КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 00 — Вводная.

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2022

2022.02.09

Оглавление

Рабочая программа (2014.12.9)

Раздел 5. Язык ассемблера. Архитектура персонального компьютера.

13	Архитектура персонального компьютера. Типы данных и их представление	Введение. Архитектура персонального компьютера. Системы счисления. Двоичная арифметика. Типы данных и порядок байт. Числа с плавающей запятой.
14	Устройство процессора. Организация памяти программы. Структура программы.	Программно-доступные регистры процессора. Сегментная организация памяти. Модели памяти. Структура программы типа COM и EXE. Организация стека.
15	Форматы команд. Директивы ассемблера. Компоновка программ.	Форматы команд. Стандартные директивы определения сегментов. Простейшие директивы определения сегментов. Объявление и инициализация данных. Резервирование памяти. Подготовка, компиляция, компоновка, загрузка, отладка и выполнение ассемблерных программ.

Рабочая программа (продолжение)

Раздел 5. Язык ассемблера. Архитектура персонального компьютера (ч.2).

16	Способы адресации данных. Основные операции работы с данными.	Непосредственная, прямая, регистровая, косвенная регистровая, относительная косвенная регистровая, базовая индексная и неявная адресации. Команды пересылки данных.
17	Основные команды языка ассемблера	Арифметические и логические команды. Команды сдвига. Команды передачи управления. Оператор безусловного перехода. Операторы условного перехода. Внутрисегментные и межсегментные прямые и косвенные переходы. Команды организации циклов. Строковые команды. Префиксы повторения.

Рабочая программа (продолжение)

Раздел 5. Язык ассемблера. Архитектура персонального компьютера (ч.3).

18	Ввод-вывод данных. Взаимодействие с аппаратурой	Ввод информации с клавиатуры. Системная процедура обработки прерываний от клавиатуры. Использование функций DOS, BIOS для ввода информации. Использование средств DOS, BIOS, обращение как к файлу. Использование системных вызовов для ввода/вывода информации. Работа с файлами. Работа с портами. Логическая организация видеобуфера. Прямое обращение к видеобуферу.

Рабочая программа (продолжение)

Раздел 6. Структура многомодульных программ на языках высокого уровня и языке ассемблера.

19	Процедуры. Интерфейс взаимодействия с программами на языках высоко-го уровня	Процедуры. Способы передачи параметров в процедуру. Использование встроенного ассемблера (AT&T). Вызов ассемблерной процедуры из С-функции. Вызов функций, написанных на С из программ на языке ассемблера. Соглашения о вызове процедур для режимов работы 32 и 64 бит.
20	Макросредства. Модули программ.	Повторяющиеся блоки. Макрокоманды. Использование библиотек макросов. Связь по данным между модулями. Структура многомодульной программы.

Рабочая программа (продолжение)

Раздел 6. Конструирование программ. Структура многомодульных программ на языках высокого уровня и языке ассемблера (ч.2).

21	Управление процессами и систе-ма прерываний	Распределение адресного пространства ПЭВМ. Процедура обработки прерывания в ПЭВМ. Структура обработчика прерываний.
22	Параллельное и непараллельное программирование. Метаязыки. Сценарные (скриптовые) языки.	Основы параллельного программирования. Понятие потока и контекста. Конкуренция при доступе к ресурсам. Модели памяти. Алгоритмы обеспечения когерентности кэша. Понятие метаязыка. Понятие интерпретатора.

Арифметические расширения команд процессора: FPU, MMX, XMM, SSE, SSE2, SSE3

```
$ lscpu
Архитектура:             x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Порядок байт:          Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
ID прроизводителя:       GenuineIntel
  Имя модели:            Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
    Семейство ЦПУ:       6
    Модель:              165
    Thread(s) per core:  2
    Ядер на сокет:       6
    Сокетов:             1
    Степпинг:            2
    CPU max MHz:         5000,0000
    CPU min MHz:         800,0000
    BogoMIPS:            5199.98
```

