 2. Введение в ассемблер

Архитектура Фон Неймана

- Принцип однородности памяти
- Принцип адресности
- Принцип программного управления



Введение в ассемблер

- Мы будем использовать AT&T синтаксис для записи инструкций ассемблера x86
- Для компиляции используем **gcc - gcc -m32 source.S**
- Расширение для файлов исходного кода - **.S**
- 64-битная версия системы может не поддерживать 32-битную архитектуру, лечение - **sudo apt-get install gcc-multilib**
- Первое время разрешено использовать мини-библиотеку **simpleio_i686.S**

Регистры процессора

- Сверхбыстрая память внутри процессора, предназначенная прежде всего для хранения промежуточных результатов вычисления или содержащая данные, необходимые для работы процессора
- Для большинства операций значение должно быть загружено в один из регистров
- Регистры общего назначения, сегментные регистры, регистр флагов, указатель команды

- **%eax**: Accumulator register — аккумулятор, применяется для хранения результатов промежуточных вычислений
- **%ebx**: Base register — базовый регистр, применяется для хранения адреса (указателя) на некоторый объект в памяти
- **%ecx**: Counter register — счетчик, его неявно используют некоторые команды для организации циклов
- **%edx**: Data register — регистр данных, используется для хранения результатов промежуточных вычислений и ввода-вывода

Регистры общего назначения

- **%esp**: Stack pointer register — указатель стека. Содержит адрес вершины стека
- **%ebp:** Base pointer register — указатель базы кадра стека (англ. stack frame). Предназначен для организации произвольного доступа к данным внутри стека
- **%esi**: Source index register — индекс источника, в цепочечных операциях содержит указатель на текущий элемент-источник
- **%edi**: Destination index register — индекс приёмника, в цепочечных операциях содержит указатель на текущий элемент-приёмник

Регистр флагов

- Нужно рассматривать как массив битов, за каждым из которых закреплено определённое значение
- Неявно передаётся дополнительная информация, которая не записывается непосредственно в результат вычислений
- **cf**: carry flag, флаг переноса (1 - был перенос из старшего бита результата, беззнаковое переполнение)
- **zf**: zero flag, флаг нуля (1 - результат операции нулевой)
- **of**: overflow flag, флаг переполнения (1 - было знаковое переполнение)

Указатель команды

- **eip**: instruction pointer, указатель команды, регистр напрямую недоступен, изменяется неявно командами условных и безусловных переходов, вызова и возврата из подпрограмм

Команды

- Состоят из обозначения инструкции процессора и операндов
- Операндов может не быть, или быть несколько (до 3х)
- Конкретное значение, известное на этапе компиляции (непосредственные) - через **$**
- Регистр - название регистра с префиксом **%**
- Указатель на ячейку памяти - об этом в следующий раз

Команды

- Суффикс обозначения команды говорит о том, к какому количеству байт применяется операция
- b (от англ. byte) — 1 байт
- w (от англ. word) — 2 байта
- l (от англ. long) — 4 байта
- q (от англ. quad) — 8 байт

Команды

- Важной особенностью всех команд является то, что они не могут работать с двумя операндами, находящимися в памяти. Хотя бы один из них следует сначала загрузить в регистр, а затем выполнять необходимую операцию

Метки и символы

- Метка — это просто константа, значение которой — адрес
- **hello_str: .string "Hello"**
- Псевдометка **.** - текущий адрес
- Значение метки - всегда адрес. Символ - некоторая константа (в т.ч. метка - символ)
- Символ можно сделать глобальным (экспортируемым) с помощью директивы **.global**

Некоторые команды

- Суффиксы - b - 1 байт, w - 2 байта, l - 4 байта, q - 8 байт
- **mov ИСТОЧНИК, НАЗНАЧЕНИЕ**
- Хотя бы один из аргументов - регистр

Некоторые команды

- **inc ОПЕРАНД**
- **dec ОПЕРАНД**
- **add ИСТОЧНИК, ПРИЕМНИК**
- **sub ИСТОЧНИК, ПРИЕМНИК**
- **mul МНОЖИТЕЛЬ1** - второй множитель находится в **%eax**, результат - в **%eax** и **%edx**

Некоторые команды

- **loop МЕТКА** - уменьшить %ecx на 1, если получился 0 - идти дальше, нет - перейти на метку
- **cmp ОПЕРАНД1, ОПЕРАНД2** - вычислить ОПЕРАНД1 - ОПЕРАНД2 и установить флаги. Результат не сохраняется
- **jz, jnz, jc, jnc, jo, jno, jg, jge, jl, jle, ja, jae, jb, jbe** - условные переходы на основании значений флагов, аргумент - адрес (метка)
- **jmp АДРЕС** - безусловный переход

Некоторые команды

- **and, or, xor, not** - логические операции, значение записывается в приемник (последний аргумент)
- **test** - как команда , но не сохраняет результат, а только устанавливает флаги
- Рекомендуется использовать **test** вместо cmp для сравнения с нулем
- **xor** часто применяют для обнуления регистров

Некоторые команды

- **sal/shl КОЛИЧЕСТВО_СДВИГОВ, НАЗНАЧЕНИЕ** - Shift Arithmetic Left / SHift logical Left
- **sar/shr** - аналогично для правого сдвига
- Каждый «выдвигаемый» бит попадает в флаг **cf** (сохраняется последний)
- **ror, rol** - циклический сдвиг вправо/влево, в cf сохраняется последний выдвинутый бит
- **rcr, rcl** - циклический сдвиг с cf как дополнительный бит