КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 01. Введение

Преподаватель:
Поденок Леонид Петрович
505а-5, + 375 17 293 8039
prep@lsi.bas-net.by

ftp://student@lsi.bas-net.by/

Кафедра ЭВМ, 2021

Оглавление

Взаимодействие и правила поведения на занятиях

Язык общения — русский
Фото- и видеосъемка запрещается, болтовня и прочее мычание тоже
Использование мобильных гаджетов может вызвать проблемы

prep@lsi.bas-net.by
ftp://student@lsi.bas-net.by/

Старосты групп отправляют на **prep@** со своего личного ящика сообщение, в котором указывают свой телефон и ящик, к которому имеют доступ все студенты группы. В этом же сообщении в виде вложения приводят списки своих групп.

Формат темы этого сообщения: 010901 Фамилия И.О. Список группы

Subj: [010901 Фамилия И.О. Суть сообщения]

Правила составления сообщений
- текстовый формат сообщений;

Удаляется на сервере присланное в ящик prep@ все, что:
- без темы;
- имеет тему не в формате;
- содержит html, xml и прочий мусор;
- содержит рекламу, в том числе и сигнатуры web-mail серверов;
- содержит ссылки на облака и прочие гуглопомойки вместо прямых вложений.

Введение

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 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)

Биты организуются в более сложные информационные структуры.

Самый простой — *тетрада* — группа из 4-х бит (0...15)

Стандарт ASN.1 — Abstract Syntax Notation One

Абстрактная синтаксическая нотация One, сокращенно ASN.1, является системой обозначений (нотацией) для описания абстрактных типов и значений.

Описывает, как биты организуются в более сложные информационные структуры.

Имеет отношение к независимости от платформы и языка программирования в сетевой среде.

Цель ASN.1 — дать возможность детализировать спецификации обмена информацией между взаимодействующими частями распределенного приложения, независимо от платформы и языка, используемого для реализации.

Начиная с 1995 года, существенно пересмотренный ASN.1 описывается стандартом X.680.

В России ASN.1 стандартизирован по:

ГОСТ Р ИСО/МЭК 8824-1-2001. Информационная технология. Абстрактная синтаксическая нотация версии один (АСН.1), Госстандарт России, М.: Изд. стандартов, 2001.

ГОСТ Р ИСО/МЭК 8825-93

Байты

Биты группируются в блоки по n бит, которые являются *наименьшим адресуемым количеством информации* в памяти компьютера. Каждое такое n-битное количество называется «*байтом*».

Байт должен вмещать любой из символов, которые используются в вычислительной системе.

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 (**`push rbp`**), 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/Lest 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
			70ab5 2056	A	7	0	3
			70ab 5207	F	E	2	C

MSB ← 10101111 → LSB

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

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

	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**[1].

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.

---

1 Кодовая точка

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F	0010	0011
Q	w	e	R	t	Y		§	Д (UTF-8)		Д (UTF-16BE)		Д (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

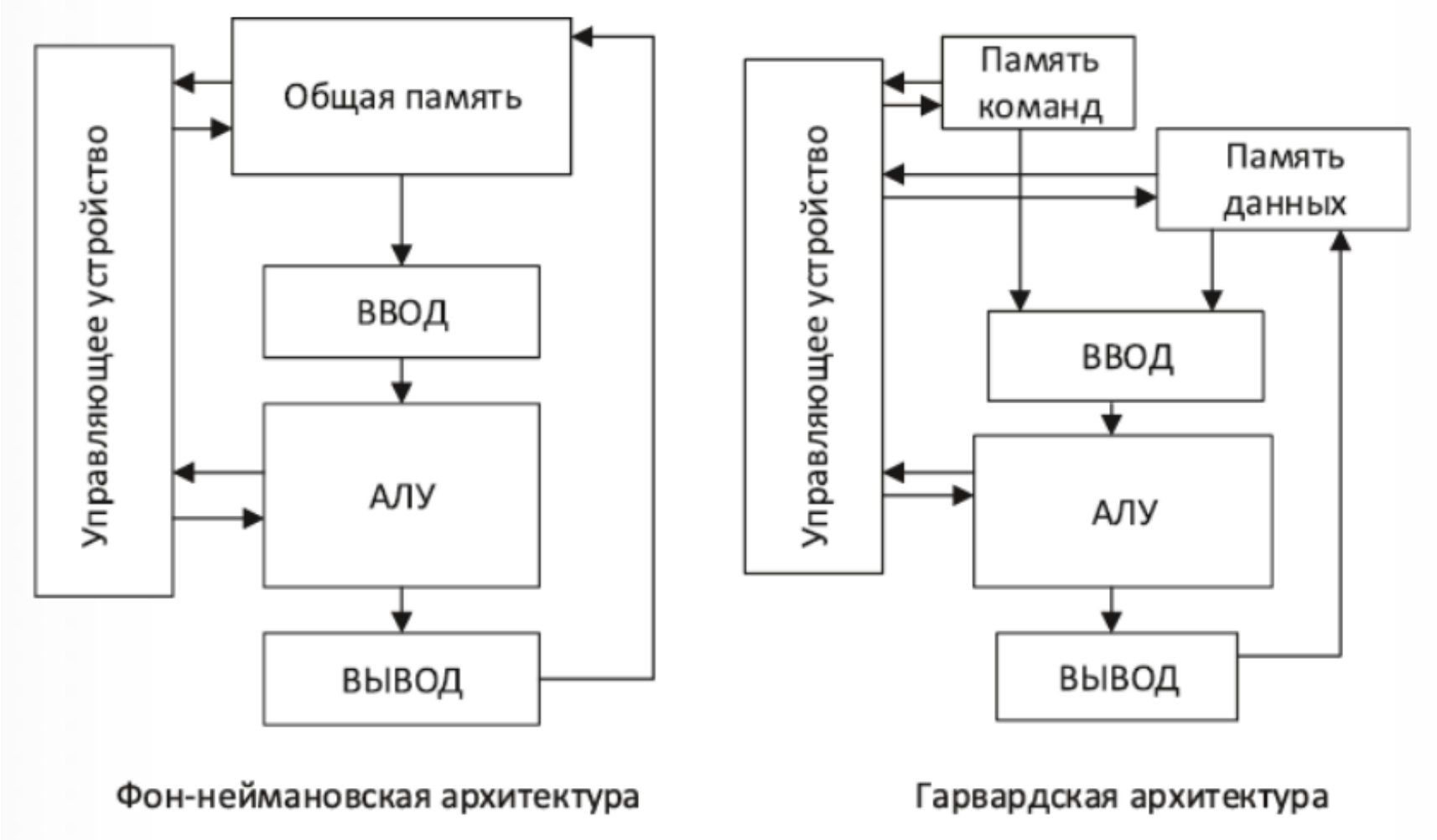
@000A — символ юникода U+0524 в кодировке UTF-16BE со значением 0x0524