Флаги: **fpu** vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr **sse** **sse2** ss ht tm pbe **syscall** nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc **cpuid** aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 **ssse3** sdb g **fma** cx16 xtpr pdcm pcid **sse4_1** **sse4_2** x2apic movbe **popcnt** tsc_deadline_timer aes xsave **avx** f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb in vpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 **avx2** smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp pku ospke md_clear flush_l1d arch_capabilities

```
Virtualization features:
  Виртуализация:         VT-x
Caches (sum of all):
  L1d:                   192 KiB (6 instances)
  L1i:                   192 KiB (6 instances)
  L2:                    1,5 MiB (6 instances)
  L3:                    12 MiB (1 instance)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-11
Vulnerabilities:
  Itlb multihit:         KVM: Mitigation: VMX disabled
  L1tf:                  Not affected
  Mds:                   Not affected
  Meltdown:              Not affected
  Spec store bypass:     Mitigation; Speculative Store Bypass disabled via prctl and seccomp
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:            Mitigation; Enhanced IBRS, IBPB conditional, RSB filling
  Srbds:                 Not affected
  Tsx async abort:       Not affected
```

Литература

0) Stephen Prata C Primer Plus. Sixth Edition. Addison-Wesley. 2014.

1) Прата, Стивен. Язык программирования С. Лекции и упражнения, 6-е изд. : Пер. с англ. — М.ООО “И.Д. Вильямс”, 2015. — 928 с. : ил.

2) Peter Prinz, Tony Crawford. C in a Nutshell, Second Edition. O’Reilly Media, Inc., 2015.

3) ISO/IEC 9899-2011[2012] — Information technology — Programming languages — C.

4) Ричард Столмен. Отладка с помощью GDB / Столмен Р., Пеш Р., Шебс С. и др. FSF Inc., 2000

5) Столяров А. В. Программирование: введение в профессию. II: Низкоуровневое программирование. — М.: МАКС Пресс, 2016, 496 с.

6) Аблязов Р. З. Программирование на ассемблере на платформе x86-64. — М.: ДМК, 2011. — 304 с.: ил.

7) Hall B. R. Assembly Programming and Computer Architecture for Software Engineers / Brian R. Hall, Kevin J. Slonka. — Prospect Press, Inc., 2018, 413 p.

8) http://rutracker.org, http://gen.lib.rus.ec

9) Intel 64 and IA-32 Architectures Software Developer'\''s Manuals. 325462-074US (April 2021).pdf

Платформа и инструментарий

Операционная система — **Linux.** (базовый, Dual-boot, VM)

Как вариант, BSD, OS X, QNX, и любая U*X-совместимая, поддерживающая файловую систему стандарта POSIX, работу в локалях utf8, в частности ru_RU.utf8, в том числе и в консоли.

Ассемблер — **nasm**;

Компилятор C — **gcc**;

Сборка программ — **make**;

Отладчик для C — **gdb**;

Отладчик для nams — **edb**;

Управление файлами — **mc** (Midnight Commander).

Компоновщик — **ld** (binutils);

Редактор текста или IDE, поддерживающие подсветку синтаксиса (kate, **slickedit**).

Файл **stud_io.inc** из архива материалов к книге «Столяров А.В. Программирование: Введение в профессию. Т.2. Низкоуровневое программирование. 2016.pdf»

Тестовый проект

```
$ ls -1
makefile
stud_io.inc
test.asm
$
```

Программа

```
$ cat test.asm
%include "stud_io.inc"
global _start
section   .text
_start:
    PRINT  "Привет, nasm!"
    PUTCHAR   10
    FINISH
$
```

Содержимое файла makefile

```
#  makefile
all: test
# правила сборки
test: test.o
    ld -m elf_i386 -o test test.o

# правила ассемблирования
test.o: test.asm makefile
    nasm -Wall -f elf -gdwarf -l test.lst -o test.o test.asm

.PHONY: clean
clean:
    rm -f test.o test.lst
```

Отладчик EDB

Этот отладчик, возможно, есть в составе некотороых дистрибутивов.

Тем не менее, имеет смысл его собрать самому.

