Языки программирования

3. Влияние архитектуры

Структура компьютера

1. Данные;
2. Элементарные операции;
3. Управление последовательностью действий;
4. Доступ к данным;
5. Управление памятью;
6. Операционная среда.

Данные

Хранение:

- *Оперативная память;*
- *Быстрые регистры;*
- *Внешние файлы.*

Встроенные типы данных:

- *Целое число;*
- *Вещественное число одинарной точности;*
- *Символьные строки фиксированной длины;*
- *Строки битов фиксированной длины;*
- *Программа на машинном языке.*

Операции

- Арифметические действия (+, -, *, /) для целочисленных и вещественных чисел;
- Проверка различных свойств элементов данных;
- Доступ к различным частям элементов данных и их модификация;
- Управление устройствами ввода-вывода данных;
- Управление последовательностью выполнения команд.

Управление последовательностью действий

При выполнении программы на машинном языке адрес очередной команды определяется регистром программных адресов (счетчиком команд).

Этот регистр используется только интерпретатором, который и управляет последовательностью выполнения операций.

Алгоритм интерпретации

Доступ к данным

Операнды хранятся в ОП или в рабочем регистре.

Областям ОП ставятся в соответствие некоторые целочисленные значения, которые играют роль адресов этих областей, а специальные операции извлекают (или сохраняют) по заданному адресу содержимое соответствующих областей памяти.

Аналогично регистры часто задаются простым целочисленным адресом.

Управление памятью

Цель: *сведение к минимуму времени бездействия каждой из его составных частей (ЦП, ОП, внешних устройств).*

Подходы:

- *Хранение выполняемой программы и ее данных в ОП;*
- *Мультипрограммирование и страничная организация памяти;*
- *Кэш – промежуточное звено между ОП и ЦП.*

Операционная среда

- Внешние запоминающие устройства;
- Устройства ввода-вывода.

Аппаратные различия обусловлены назначением устройств и скоростью доступа к ним.

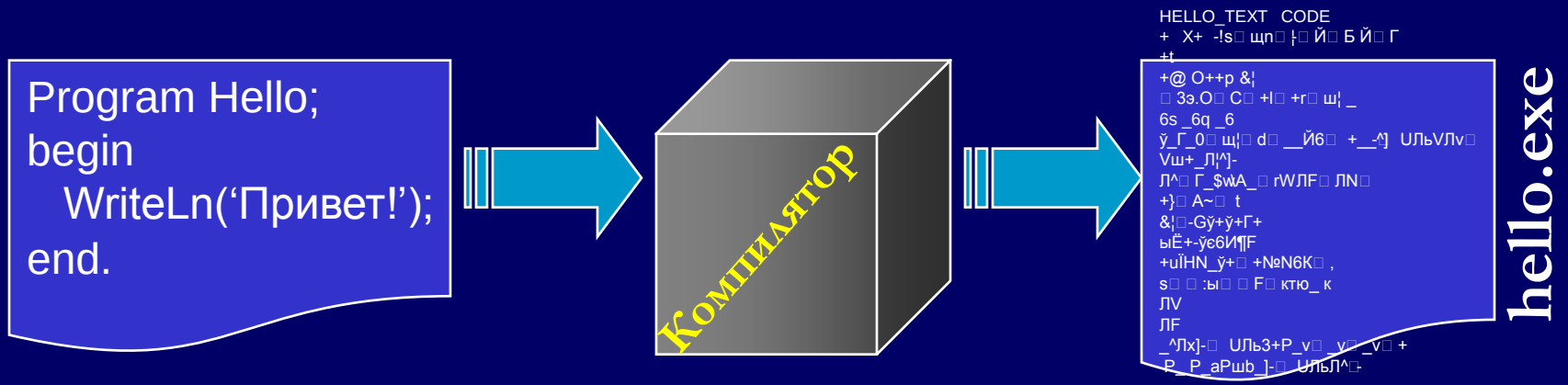
Состояние компьютера

Каждое *состояние компьютера* характеризуется содержимым ОП, регистров и внешней памяти в определенные моменты времени в процессе выполнения программы.