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

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

Концептуальный состав ЭВМ

Любой IBM PC-совместимый компьютер представляет собой реализацию суперпозиции фон-Неймановской и гарвардской архитектур.

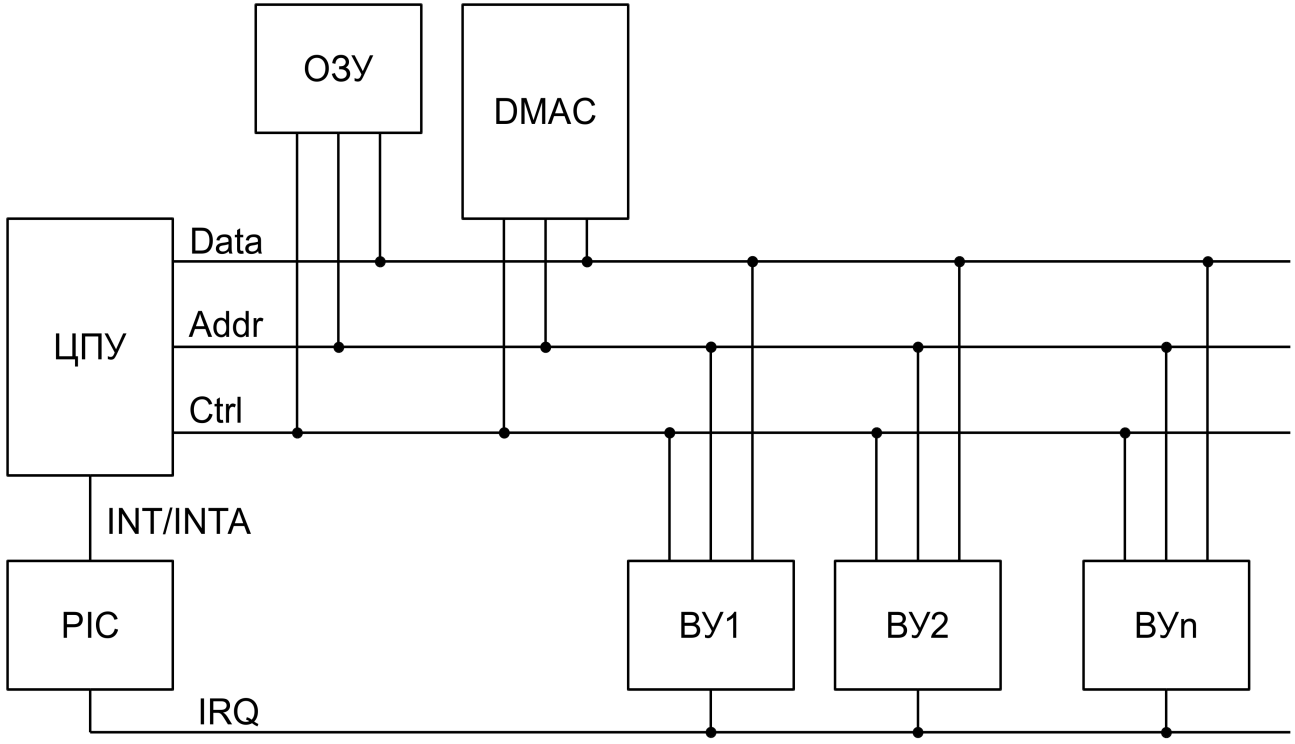


Из чего состоит

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

Данная архитектура реализует концепцию хранимой в памяти программы — программа и данные хранятся в памяти.

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



Блок управления

Процессор исполняет инструкции[2], расположенные последовательно в памяти.

Существуют специальные инструкции условного и безусловного перехода, инструкции вызова и возврата из подпрограммы. Они позволяют прервать последовательную выборку инструкций из памяти и перейти к выполнению другой последовательности инструкций. Эти инструкции содержат[3] 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[4] в блоке управления.

---

2 Иногда инструкции процессора называют командами процессора и наоборот.

3 Или позволяют определить

4 PC — Program Counter или IP — Instruction Pointer

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.

Чать 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 инструкции процессор должен выполнить как минимум три (черные) из операций, перечисленных ниже.

При выполнении команды, связанной с обращением к памяти, процессор должен выполнить как минимум четыре из операций, перечисленных ниже (хотя бы одна голубая):

1. Выборка команды. Блок управления извлекает команду из памяти, копирует ее во внутреннюю память микропроцессора и увеличивает значение счетчика команд на длину этой команды[5].

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.

5 Все, конечно, сложнее и особенно если есть обращение к памяти.