1. Устанавливаем из дистрибутива необходимые зависимости (-devel), если они не были установлены ранее:

```
Qt5, Boost, Capstone, Graphviz
```

В Fedora Core

```
$ sudo dnf install qt5-devel boost-devel capstone-devel graphviz-devel
```

2. Скачиваем, собираем и устанавливаем библиотеку **gdtoa**

```
$ git clone https://github.com/10110111/gdtoa-desktop.git
$ cd  gdtoa-desktop
$ mkdir build
$ cd  build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ make
$ sudo make install
```

3. Скачиваем, собираем и устанавливаем **edb**

```
$ git clone --recursive https://github.com/eteran/edb-debugger.git
$ cd edb-debugger
$ mkdir build
$ cd  build
$ cmake ..
$ make
$ sudo make install
```

Повторение мать учения

Что такое ЭВМ (компьютер)?

Компьютер — это машина для обработки битов.

Бит — это отдельная единица компьютерного хранилища информации, которая может принимать два альтернативных значения — мы их обычно зовем 0 и 1.

Компьютеры используются для обработки информации, но вся информация при этом представляется в виде битов. В этом случае бит — наименьшая единица информации.

Интерпретация бит и их последовательностей может быть различной.

Наборы битов могут представлять символы, цифры или любую другую информацию. Люди интерпретируют эти биты как информацию, в то время как компьютеры просто манипулируют битами, которые представляют собой отображение состояний линий и выводов микросхем (уровни) на значения 0 и 1.

Состояние линий и выводов микросхем

H — High (высокий уровень)

L — Low (низкий уровень)

Обобщенная структура вычислительной системы

Абстрактная архитектура вычислительной системы

Уровень 3	**OSM** – Уровень машины операционной системы	**Операционная система**
Уровень 2	**ISA** – Уровень архитектуры набора инструкций	Машинный код
Уровень 1	**MA** – Уровень микроархитектуры	Микропрограммы или оборудование (процессор)

Каждый верхний уровень наследует некоторое подмножество команд нижнего и добавляет новые. На уровне MA и ISA команды различных систем существенно различаются.

Байты

Блок из n бит, которые являются *наименьшим адресуемым количеством информации* в памяти компьютера, называется «*байтом*».

Современные компьютеры получают доступ к памяти в виде 8-битных блоков.

Таким образом байтом обычно называется 8-битное количество бит.

Основная память компьютера — это массив байтов, каждый из которых имеет отдельный адрес памяти.

Первый адрес байта равен 0, а последний адрес зависит от используемого аппаратного и программного обеспечения.

Байты организуются в группы.

Интерпретация данных и типизация

Байт можно интерпретировать как двоичное число. Двоичное число 01010101 равно десятичному числу 85 (64 + 16 + 4 + 1).

Число 85 может быть частью большего числа в компьютере.

Число 85 может интерпретироваться как машинная инструкция, в этом случае компьютер помещает значение регистра **`rbp`** в стек времени выполнения.

Число 85 также можно интерпретировать как символ — заглавную букву «U».

Буква «U» может быть частью символьной строки в памяти.

Биты — квант информации (1 или 0);
Тетрада — группа из 4-х бит (0...15)
Байты — квант адресуемой памяти (обычно 8 бит — 0...255);
Слова — группа из 2^k байт (наследие 8086/88);
BigEndian — адресуется старший байт (коммуникации, не x86)
LittleEndian — адресуется младший байт (x86) (Не путать с MSB/LSB — Most/Least Significant Bit)

Hexadecimal	Decimal	Binary	Adress	BE	LE
0x6B	107	0101 0101	70ab 5200	6 B	6 B
0xAF	175	1010 1111	70ab 5201	A F	A F
0xA7FE	43 006	1010 0111 1111 1110	70ab 5202	A 7	F E
			70ab 5203	F E	A 7
0x2C03 A7FE	738 437 118	0010 1100 0000 0011 1010 0111 1111 1110	70ab 5204	2 C	F E
			70ab 5205	0 3	A 7
			70ab 5206	A 7	0 3
			70ab 5207	F E	2 C

MSB ← 10101111 → LSB

Представление символов

ASCII — American standard code for information interchange. (ISO 646)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Определяет коды для символов:

- десятичных цифр;
- латинского алфавита;
- национального алфавита (национальные варианты ASCII во второй половине таблицы);
- знаков препинания;
- управляющих символов.

UNICODE — Юникод

Стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира.

Стандарт состоит из двух основных частей:

1) универсального набора символов (UCS — Universal character set);

2) семейство кодировок (UTF — Unicode transformation format).