Выполнение программы можно рассматривать как процесс последовательной смены состояний компьютера.

Языковой процессор

- *Компьютер* выполняет программы на языке машинных команд. *Программист* создает программы на языке программирования.
- *Языковый процессор* обеспечивает выполнение на компьютере директив и предложений программы, написанной программистом.



Типы языковых процессоров

- *Транслятор* – *переводит* программу на исходном языке на некоторый объектный язык.
  - *Ассемблер* – транслятор с языка низкого уровня на язык машинных команд.
  - *Компилятор* – транслятор языка высокого уровня на язык машинных команд или ассемблера.
  - *Загрузчик (редактор связей)* – транслятор, у которого исходный языком является переменный машинный код (почти идентичен объектному).
  - *Препроцессор* – транслятор, исходный язык которого является расширенной формой какого-либо ЯВУ, а объектный язык – стандартная версия этого языка.

Трансляция программы на C++

1. Программа на C++ транслируется в программу на C;
2. Программа на C компилируется в программу на языке ассемблера;
3. Программа на языке ассемблера транслируется в переменный машинный код;
4. Редактором связей программа преобразуется в выполняемый машинный код, который может быть выполнен.

Типы языковых процессоров

- *Интерпретатор* – *выполняет* программу на исходном языке.

Выполнение возможно благодаря моделированию программно-аппаратного компьютера, для которого машинным языком является исходный язык программирования.

Типы языковых процессоров

- Транслятор обрабатывает операторы, входящие в программу, в порядке их фактического ввода, а интерпретатор следует логике управления выполняемой программы.
- Обычно транслятор обрабатывает каждый оператор только один раз, в то время как интерпретатор может обрабатывать некоторые операторы многократно (тело цикла) или полностью игнорировать другие.

Структура типичной реализации

Достоинства и недостатки

Трансляция:

- Выполнение многократно выполняемых операторов эффективнее.
- Теряется информация о том, какой из операторов программы выполняется и какие объекты данных используются.
- Размер программы значительно возрастает.

Интерпретация:

- Декодирование команд при каждом из многократном выполнении.
- Размер программы сохраняется.
- Выполнение программы замедляется за счет интерпретации кода.
- Сложность в реализации программного обеспечения процесса интерпретации.

Компиляторы как класс ПО

- Многочисленные реализации известных языков высокого уровня.
- Разработка новых языков высокого уровня требует разработки компиляторов.
- Разработка новых аппаратных архитектур требует разработки новых компиляторов для известных языков высокого уровня.

Языки программирования

- Компилируемые: C, C++, Fortran, Pascal, Ada.
- Интерпретируемые: LISP, ML, Perl, Postscript, Prolog, Smalltalk.
- С промежуточным байт-кодом (*псевдокод* – машинно-независимый код низкого уровня): Java, C#.

Виртуальный компьютер

- Конструируется на основе возможностей аппаратной части и программных элементов, предоставляемых базовым компьютером.
- Представляет собой интерпретацию языка.

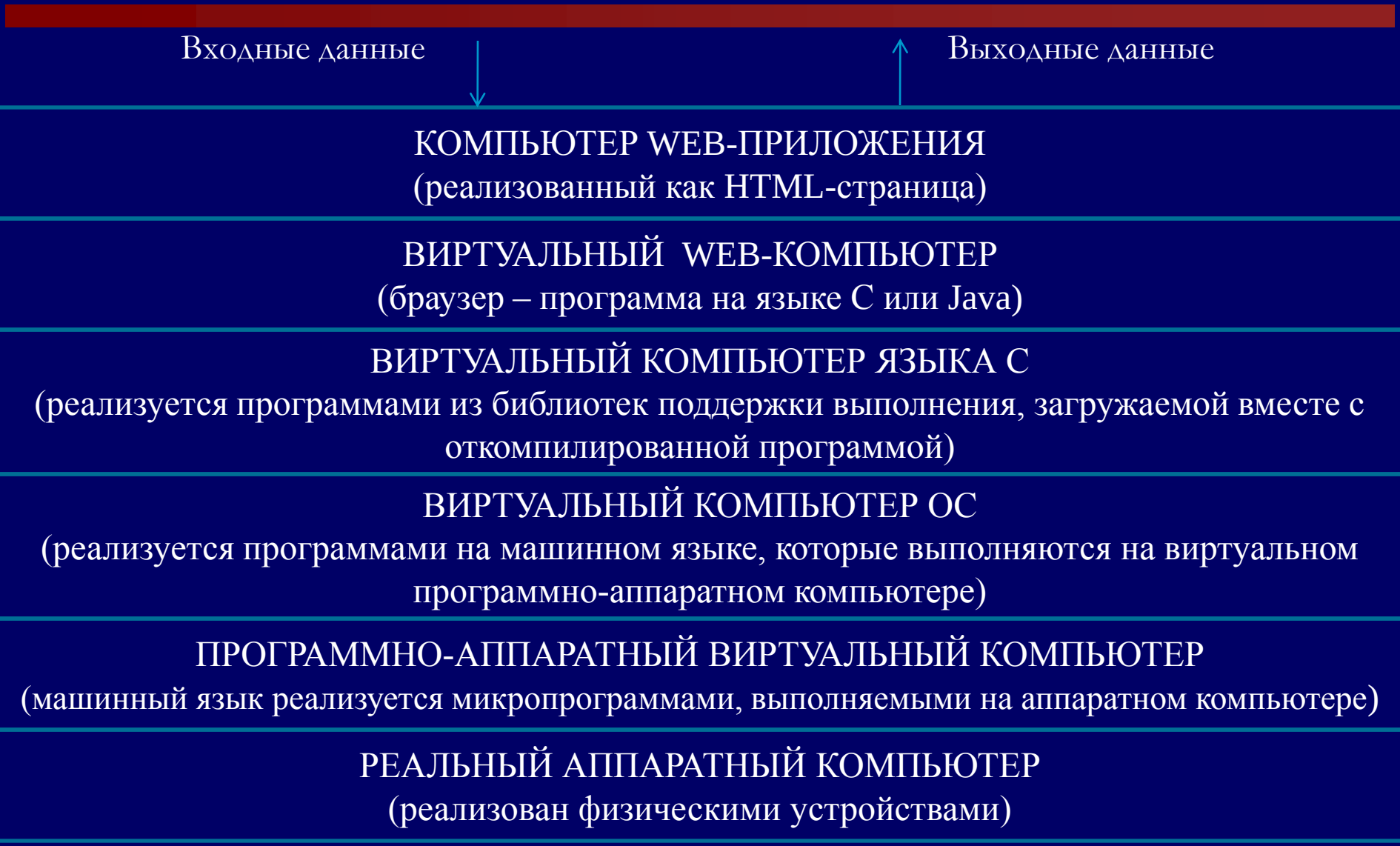
Организация и структура языка складываются с учетом аппаратных и программных возможностей базового компьютера и стоимости их использования.

Различия в реализациях языков

Различия в реализациях одного и того же языка объясняются следующими факторами:

- Разная архитектура виртуального компьютера;
- Различные базовые компьютеры предоставляют различные возможности;
- Разные реализации элементов виртуального компьютера.

Иерархия виртуальных компьютеров web-приложения



Связывание и время связывания

- *Связывание* элемента программы с конкретной характеристикой или свойством – это выбор определенного свойства из некоторого ряда допустимых свойств.
- Момент времени, когда при составлении или выполнении программы происходит связывание, называется *временем связывания*.

Классификация времени связывания

1. Во время выполнения программы:
   - при входе в подпрограмму или блок;
   - в произвольных точках программы во время ее выполнения.
2. Во время трансляции программы:
   - связывание по выбору программиста;
   - связывание по выбору транслятора;
   - связывание по выбору загрузчика.
3. Во время реализации языка.
4. Во время определения языка.

Гибкость и эффективность

- Если большая часть связываний происходит на этапе компиляции – *раннее связывание* (Fortran); при выполнении программы – *позднее связывание* (ML, HTML).
- Раннее связывание повышает эффективность программы, а позднее связывание – гибкость.
- Возможность управления временем связывания позволяет разрабатывать одновременно эффективный и гибкий язык программирования (Ada).