Память

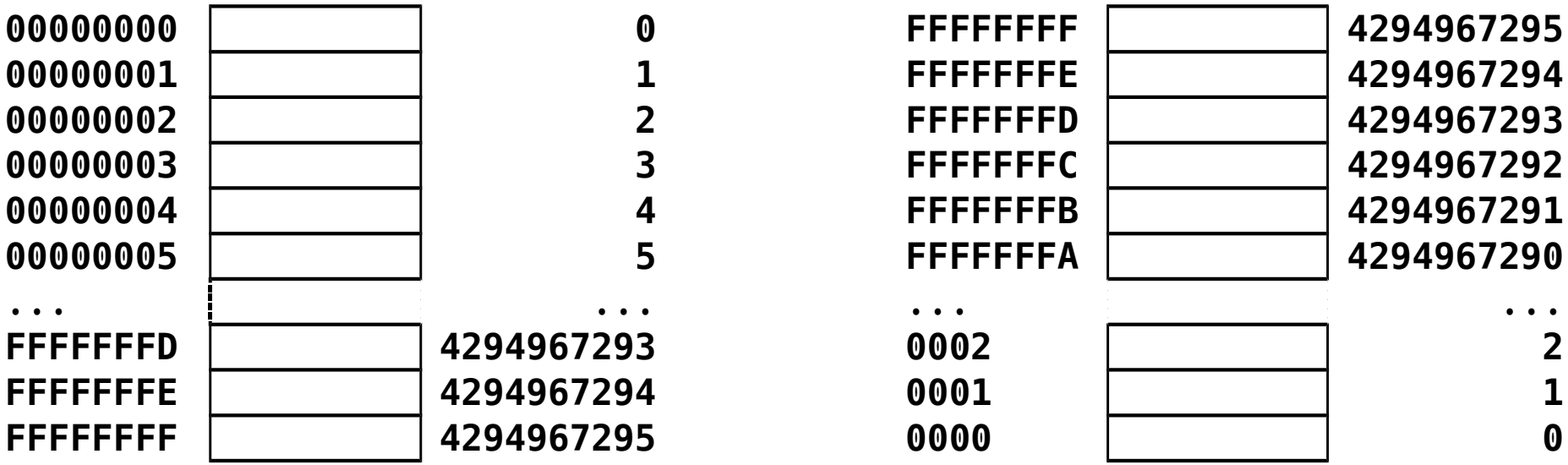
Оперативная память (ОЗУ) — массив пронумерованных ячеек одинаковой информационной емкости (размера).

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:



Периферийные устройства

Все остальные компоненты, не вошедшие в ядро. Их иногда называют устройствами ввода-вывода (УВВ), что не совсем точно.

- устройства хранения данных;
- коммуникационные устройства;
- устройства ввода-вывода.

Шины и интерфейсы

Процессор, память и периферийные устройства взаимодействуют между собой с помощью шин и интерфейсов, аппаратных и программных.

Стандартизация интерфейсов делает архитектуру компьютеров открытой.

Ячейки памяти — служат для хранения, поддерживают операции записи и чтения.
Порты — преобразование двоичной информации в электрические сигналы.
Регистры — широкое многоуровневое понятие.

Системная шина (System Bus)

Современная ВС организована вокруг системной шины (или нескольких).

Системная шина предназначена для организации обмена информацией между всеми компонентами компьютера. Все основные блоки персонального компьютера подсоединены к системной шине Основной функцией системной шины является обеспечение взаимодействия между центральным процессором и остальными электронными компонентами компьютера.

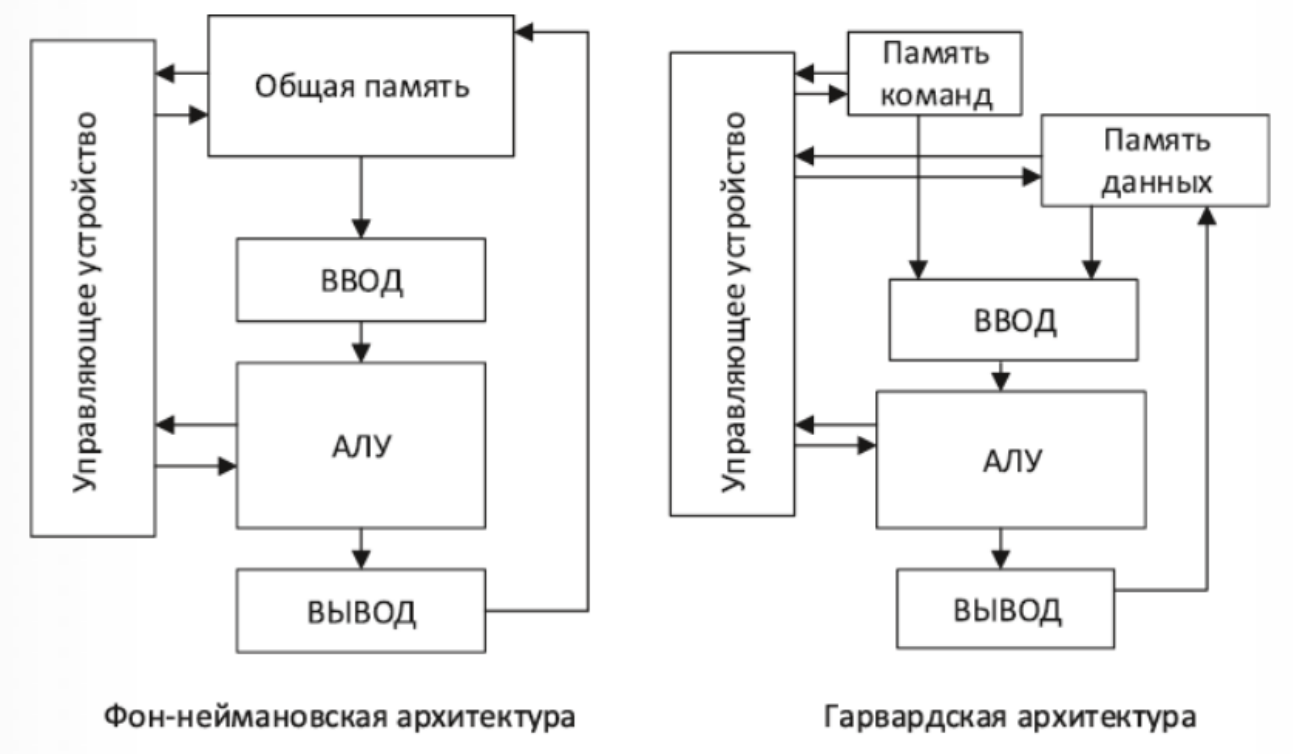
Шина представляет собой группу параллельных проводников, с помошью которых данные передаются от одного устройства компьютерной системы к другому. Обычно системная шина компьютера состоит из трех разных шин:

Шина данных (data bus) используется для обмена команд и данных между ЦПУ и оперативной памятью, а также между устройствами ввода-вывода и ОЗУ.

По шине управления (control bus) передаются специальные сигналы, синхронизирующие работу всех устройств, подключенных к системной шине.

Шина адреса (address bus) используется для указания адреса ячейки памяти в ОЗУ, к которой в текущий момент происходит обращение со стороны ЦПУ или устройств ввода-вывода.

Архитектура фон-Неймана. Принципы.



Принцип однородности памяти

Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования.

Это значит, что одно и то же значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости от способа обращения к нему.

Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей, например модификацию команд.

Принцип адресности

Структурно основная память состоит из пронумерованных ячеек, причём процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — адреса.

Принцип программного управления

Все вычисления, предусмотренные алгоритмом решения задачи, представлены в виде программы, состоящей из последовательности управляющих слов — команд. Каждая команда предписывает выполнение некоторой операции из реализуемого набора.

Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются в естественной последовательности, то есть в порядке их положения в программе.

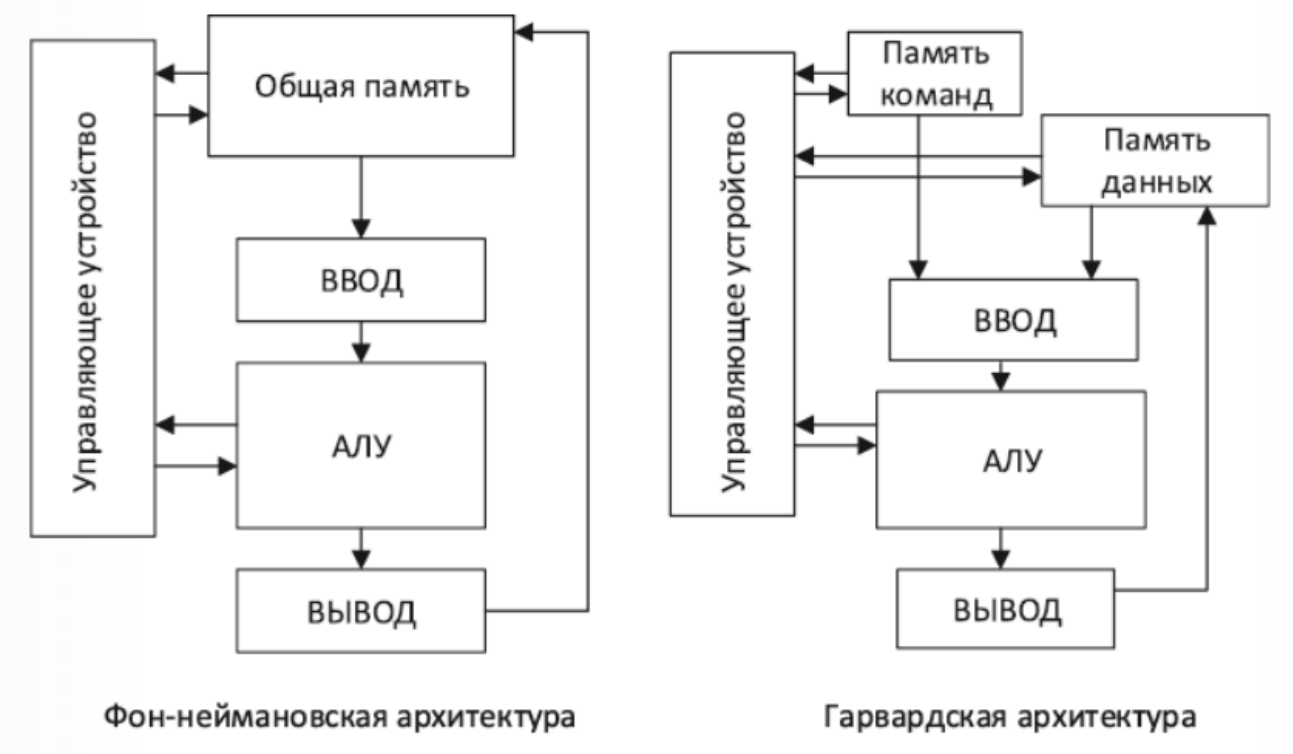
При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

Принцип двоичного кодирования

Вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1. Каждый тип информации представляется двоичной последовательностью и имеет свой формат (int, float …). В формате команды в простейшем случае можно выделить два поля — поле кода операции и поле адресов/операндов.

КОП	Операнды

Гарвардская архитектура



– хранилище инструкций и хранилище данных представляют собой разные физические устройства;

– канал инструкций и канал данных так же физически разделены.

В гарвардской архитектуре характеристики устройств памяти для инструкций и памяти для данных не обязательно должны быть одинаковыми.

Ширина слова, протокол доступа, технология реализации и структура адресов памяти могут различаться.

В некоторых системах инструкции могут храниться в памяти только для чтения (ПЗУ, ROM), в то время как для сохранения данных обычно требуется память с возможностью чтения и записи (ОЗУ, RAM).

В некоторых системах требуется значительно больше памяти для инструкций, чем для данных, что обуславливает различную ширину шин адреса памяти инструкций и данных.

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 (ВЭМ)
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.

Некоторые термины

Биты — квант информации (1 или 0)

Байты — квант адресуемой памяти

Слова — группа байт (наследие 8086/88)

BigEndian — адресуется старший байт (коммуникации, не x86)

LittleEndian — адресуется младший байт (x86) (Не путать с MSB/LSB — Most/Lest Significant Bit)

Ячейки памяти — служат для хранения, поддерживают операции записи и чтения.

Порты — преобразование двоичной информации в электрические сигналы.

Регистры — широкое многоуровневое понятие.

Программирование

Программирование — процесс создания компьютерных программ.

«Программы = алгоритмы + структуры данных» ((c) Никлаус Вирт).

Для программирования используются [формальные] языки программирования, с помощью которых записываются исходные тексты программ.

Иерархия языков программирования:

- машинные коды;
- автокоды;
- языки ассемблера;
- высокоуровневые языки.

Парадигма программирования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это набор способов организации вычислений и структурирования работы, выполняемой компьютером.

императивная;

декларативная;

метапрограммирование;

....

Практически все современные языки программирования в той или иной мере допускают использование различных парадигм — **мультипарадигмальное программирование**.

Общие понятия языков программирования

Программы строятся из структур данных и алгоритмов (Вирт Н.)

Алгоритм

Алгоритм — конечная последовательность предписаний, исполнение которых позволяет получить решение некоторой задачи.

Представляет собой процедурный элемент в структуре программы и является рецептом вычислений и/или других манипуляций с данными. Бывают численные и нечисленные. Численные оперируют с числами (арифметические операции), нечисленные — с данными, которые необязательно являются числами (поиск и сортировка).

Данные

Данные — информация, представленная в виде, пригодном для обработки автоматическими средствами при возможном участии человека.

Идентификатор объекта

Идентификатор, ID (data name, identifier — опознаватель) — элемент данных, однозначно определяющий объект внутри системы. Это уникальный признак объекта, позволяющий отличать его от других объектов и ссылаться на него.

Структура

Структура — совокупность связей между компонентами системы, объекта, программы, обеспечивающая их целостность.

Структура данных

Структура данных — множество элементов данных, упорядоченных (организованных) каким-либо из допустимых способов. Относится к «пространственным» понятиям — ее можно свести к схеме организации данных в памяти компьютера.

Структуры данных, применяемые в алгоритмах, могут быть чрезвычайно сложными, вследствие чего выбор правильного представления данных является ключевым фактором удачного программирования и может в большей степени повлиять на производительность, нежели детали самого алгоритма.

Общей теории выбора структур данных не существует и вряд ли появится когда либо. В действительности это более инженерное мастерство, нежели строгий набор правил и рецептов.

Физическая структура данных отражает способ физического представления данных в памяти ЭВМ (структура хранения, внутренняя структура, структура памяти) и зависит от архитектуры.

Логическая (абстрактная) структура данных не учитывает представление в памяти ЭВМ и не зависит от деталей ее архитектуры.

В ряде случаев для отображение логической структуры в физическую и наоборот используются специальные процедуры (сериализация данных при выгрузке на диск или для передачи по сети).

Переменная

Ссылка на элемент данных.

Переменная в императивном программировании — поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются *значением этой переменной*.

В других парадигмах программирования, например, в функциональной и логической, понятие переменной может быть несколько иным. В таких языках переменная определяется как имя (идентификатор), с которым может быть связано значение, или как место (location) для хранения значения. Переменной может быть присвоено некоторое значение с помощью операторов.

Константа

Константа (constant) — способ адресации/идентификации/именования данных, изменение которых рассматриваемой программой не предполагается или запрещается.

Константы сохраняют постоянное значение в той части программы, где они определены.

Константы могут быть именованные (имеют ID) и неименованные — литералы (literal).

Тип данных

Тип данных определяет возможные значения и их смысл (семантику), операции, а также способы хранения значений типа. Неотъемлемой частью большинства языков программирования являются системы типов, использующие типы для обеспечения той или иной степени *безопасность типов*.

Тип возникает/появляется из того факта, что имя переменной или константы имеет смысл только для программиста, а компьютеру ничего не говорит. Для правильного связывания идентификатора с определенным адресом памяти компилятор нуждается в дополнительной информации, используя которую он либо выполнит надлежащее связывание, либо сообщит о невозможности это сделать.

Безопасность типов (type safety)

Безопасность типов языка программирования означает безопасность (или надёжность) его системы типов.

Ошибка согласования типов или ошибка типизации (type error) представляет собой несогласованность типов компонентов выражений в программе, например попытку использовать целое число в роли функции. Пропущенные ошибки согласования типов на этапе выполнения могут приводить к ошибкам и крахам программ. В зависимости от назначения ЯП защита типов на этапе компиляции может быть более или менее жесткой.

PASCAL, задуманный, как инструмент для иллюстрирования структур и алгоритмов данных, или ADA, язык для проектирования сверхнадежных систем реального времени, расценивают смешение в одном выражении данных разных типов, или применение к типу несвойственных для него операций, как фатальную ошибку.

Язык С, задуманный, как инструмент для системного программирования, является языком со *слабой защитой типов*. Тем не менее, о нарушениях такого рода компилятор выдает предупреждения, если это не отменяется явно или не предусмотрено обратное. Это дает программисту широчайшие возможности для манипулирования данными, однако под его ответсвенность.

Выражение

Выражение (expression) — комбинация значений, констант, переменных, операций и функций, которая интерпретируется в соответствии с правилами конкретного языка. Интерпретация (выполнение) такого выражения приводит к ***вычислению некоторого значения*** (например, числа, строки или значения логического типа).

Например, 2+3 — выражение, которое имеет значением после его вычисления число 5. Отдельная константа или отдельная переменная также является выражением, значением которого является, соответственно, сама константа или значение переменной. Во многих языках программирования выражения, содержащие вызовы функций, могут иметь побочные эффекты.

Операция

Операция — конструкция языка программирования, специальный способ записи некоторых действий. Наиболее часто применяются арифметические, логические и строковые операции.

В отличие от функций, операции часто являются базовыми элементами языка и обозначаются различными символами пунктуации, а не алфавитно-цифровыми. Терминология и содержание этого понятия отличается от языка к языку.

Операции делятся по количеству принимаемых аргументов на:

унарные — один аргумент (отрицание, унарный минус);

бинарные — два аргумента (сложение, вычитание, умножение и т. д.);

тернарные — три аргумента («условие ? выражение1 : выражение2»).

Подпрограмма, процедура, функция

Подпрограмма (subroutine) — фрагмент программного кода, к которому можно обратиться из другого места программы.

Если подпрограмма возвращает значение и не меняет состояния переменных программы, она называется ***функцией***.

Если подпрограмма не возвращает значения (возвращает пустое значение), но меняет состояния переменных программы, она называется ***процедурой***.

Многие языки допускают безымянные функции (лямбда-функции).

С именем функции/процедуры связан адрес первой инструкции, входящей в функцию, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается обратно в точку программы, где данная функция была вызвана (адрес возврата).

В некоторых языках программирования объявления функций и процедур имеют различный синтаксис, в частности, могут использоваться различные ключевые слова (fortran).

В объектно-ориентированном программировании функции, объявления которых являются неотъемлемой частью определения класса, называются методами.

Для языка C++, несмотря на то, что он имеет элементы ООП, понятие «метод» не используется.

Парадигмы программирования

Императивная парадигма

Императивное программирование — стиль написания исходного кода компьютерной программы, для которого характерно следующее:

– в исходном коде программы записываются инструкции (команды);
– инструкции должны выполняться последовательно;
– данные, полученные при выполнении инструкции, могут записываться в память.
– данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;

Императивная программа похожа на приказы, выражаемые повелительным наклонением в естественных языках, и представляет собой последовательность команд, которые должен выполнить компьютер.

При императивном подходе к составлению кода широко используется **присваивание**.

Основные черты императивных языков:

– использование именованных переменных;
– использование оператора присваивания;
– использование составных выражений;
– использование подпрограмм.

Процедурная

Программа разбивается на структурные единицы, которые могут называться по-разному — процедуры, функции, подпрограммы и т. п.

Подпрограммы могут вызывать другие подпрограммы, передавая им параметры и получая обратно результат выполнения.

Важно, что у подпрограммы одна точка входа, то есть все подпрограммы, вызывающие данную, переходят на один и тот же адрес точки входа.

Как правило, подпрограммы имеют одну точку выхода, возвращающую управление в точку вызова, но это менее существенно, чем требование одной точки входа для каждой подпрограммы. Такой код обычно получается в результате компиляции программы.

Процедурное программирование является отражением архитектуры традиционных ЭВМ, которая была предложена Фон Нейманом в 1940-х годах.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, при этом программа последовательно обновляет содержимое памяти.

ADA, C, C++, Fortran, Cobol, ...

Структурная

Парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков.

Возможность структурной организации программ математически обоснована.

В соответствии с парадигмой, любая программа, которая строится без использования оператора **goto**, состоит из трёх базовых управляющих структур:

- последовательность;
- ветвление;
- цикл.

кроме этого, используются подпрограммы.

Последовательность — однократное выполнение операций в том порядке, в котором они записаны в тексте программы.

Ветвление — однократное выполнение одной из двух или более операций, в зависимости от выполнения заданного условия.

Цикл — многократное исполнение одной и той же операции до тех пор, пока выполняется заданное условие (условие продолжения цикла).

Основные принципы структурного программирования:

Принцип 1. Оператор безусловного перехода **goto** не используется.

Принцип 2. Любая программа строится из трёх базовых управляющих конструкций:

- последовательность;
- ветвление;
- цикл.

Принцип 3. В программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом. Никаких других средств управления последовательностью выполнения операций не предусматривается.

Принцип 4. Повторяющиеся фрагменты программы можно оформить в виде подпрограмм (процедур и/или функций).

В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция «Вызов подпрограммы». При выполнении такой инструкции работает вызванная подпрограмма. После этого продолжается исполнение основной программы, начиная с инструкции, следующей за командой «Вызов подпрограммы».

Таким же образом в виде подпрограмм можно оформить логически целостные фрагменты программы, даже если они не повторяются.

Принцип 5. Каждую логически законченную группу инструкций следует оформить как блок. Блоки являются основой структурного программирования.

Блок — это логически сгруппированная часть исходного кода, например, набор инструкций, записанных подряд в исходном коде программы.

Понятие блок означает, что к блоку инструкций следует обращаться как к единой инструкции. Блоки служат для ограничения области видимости переменных и функций. Блоки могут быть пустыми или вложенными один в другой. Границы блока строго определены.

Например, в **`if`**-инструкции блок ограничен кодом **`BEGIN..END`** (в языке Pascal) или фигурными скобками **`{…}`** (в языке C) или отступами (Python).

Принцип 6. Все перечисленные конструкции должны иметь один вход и один выход.

Принцип 7. Разработка программы ведётся пошагово, методом «сверху вниз».

Объектно-ориентированная

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

ООП — подход к программированию как к моделированию информационных объектов, позволяющий в ряде случаев упростить управление данными и упорядочить их организацию при реализации крупных проектов.

Основные принципы ООП

- класс;
- объект;
- абстракция данных;
- инкапсуляция;
- наследование;
- полиморфизм подтипов.

Класс — универсальный, комплексный тип данных, состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями). Класс является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей).

Объект — сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных, подразумевая под этим набор наиболее значимых характеристик объекта, котороые доступны остальной программе.

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в виде единой сущности — класса.

Язык C++, например, отождествляет инкапсуляцию с сокрытием, но другие, например, Smalltalk, различают эти понятия.

Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

Полиморфизм в языках программирования и теории типов — способность функции обрабатывать данные разных типов.

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. В ООП под термином «полиморфизм» обычно подразумевают наследование.

Другой вид полиморфизма — параметрический — в ООП называют обобщённым программированием.

Ключевая проблема больших программных проектов

Отличительной особенностью объектно-ориентированного кода является то, что он может сделать маленькие и простые задачи похожими на большие и сложные.

ОО концепция — нечеткая концепция, которая имеет как хорошие, так и плохие стороны. Прежде всего, он часто реализуется неполным, неполноценным способом (C++, Java) и это подрывает ее полезность.

Чтобы воспользоваться преимуществами ОО-программирования, язык должен обеспечить ряд условий:

- распределение всех переменных в куче;
- обеспечить доступность сопрограмм;
- правильную реализацию обработки исключений;
- сборку мусора.

Все это очень дорого с точки зрения реализации и исполнения.

Тем не менее, ОО-модель включает в себя несколько хороших идей:

1) Идея равноценности между процедурами и структурами данных.

Эта идея эквивалентности является очень элегантной, очень инновационной в то время концепцией, которая первоначально появилась в языках моделирования и вошла в языки программирования общего назначения под влиянием Simula 67.

2) Иерархическое разбиение пространства имен переменных на деревья с «наследованием» — очень аккуратный метод доступа к данным более низкого уровня из абстракций более высокого уровня. Чем-то это напоминает мне структурирование файловой системы Unix. Это то, что делает создание больших, сложных систем проще, или даже просто возможным. Без пространств имен разработка больших программ требует железной дисциплины именования переменных, на которую в прошлом была способна только IBM. Но это не единственный способ реализовать пространства имен. Существуют и неиерархические пространства имен и они также очень полезны (общий блок в Фортране).