Универсальный набор символов (UCS) перечисляет допустимые по стандарту Юникод символы и присваивает каждому символу код в виде неотрицательного целого числа, записываемого обычно в шестнадцатеричной форме с префиксом **U+**, например, **U+040F**.

Семейство кодировок определяет способы преобразования кодов символов для передачи в потоке или в файле.

UTF8 — обеспечивает наибольшую компактность и обратную совместимость с 7-битной системой ASCII. Текст, состоящий только из символов с номерами меньше 128, при записи в UTF-8 превращается в обычный текст ASCII и может быть отображён любой программой, работающей с ASCII. Стандарт RFC 3629 и ISO/IEC 10646 Annex D.

UTF-16 — каждый символ записывается одним или двумя словами (суррогатная пара).

UTF-32 — способ представления Юникода, при котором каждый символ занимает ровно 4 байта.

В потоке данных младший байт UTF-16 может записываться либо перед старшим (UTF-16 little-endian, UTF-16LE), либо после старшего (UTF-16 big-endian, UTF-16BE).

Аналогично существует два варианта четырёхбайтной кодировки — UTF-32LE и UTF-32BE.

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F	0010	0011
Q	w	e	R	t	Y		§	Ԥ (UTF-8)		Ԥ (UTF-16BE)		Ԥ (UTF-16LE)		𝄞			
51	77	65	52	74	59	00	FD	D4	A4	05	24	24	05	F0	9D	84	9E

@0000 — строка ASCII символов «QWERTY» или 'Q','W','E','R','T','Y','\0'

— байт со значением 0x51 (81)

— двубайтное слово со значением 0x7751 (LE) или 0x5177 (BE)

— четырехбайтное слово со значением 0x52657751 (LE) или 0x51776552 (BE)

@0007 — символ «§» (параграф) или байт со значением 0xFD (253 или -3)

@0008 — символ юникода U+0524 в кодировке UTF-8 со значением 0xD4A4

@000A — символ юникода U+0524 в кодировке UTF-16BE со значением 0x0524

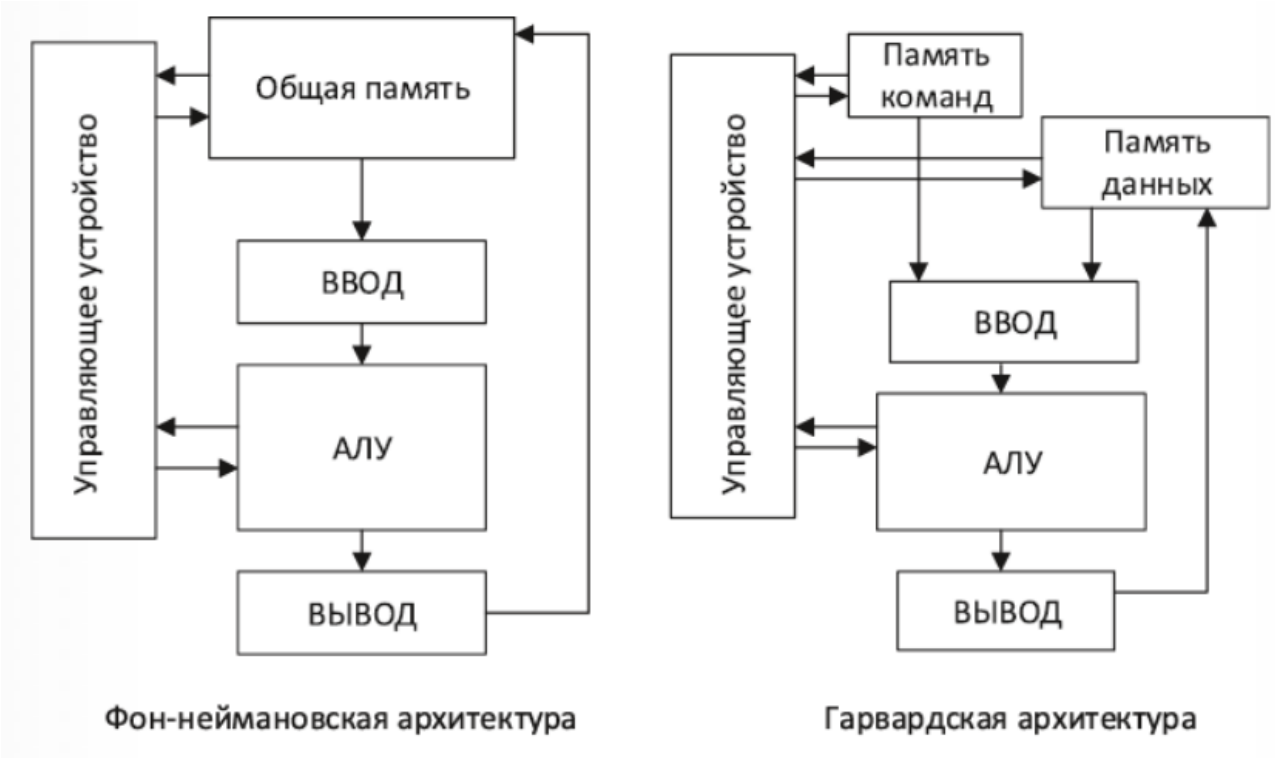
@000C — символ юникода U+0524 в кодировке UTF-16LE со значением 0x2405

