НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Введение в вычислительную технику

Москва
2016

Содержание

1. Основы вычислительной техники

1.1 Системы счисления

Кодирование – переход к другому алфавиту.

p-ичная ***система счисления*** :

$$a_k p^k + a_{k-1} p^{k-1} + \ldots + a_2 p^2 + a_1 p^1 + a_0 \quad , \qquad a_j < p$$

число в ***позиционной*** форме $a_k a_{k-1} \ldots a_2 a_1 a_0$

Двоичная система счисления (**p**=2):

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1024	**512**	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
					1	1	0	0	0	1	$= 110001_2$
					+32	+16				+1	$= 49_{10}$

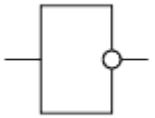
1. Основы вычислительной техники

1.2 Двоичная логика

Два утверждения: ***истина (true)*** = 1, ***ложь (false)*** = 0.

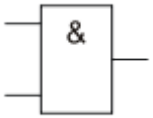
Логические операции:

Инверсия (отрицание)
(¬ , «НЕ», not, $\bar{}$)

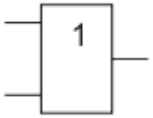


x	¬ *x*
0	1
1	0

Конъюнкция
(Λ , & , «И», and, ×, ·)



Дизъюнкция
(V , | , «ИЛИ», or, +)



x	*y*	*x* · *y*	*x* + *y*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Разложение логических функций (ДНФ)

$$f(x,y) = f(1,1)xy + f(1,0)x\bar{y} + f(0,1)\bar{x}y + f(0,0)\bar{x}\bar{y}$$

Пример: $f(x,y) = (x+\bar{y}) \cdot \overline{(xy+y)} = 0xy + 1x\bar{y} + 0\bar{x}y + 1\bar{x}\bar{y} = x\bar{y} + \bar{x}\bar{y}$

1. Основы вычислительной техники

1.3 Реализация битовых операций

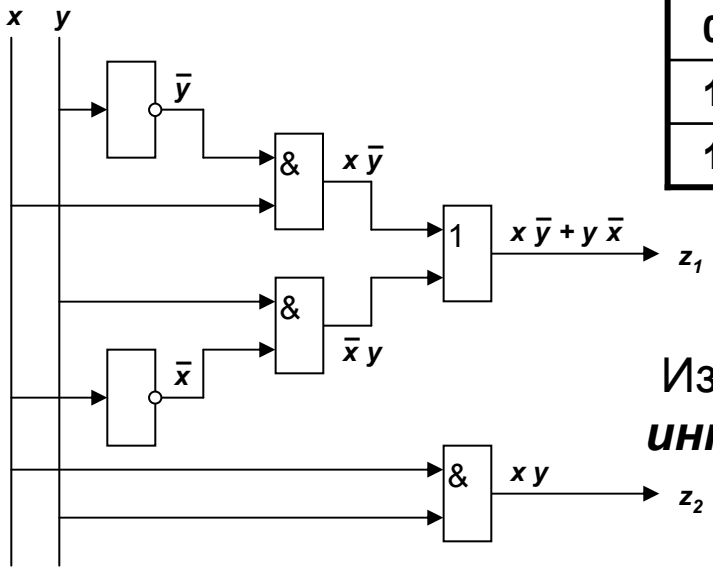
Вентиль – конструктивный логический элемент, выполняет элементарную логическую операцию, срабатывает, когда приходит ***тактовый*** импульс (т.е. в ***дискретном*** времени).

Арифметическое сложение:

$$x + y = z = z_2 z_1$$

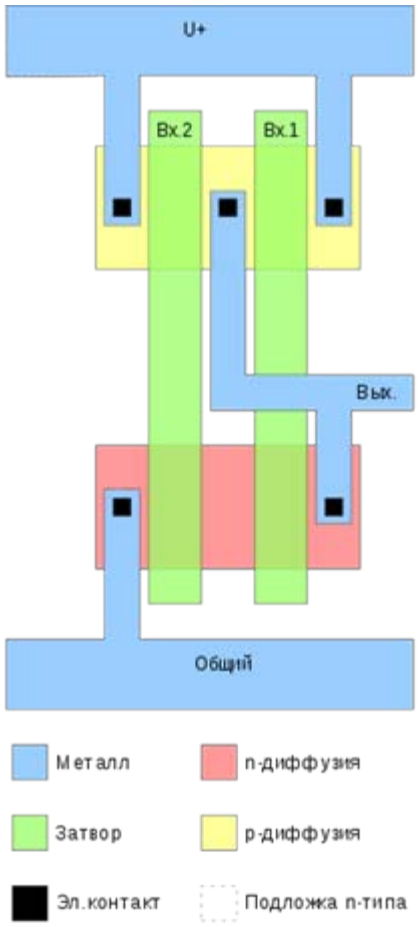
x	y	z_{bin}		z_{dec}
		z_2	z_1	
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	2

Схема сумматора:



(Три ***такта***.)

Из вентилей строятся ***интегральные схемы.***



КМОП схема «И-НЕ»

2. Архитектура ЭВМ

2.1 Память

Память – линейная упорядоченная последовательность ***ячеек***.
Время чтения/записи для всех ячеек одинаково (однородность памяти).

Адрес – номер ячейки. ***Разрядность*** памяти – количество бит в ячейке.

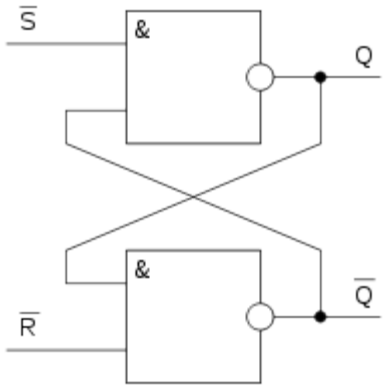
Триггер – конструктивный элемент памяти.

Сохраняет свое предыдущее состояние при нулевых входах.

При подаче 1 на вход
S (set) – на выходе 1;
R (reset) – на выходе 0.

Принцип обратной связи

S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	–
1	1	1	–



Асинхронный RS-триггер на базе «И-НЕ»

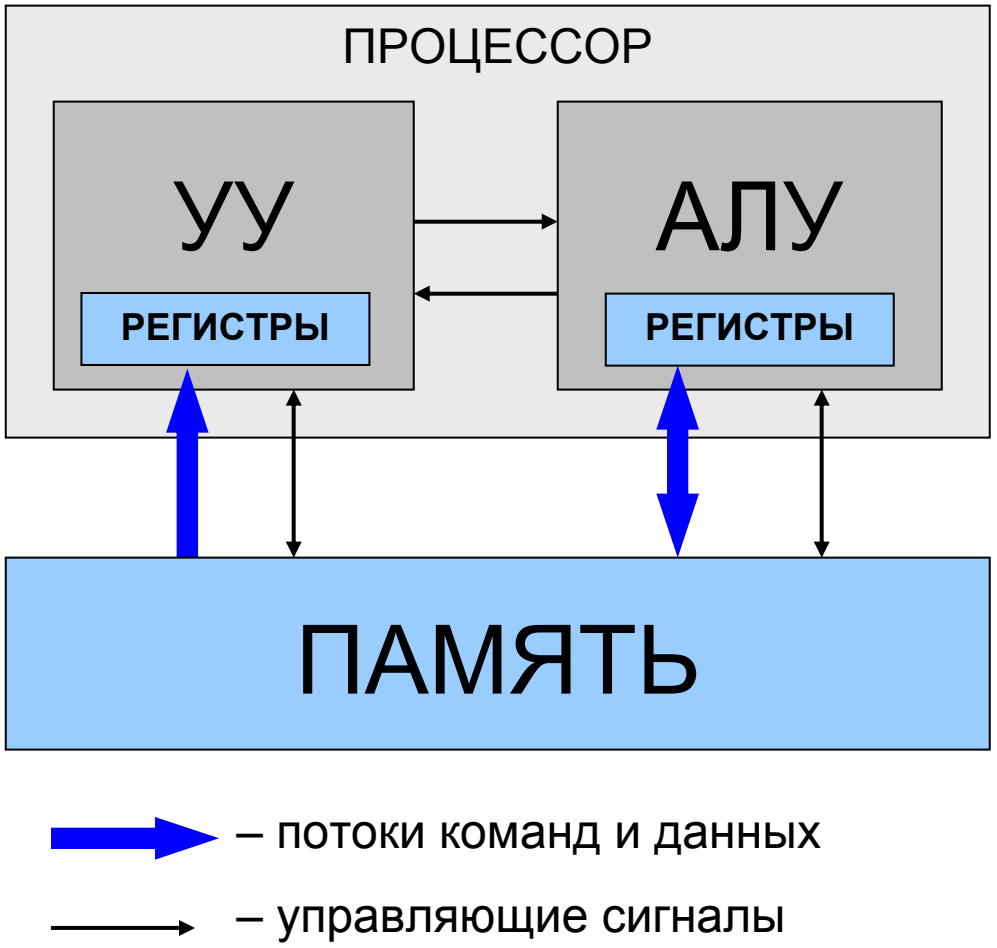
2. Архитектура ЭВМ

2.2 Машина фон Неймана

Арифметико-логическое устройство (АЛУ):
- считывает содержимое памяти в регистры,
- выполняет операции над содержимым регистров,
- записывает содержимое регистров в память.

Устройство управления (УУ):
- автоматическая работа,
- последовательное выполнение команд.

Каждое устройство отвечает за выполнение только своих функций (*специализация*).



2. Архитектура ЭВМ

2.3 Данные, команды, программы

Машинное слово – содержимое ячейки памяти – либо ***команда***, либо элемент ***данных***. Команды и данные неотличимы друг от друга.

***Совместное хранение* программ (команд) и данных в общей памяти.**

Программа может изменяться во время счета – гибкость вычислений.

Структура двухадресной команды:

КОП	A1	A2
4 бита	6 бит	6 бит

разрядность: 16 бит
доступная память: 2^6 = 64 ячейки
можно закодировать: 2^4 = 16 команд

КОП – код операции, A1 и A2 – адреса операндов.

Введем коды операций:
0001 – сложение,
0010 – чтение из памяти в регистр,
1111 – запись из регистра в память.

Выделим память под переменные:
x – адрес 010100,
y – адрес 010101,
z – адрес 010110.

Закодируем регистр R числом 000001.

Пример: программа в машинных кодах, которая вычисляет сумму z = x + y

Программа	Комментарий
0010 000001 010100	**R := x**
0001 000001 010101	**R := R + y**
1111 000001 010110	**z := R**

3. Языки программирования

3.1 Язык ассемблера

Assembler (англ. – *сборщик*) – машинно-ориентированный язык.

Преимущества:

- мнемонические коды операций (КОП) – ***команды***,
- символические имена регистров и ячеек памяти – ***переменные***,
- различные схемы адресации,
- представление чисел в различных системах счисления,
- использование меток **и др.**

Тот же пример: программа, которая вычисляет сумму z = x + y

Машинный код	Код на Ассемблере	Комментарий
0010 000001 010100	`mov AX, x`	**R := x** (**AX** – имя регистра **R**)
0001 000001 010101	`add AX, y`	**R := R + y**
1111 000001 010110	`mov z, AX`	**z := R**

Код на языке высокого уровня

```
z = x + y      BASIC
z := x + y;    Pascal
z = x + y;     C/C++
```

при этом выполняется весь набор необходимых машинных инструкций.

1. Языки программирования

1.1 Язык ассемблера

Пример программы на языке TASM (Turbo Assembler) для DOS.
Программа вычисляет сумму z = x + y, приведена целиком.

```
   .MODEL SMALL      ;задать механизм распределения памяти под команды и данные
   .DATA             ;начало участка программы с данными
   x  DW  15         ;выделить память для переменной x, ее значение =15
   y  DW  38         ;выделить память для y (DW - два байта), ее значение =38
   z  DW  ?          ;выделить память для z, ее значение не определено
   .CODE             ;начало участка программы с командами
Begin:               ;метка начала выполняемой программы
   mov  AX,@Data     ;записать адрес сегмента данных в регистр AX
   mov  DS,AX        ;записать содержимое AX в регистр сегмента данных DS[1]
   mov  AX,x
   add  AX,y         ;вычисление по формуле  z = x + y
   mov  z,AX
   mov  AH,4Ch       ;загрузка в регистр AX[2] функции DOS завершения программы(4Ch[3])
   int  21h          ;вызов прерывания для выполнения функции в регистре AX
   END  Begin        ;окончание программы, указание места начала ее выполнения
```

[1] – непосредственно изменять содержимое регистра **DS** запрещено, поэтому сначала адрес сегмента данных помещается в регистр **AX**.

[2] – регистр **AX** состоит из младшего(**AL**) и старшего(**AH**) байтов.

[3] – **4Ch** - это число **4C** в 16-чной системе счисления (на что указывает **h** – hexadecimal), оно равно **76** в десятичной системе счисления. Аналогично - число **21h**, равно **33**.

3. Языки программирования

3.1 Язык ассемблера

Программа состоит из:
- ***команд*** – мнемонических *инструкций* процессору (**`mov`**, **`add`**, **`int`**, …),
- ***директив*** – *команд* для компилятора (**`DATA`**, **`CODE`**, **`END`**, …).

Программирование на ассемблере.

Достоинства:
- самый **быстрый** и **компактный** код *для данного процессора;*
- близкий к **оптимальному** код по сравнению с транслятором;
- максимальное использование **возможностей** *конкретной платформы;*
- непосредственный доступ к **аппаратуре**.

Недостатки:
- сложнее **читать** и **понимать** программу;
- высокая **трудоёмкость** и вероятность внесения **ошибок**;
- повышенная **квалификация** программиста;
- **машиннозависимые** программы;
- отсутствие **переносимости** программ на новую платформу.

3. Языки программирования

3.2 Классификация языков программирования

Уровень языка тем выше, чем большее **количество машинных команд** выполняют команды этого языка.

- ***низкий*** (ассемблеры, макроассемблер)
- **[*средний* (C)]**
- ***высокий*** (BASIC, FORTRAN, PL/1, Pascal, Ada, C++, Java, C#, …)
- ***сверхвысокий*** (Perl, Haskell, Ruby, …)

Основные ***парадигмы*** программирования:

- ***императивная****:*
  - *процедурная* (объединение операторов в подпрограммы),
  - *структурная* (программа как иерархическая структура блоков),
  - *модульная* (разделение программы на отдельные модули),
  - *объектно-ориентированная* (*ООП*, концепции объектов и классов);
- ***декларативная****:*
  - *функциональная* (вычисление значений функций в матем.понимании),
  - *логическая* (автоматическое доказательство, логический вывод).

В соответствии с парадигмами классифицируют языки.