3) Использование кучи вместо стека для размещения переменных.

На языке общего назначения эта концепция впервые была реализована в PL/1 в начале 60-х годов. Так что этой технике 50 лет и до PL/1 от LISP. Тем не менее доминирование C, который был языком системного программирования, сильно повлияло на приемлемость этой идеи, существенно замедлив ее широкое распространение в течение приблизительно 12 лет (Java был создан в 1995 году, в то время как C был создан примерно между концом 60-х годов и 1973 годом).

Java был первым широко используемым языком с распределением переменных в куче. Также важную роль сыграли ограничения раннего оборудования.

С был задуман как язык системного программирования. Но позже из-за доступности компиляторов с открытым исходным кодом область его применения была расширена для разработки приложений, несмотря на отсутствие автоматического распределения памяти и проверки индексов, что при неаккуратном программировании приводит к уязвимости переполнения буфера. Последнее стало огромной проблемой безопасности, от которой мы до сих пор страдаем. Лучшие, более безопасные языки стали возможны где-то с 1990 года благодаря огромному прогрессу в аппаратном обеспечении. Для разработки приложений стало возможным использовать лучшие, более безопасные языки, а C++ стал попыткой исправить некоторые недостатки C, как языка разработки приложений, заимствуя некоторые идеи из Simula67.

Позже Java ввела распределение переменных в куче и сборку мусора в мейнстрим. Это, скорее всего, объясняет, почему Java, несмотря на то, что она была крайне плохо спроектирована, смогла обогнать Cobol.

4) Естественность группирования связанных процедур, работающих на одних и тех же данных, под «одной крышей». Это было впервые разработано в PL/1 на основе многовходовых процедур. Данная идея также связана с общей равноценностью процедур и структур данных.

5) Идея конструктора — это, по сути, идея инициализации такой динамически распределенной структуры данных, с которой работает такая процедура с несколькими точками входа. Все вот так просто, так что все эти разговоры о классах как об абстракциях объектов с множеством операций над ними — просто дымовая завеса, скрывающая простоту идеи.

Распространение идеи типа на структуры и предоставление механизма создания нового экземпляра структуры с автоматическим заполнением необходимых элементов с помощью специальной процедуры инициализации, называемой конструктором. Источником вдохновения была Simula-67, но еще до Simula-67, в PL/1 была изобретена концепция создания похожей структуры из базовой с помощью оператора like.

В ОО эта идея получила новый соус и новую терминологию, но, по мнению автора компилятора (Б. Страуструп), это та же идея.

Самая большая ловушка — это вера в то, что ООП — это серебряная пуля или «одна идеальная парадигма».

Декларативная парадигма

Декларативное программирование — это парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается, что представляет собой проблема и каков должен быть результат.

Декларативные программы не используют понятия состояния, то есть не содержат переменных и операторов присваивания.

SQL — «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Функциональная

Функциональное программирование — раздел дискретной математики и парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в их математическом понимании, а не как функций- подпрограмм в процедурном программировании.

Основной особенностью функционального программирования является то, что в ней реализуется **модель вычислений без состояний**. Если императивная программа на любом этапе исполнения имеет состояние (в виде совокупности значений всех переменных) и производит побочные эффекты, то чисто функциональная программа ни целиком, ни ее части состояния не имеет и побочных эффектов не производит.

То, что в императивных языках делается путём присваивания значений переменным, в функциональных достигается путём передачи выражений в параметры функций. Непосредственным следствием становится то, что чисто функциональная программа не может изменять уже имеющиеся у неё данные, а может лишь порождать новые путём копирования и/или расширения старых.

Основные понятия функциональной парадигмы — функции высших порядков, чистые функции и рекурсия.

Функции высших порядков — это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции. Математики такую функцию чаще называют оператором, например, оператор взятия производной или оператор интегрирования.

Чистыми называют функции, которые не имеют побочных эффектов ввода-вывода и памяти (они зависят только от своих параметров и возвращают только свой результат).

Рекурсия — в функциональных языках в виде рекурсии обычно реализуется цикл.

Рекурсивные функции вызывают сами себя, позволяя операции выполняться снова и снова.

Мультипарадигмальное программирование

Парадигма программирования не определяется однозначно языком программирования — практически все современные языки программирования в той или иной мере допускают использование различных парадигм.

В языке Си отсутствуют встроенные встроенных средств поддержки ООП, в связи с чем он не является объектно-ориентированным, однако реализация объектно-ориентированного подхода на языке возможна.

В качестве наиболее ярких примеров можно привести библиотеки OpenGL, OpenCL, OpenMAX AL и т. п., которые реализуют именно ООП средствами языка С.

Функциональное программирование можно применять при работе на любом императивном языке, в котором имеются функции.

Существующие парадигмы пересекаются друг с другом в деталях (например, модульное и объектно-ориентированное программирование).

Детальное рассмотрение особенностей языков программирования
https://ru.wikipedia.org/wiki/
Список_языков_программирования_по_категориям
Сравнение_языков_программирования

Представлены таблицы встроенных возможностей ЯП (в русской версии отсутствуют Pascal, Fortran).

Существуют неформальные метрики ЯП:

1) Стоимость исполнения. Как быстро работает программа?

2) Стоимость компиляции. Сколько времени нужно, чтобы перейти от исходного кода к исполняемому?

3) Стоимость мелкомасштабной разработки. Каковы затраты рабочего времени индивидуального программиста?

4) Стоимость крупномасштабной разработки. Каковы затраты рабочего времени команды программистов?

5) Стоимость языковых особенностей. Насколько сложно выучить или использовать язык программирования, чтобы достичь определенного уровня навыков (кривая обучения)?

Жизненный путь программы

Обычно программа проходит нескольких стадий:

— текст на алгоритмическом языке;
— объектный модуль;
— загрузочный модуль;
— бинарный образ в памяти.

Трансляция программы — преобразование программы, представленной на одном из языков программирования, в программу на другом языке. Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Виды трансляции:

- компиляция;
- интерпретация;
- динамическая компиляция.

Компилятор (compiler) — транслятор, преобразующий исходный код с какого-либо языка программирования на машинный язык.

Процесс компиляции, как правило, состоит из нескольких этапов:

- лексический анализ;
- синтаксический анализ;
- семантический анализ;
- создание на основе результатов анализов промежуточного кода;
- оптимизация промежуточного кода;
- создание объектного кода, в данном случае машинного.

Используемые программой адреса в каждом конкретном случае могут быть представлены различными способами. Например, адреса в исходных текстах обычно символические.

Компилятор связывает эти символические адреса с перемещаемыми адресами (такими как N байт от начала модуля).

Загрузчик или компоновщик (linker), в свою очередь, связывают эти перемещаемые адреса с виртуальными/физическим адресами и создают исполняемый файл.

Каждое связывание — отображение одного адресного пространства в другое.

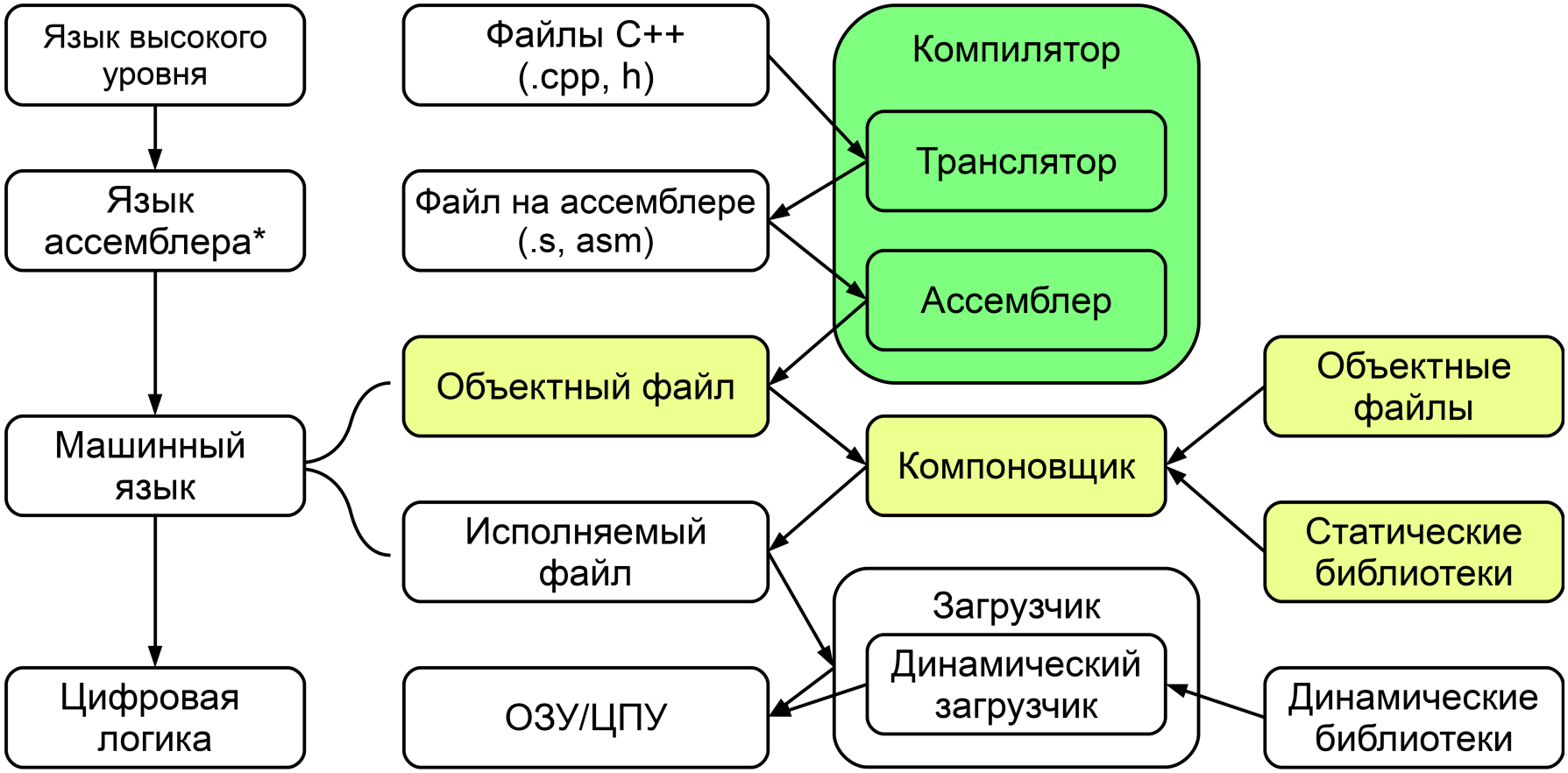
Привязка инструкций и данных к памяти (настройка адресов) в принципе может быть сделана на следующих шагах:

этап компиляции (Compile time). Когда на стадии компиляции известно точное место размещения процесса в памяти, тогда генерируются абсолютные адреса. Если стартовый адрес программы меняется, необходимо перекомпилировать код. В качестве примера можно привести **`.com`** программы MS-DOS, которые связывают ее с физическими адресами на стадии компиляции.

этап загрузки (Load time). Если на стадии компиляции не известно где процесс будет размещен в памяти, компилятор генерирует перемещаемый код. В этом случае окончательное связывание откладывается до момента загрузки. Если стартовый адрес меняется, нужно всего лишь перезагрузить код с учетом измененной величины.

этап выполнения (Execution time). Если процесс может быть перемещен во время выполнения из одного сегмента памяти в другой, связывание откладывается до времени выполнения. Здесь желательно специализированное оборудование, например регистры перемещения. Их значение прибавляется к каждому адресу, сгенерированному процессом. Например, x86 использует четыре таких (сегментных) регистра.

Связь между файлами программы и утилитами



make — утилита, отслеживающая изменения в файлах и вызывающая необходимые программы из набора, использующегося для компиляции и генерации выполняемого кода из исходных текстов (toolchain).