@000E — символ юникода U+1D11E в кодировке UTF-8 со значением 0xF09D849E

Из чего состоит компьютер

- управляющее устройство (блок управления);
- арифметико-логическое устройство (АЛУ);
- память (общая для данных и команд/отдельно команды и отдельно данные);
- устройства ввода/вывода.

Управляющее устройство и АЛУ определяют действия, которые может выполнять ЭВМ, и составляют центральный процессор (ЦП, CPU).



Фон-неймановская архитектура

Гарвардская архитектура

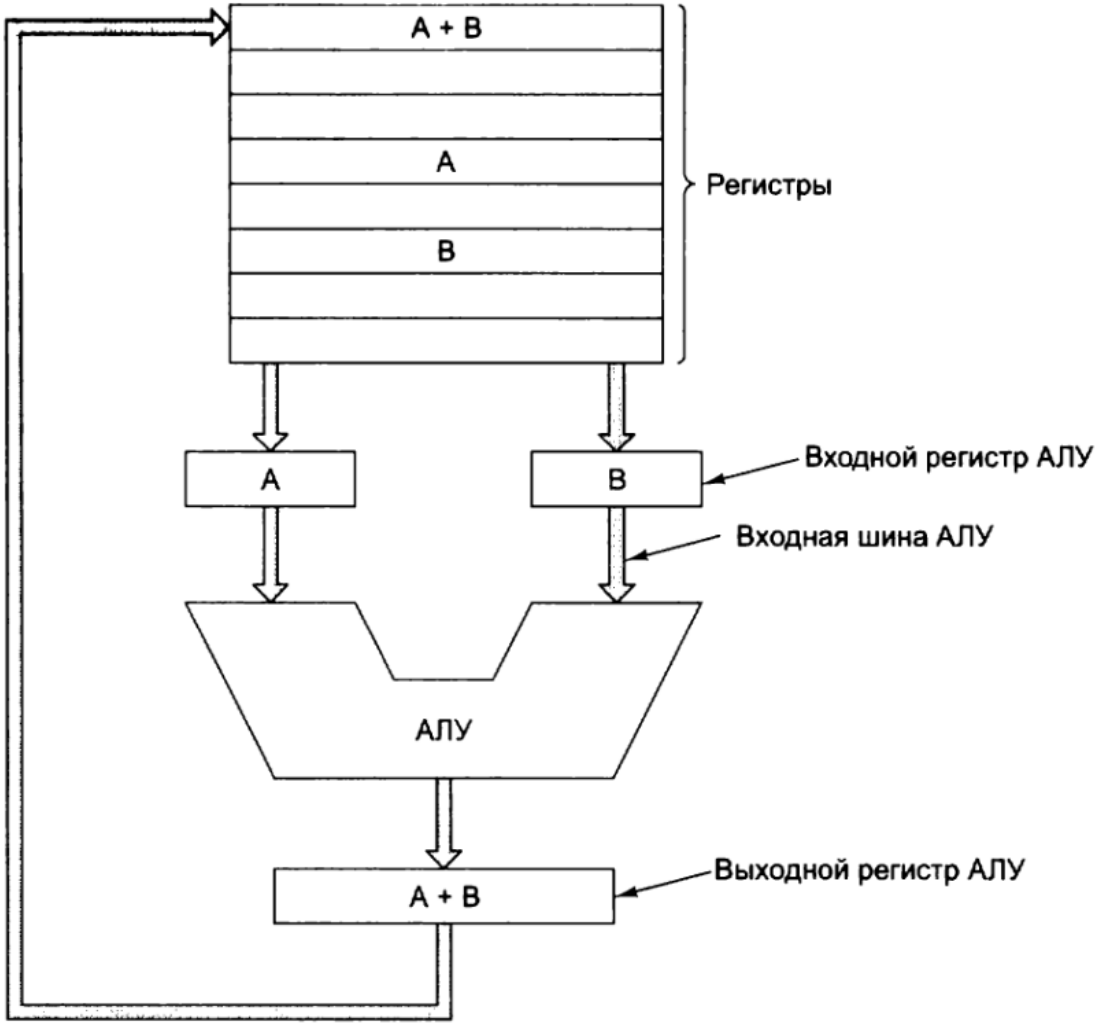
Центральный процессор

Реализуется на микропроцессоре семейства x86 (8086/88 … Intel Core i7/Xeon AMD FX). Для потоковой обработки SIMD используются расширения MMX, SSE, 3DNow!…

Процессор имеет набор регистров, часть из которых доступна для хранения операндов и выполнения над ними различных действий, включая арифметические, логические и адресные операции.

Часть регистров используется для служебных (системных) целей и доступ к ним, как правило, ограничен (программно-невидимые регистры).

Все компоненты ЭВМ представляются для процессора в виде наборов ячеек памяти и/или портов ввода/вывода, в которые процессор может записывать и/или считывать содержимое.



Цикл выполнения инструкции

Процессор исполняет инструкции, расположенные последовательно в памяти.

Существуют специальные инструкции условного и безусловного перехода, вызова и возврата из подпрограммы. Они позволяют прервать последовательную выборку инструкций из памяти и перейти к выполнению другой последовательности инструкций. Эти инструкции содержат адрес следующей инструкции.

Остальные инструкции предполагают последовательное выполнение, и поэтому не содержат адреса следующей инструкции.

Этот тип архитектуры называется архитектура, управляемая потоком команд.

Адрес очередной команды в памяти задается счетчиком адреса в блоке управления.

При выполнении команды, связанной с обращением к памяти, процессор должен выполнить как минимум пять операций, перечисленных ниже:

1. Выборка команды. Блок управления извлекает команду из памяти, копирует ее во внутреннюю память микропроцессора и увеличивает значение счетчика команд на длину этой команды.

2. Декодирование команды. Блок управления определяет тип выполняемой команды, пересылает указанные в ней операнды в АЛУ и генерирует электрические сигналы управления АЛУ, соответствующие типу выполняемой операции.

3. Выборка операндов. Если в команде используется операнд, расположенный в памяти, блок управления инициирует операцию по его выборке из памяти.

4. Выполнение команды. АЛУ выполняет указанную в команде операцию, сохраняет полученный результат в заданном месте и обновляет состояние флагов, по значению которых программа может судить о результате выполнения команды.

5. Запись результата в память. Если результат выполнения команды должен быть сохранен в памяти, блок управления инициирует операцию сохранения данных в памяти.

Память

Оперативная память (ОЗУ) — массив пронумерованных ячеек одинаковой информационной емкости (размера).

ОЗУ **физически** (на микросхеме) обычно организована в виде двумерного массива [бит] размерностью 2^{N-M} строк и 2^{M} столбцов, что позволяет очень просто реализовать нумерацию (физическую адресацию) 2^{N} ячеек в виде N-разрядного двоичного числа.

Для повышения производительности между памятью и процессором располагается сверхоперативная память небольшого объема (до трех уровней кеш-памяти).

Процессор, память и необходимые элементы взаимодействия их между собой и с другими устройствами называют центральной частью или ядром ЭВМ.

Представления расположений ячеек: