Глава 5

ЦИФРОВЫЕ ФУНКЦИОНАЛЬНЫЕ УЗЛЫ

Цифровая схемотехника и архитектура компьютера, второе издание

Дэвид М. Харрис и Сара Л. Харрис

Цифровая схемотехника и архитектура компьютера

Эти слайды предназначены для преподавателей, которые читают лекции на основе учебника «Цифровая схемотехника и архитектура компьютера» авторов Дэвида Харриса и Сары Харрис. Бесплатный русский перевод второго издания этого учебника можно загрузить с сайта компании Imagination Technologies:

https://community.imgtec.com/downloads/digital-design-and-computer-architecture-russian-edition-second-edition

Процедура регистрации на сайте компании Imagination Technologies описана на станице:

http://www.silicon-russia.com/2016/08/04/harris-and-harris-2/

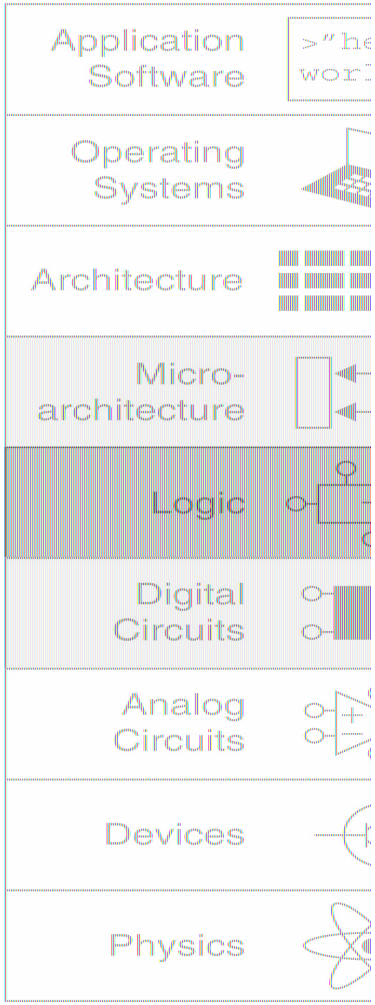
Благодарности

Перевод данных слайдов на русский язык был выполнен командой сотрудников университетов и компаний из России, Украины, США в составе:

- Александр Барабанов - доцент кафедры компьютерной инженерии факультета радиофизики, электроники и компьютерных систем Киевского национального университета имени Тараса Шевченко, кандидат физ.-мат. наук, Киев, Украина;
- Антон Брюзгин - начальник отдела АО «Вибро-прибор», Санкт-Петербург, Россия.
- Евгений Короткий - доцент кафедры конструирования электронно-вычислительной аппаратуры факультета электроники Национального технического университета Украины «Киевский Политехнический Институт», руководитель открытой лаборатории электроники Lampa, кандидат технических наук, Киев, Украина;
- Евгения Литвинова – заместитель декана факультета компьютерной инженерии и управления, доктор технических наук, профессор кафедры автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, Харьков, Украина;
- Юрий Панчул - старший инженер по разработке и верификации блоков микропроцессорного ядра в команде MIPS I6400, Imagination Technologies, отделение в Санта-Кларе, Калифорния, США;
- Дмитрий Рожко - инженер-программист АО «Вибро-прибор», магистр Санкт-Петербургского государственного автономного университета аэрокосмического приборостроения (ГУАП), Санкт-Петербург, Россия;
- Владимир Хаханов – декан факультета компьютерной инженерии и управления, проректор по научной работе, доктор технических наук, профессор кафедры автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, Харьков, Украина;
- Светлана Чумаченко – заведующая кафедрой автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, доктор технических наук, профессор, Харьков, Украина.

Глава 5 :: Темы

- **Введение**
- **Арифметические схемы**
- **Представление чисел**
- **Последовательностные функциональные блоки**
- **Матрицы памяти**
- **Матрицы логических элементов**

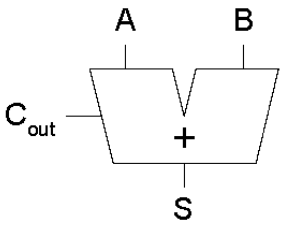


Введение

- **Цифровые функциональные блоки:**
  - Логические элементы, мультиплексоры, декодеры, регистры, схемы арифметики, счетчики, матрицы памяти и матрицы логических элементов
- **Функциональные блоки демонстрируют принципы иерархичности, модульности и регулярности проектируемых систем**
  - Иерархия более простых компонентов
  - Строго определенные интерфейсы и функции
  - Регулярная структура легко масштабируется в системы различных размеров
- **Подобные функциональные блоки будут использованы в главе 7 для проектирования микропроцессора**

Одноразрядный сумматор. 1

Half Adder

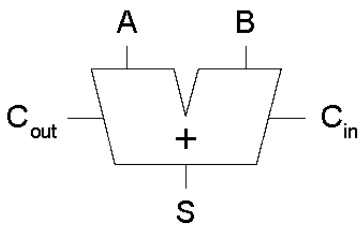


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

S =

C_{out} =

Full Adder



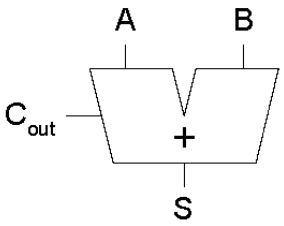
C_{in}	A	B	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

S =

C_{out} =

Одноразрядный сумматор. 2

Half Adder

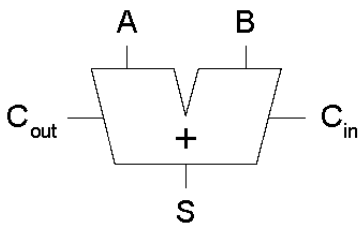


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S =

C_{out} =

Full Adder



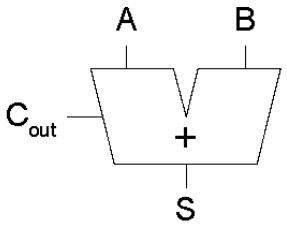
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S =

C_{out} =

Одноразрядный сумматор. 3

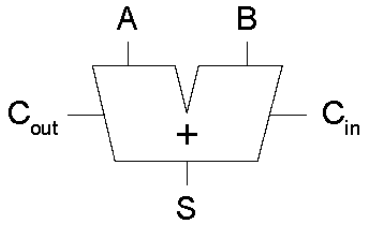
Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$

Full Adder



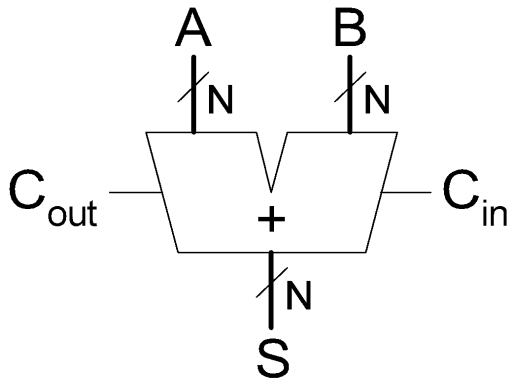
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Многоразрядные сумматоры

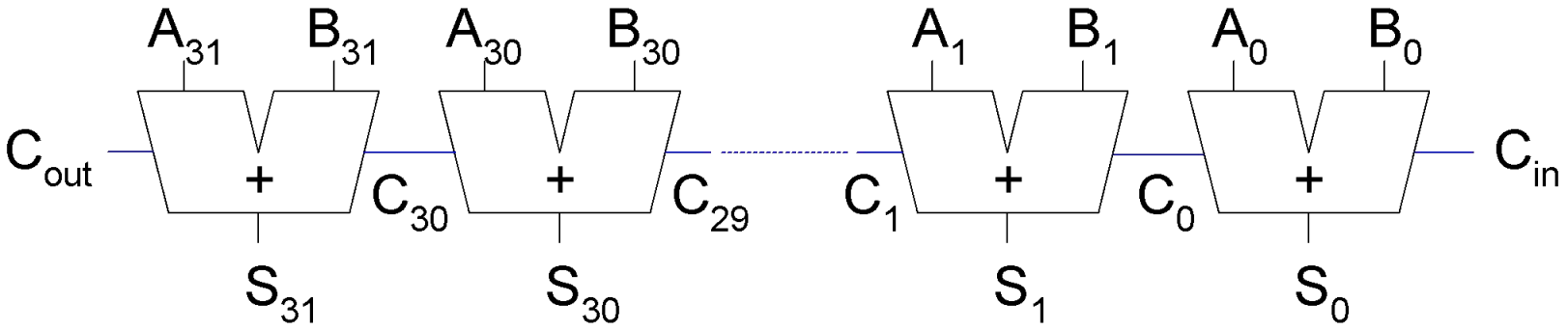
- Типы распространения переносов:
  - Последовательный (медленный)
  - Ускоренный групповой (быстрый)
  - Префиксный (самый быстрый)
- Два последних типа используются для многоразрядных сумматоров, но их реализация требует дополнительных аппаратных затрат

Условное обозначение



Сумматор с последовательным переносом

- Цепь одноразрядных сумматоров
- Перенос проходит через всю цепочку
- Недостаток: **медленное суммирование**



Сумматор с последовательным переносом

Задержка сумматора складывается из задержек разрядов в каждом звене:

$$t_{ripple} = Nt_{FA}$$

где t_{FA} – задержка одного полного сумматора

Сумматор с ускоренным групповым переносом (СУГП)

- Определить значение переноса (C_{out}) в каждом блоке *k*-разрядного сумматора, используя сигналы *generate* и *propagate*
- **Некоторые определения:**
  - Разряд *i* формирует перенос либо путем его генерирования (*generating*) либо путем распространения (*propagating*) переноса со своего соответствующего входа на свой выход
  - Генерирование (G_i) и распространение (P_i) сигналов для каждого разряда:
    - Разряд *i* будет генерировать перенос, если A_i и B_i (оба) равны 1.

$$G_i = A_i B_i$$

    - Разряд *i* будет распространять перенос от соответствующего входа к соответствующему выходу, если A_i или B_i равен 1.

$$P_i = A_i + B_i$$

    - Перенос разряда *i* (C_i):

$$C_i = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1}$$

Суммирование с ускоренным групповым переносом

- **Шаг1:** Вычислить G_i и P_i для всех разрядов
- **Шаг 2:** Вычислить G и P для всех k-битовых блоков сумматора
- **Шаг 3:** Перенос C_{in} распространяется через все k-битовые блоки генерации/распространения

Сумматор с ускоренным групповым переносом

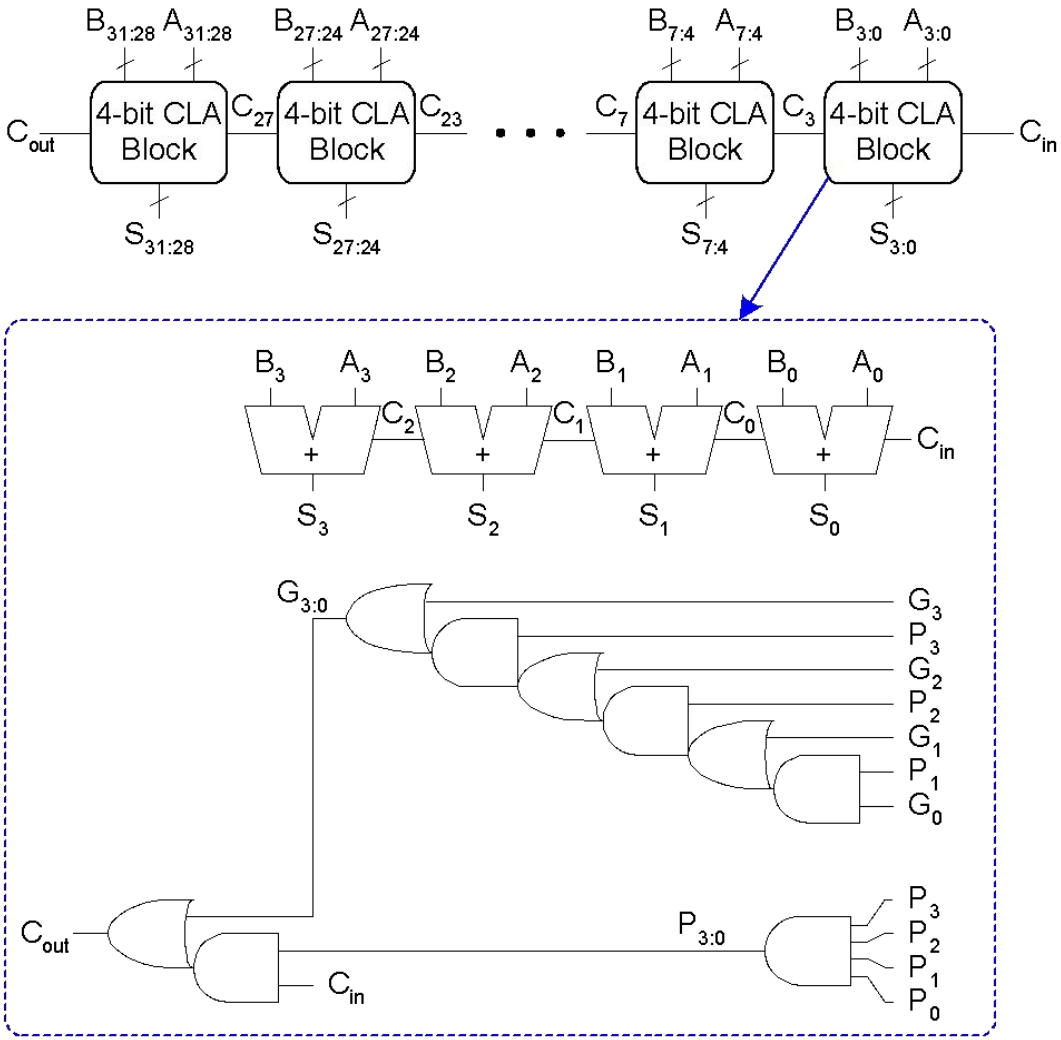
- **Пример:** 4-разрядные блоки ($G_{3:0}$ и $P_{3:0}$) :

$$G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0))$$
$$P_{3:0} = P_3 P_2 P_1 P_0$$

- **В общем случае,**

$$G_{i:j} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} G_j))$$
$$P_{i:j} = P_i P_{i-1} P_{i-2} P_j$$
$$C_i = G_{i:j} + P_{i:j} C_{i-1}$$

32-разрядный сумматор с ускоренным групповым переносом с 4-разрядными блоками



Задержки сумматор с ускоренным групповым переносом

Для N-разрядного сумматор с ускоренным групповым переносом с k-разрядными блоками:

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + kt_{FA}$$

- t_{pg} : задержка генерации всех P_i, G_i
- t_{pg_block} : задержка генерации всех $P_{i:j}$, $G_{i:j}$
- t_{AND_OR} : задержка тракта вход C_{in} - выход C_{out} из элементов И/ИЛИ в k-разрядном блоке сумматор с ускоренным групповым переносом

N-разрядный сумматор с ускоренным групповым переносом практически всегда более быстрый, чем сумматор с последовательным переносом для $N > 16$

Префиксный сумматор

- Вычисляет перенос на входе (C_{i-1}) для каждого разряда, затем вычисляет сумму:

$$S_i = (A_i \oplus B_i) \oplus C_i$$

- Вычисляет G и P для 1-, 2-, 4-, 8-разрядов блоков, до тех пор, пока не станут известны все переносы G_i (входные переносы всех разрядов)
- Количество каскадов $\log_2 N$

Префиксный сумматор

- Перенос на входе либо генерируется *для данного разряда* либо распространяется от предыдущего.
- Разряд -1 соответствует C_{in}, тогда

$$G_{-1} = C_{in}, P_{-1} = 0$$

- Значение переноса на входе разряда *i* равно *значению переноса на выходе разряда i-1*:

$$C_{i-1} = G_{i-1:-1}$$

$G_{i-1:-1}$: сигнал генерации блока разрядов от *i*-1 до -1

- Выражение для суммы:

$$S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$$

- **Цель:** быстрое вычисление $G_{0:-1}$, $G_{1:-1}$, $G_{2:-1}$, $G_{3:-1}$, $G_{4:-1}$, $G_{5:-1}$, … (называемых ***префиксами)***

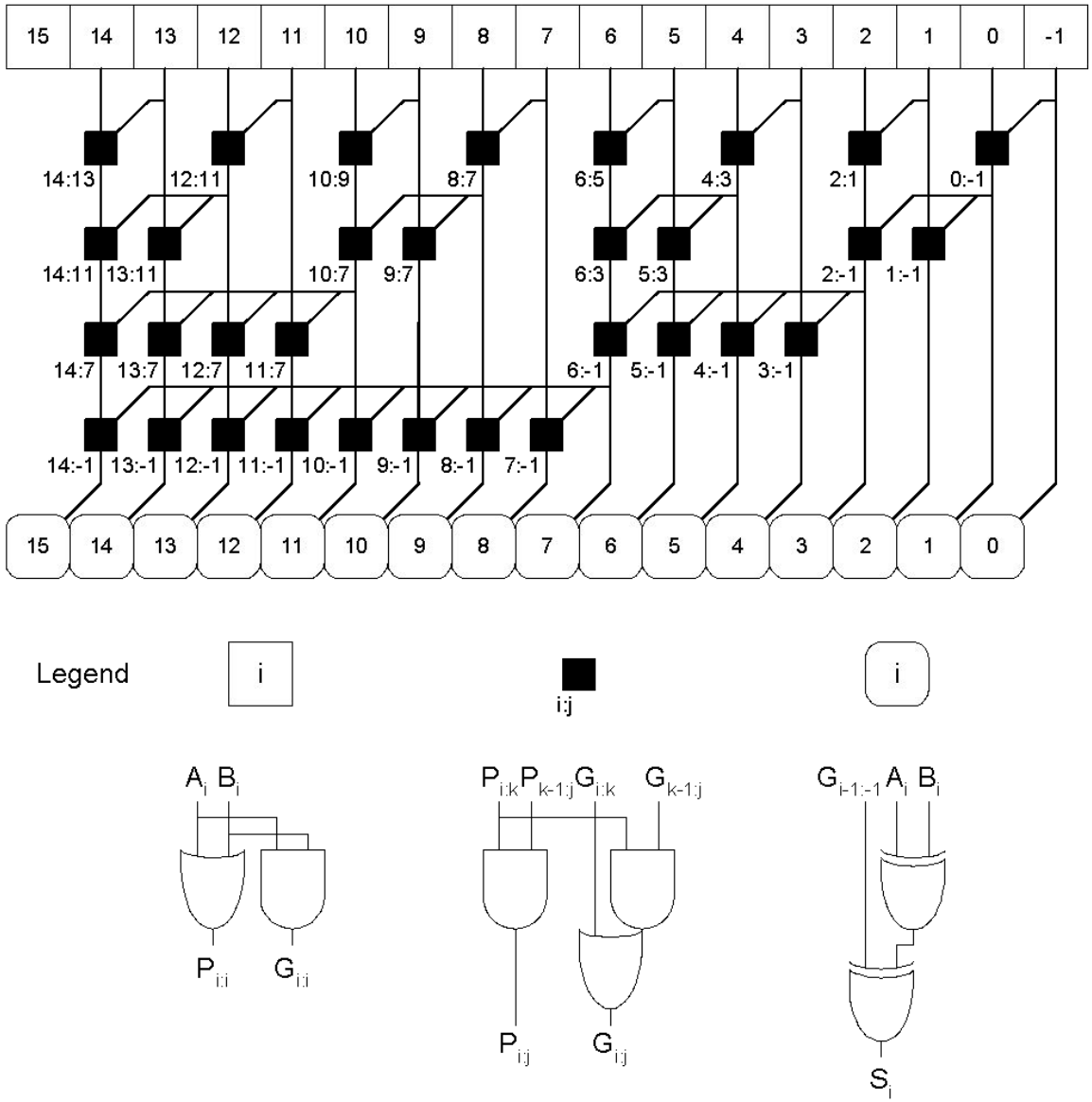
Префиксный сумматор

- Сигналы генерации и распространения блока, охватывающего разряды $i{:}j$:

$$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k-1:j}$$
$$P_{i:j} = P_{i:k} P_{k-1:j}$$

- Более детально:
  - **Генерация:** блок $i{:}j$ генерирует перенос, если:
    - Старшие разряды ($i{:}k$) генерируют перенос или
    - Старшие разряды распространяют перенос, сгенерированный в младших разрядах (k-1:j)
  - **Распространение:** блок $i{:}j$ распространяет перенос, если и старшие и младшие разряды *распространяют перенос*

Схема префиксного сумматора



Задержка префиксного сумматора

$$t_{PA} = t_{pg} + \log_2 N(t_{pg\_prefix}) + t_{XOR}$$

- t_{pg}: задержка формирования P_i G_i (элементы И или ИЛИ)
- t_{pg_prefix}: задержка черной префиксной ячейки (элементы И-ИЛИ)

Сравнение задержек

Сравнить задержки 32-разрядный сумматор с последовательным переносом, сумматор с ускоренным групповым переносом, и префиксного сумматора

- сумматор с ускоренным групповым переносом содержит 4-разрядные блоки
- Задержка 2-входового вентиля = 100 ps; задержка полного сумматора = 300 ps

Сравнение задержек

Сравнить задержки 32-разрядный сумматор с последовательным переносом, сумматор с ускоренным групповым переносом и префиксный сумматор

- сумматор с ускоренным групповым переносом содержит 4-разрядные блоки
- Задержка 2-входового вентиля = 100 ps; задержка полного сумматора = 300 ps

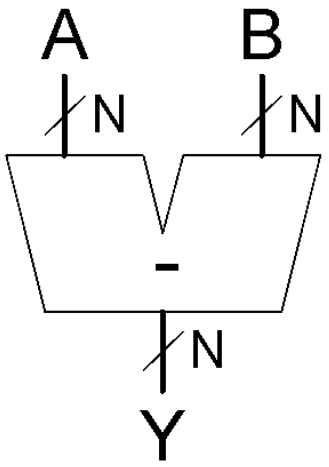
$$t_{\text{ripple}} = Nt_{FA} = 32(300\text{ ps}) = \mathbf{9.6\text{ ns}}$$

$$t_{CLA} = t_{pg} + t_{pg\_\text{block}} + (N/k - 1)t_{\text{AND\_OR}} + kt_{FA}$$
$$= [100 + 600 + (7)200 + 4(300)]\text{ ps} = \mathbf{3.3\text{ ns}}$$

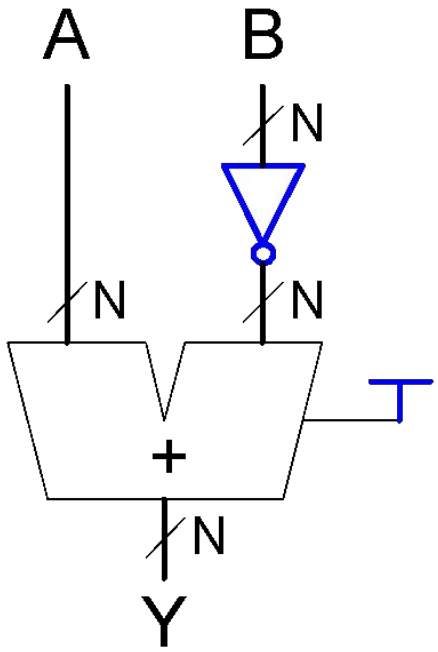
$$t_{PA} = t_{pg} + \log_2 N(t_{pg\_\text{prefix}}) + t_{\text{XOR}}$$
$$= [100 + \log_2 32(200) + 100]\text{ ps} = \mathbf{1.2\text{ ns}}$$

Устройство вычитания

Symbol

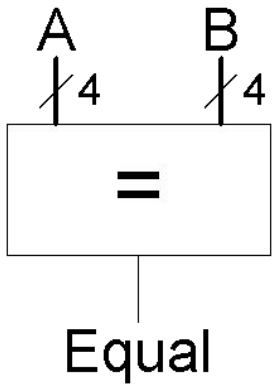


Implementation

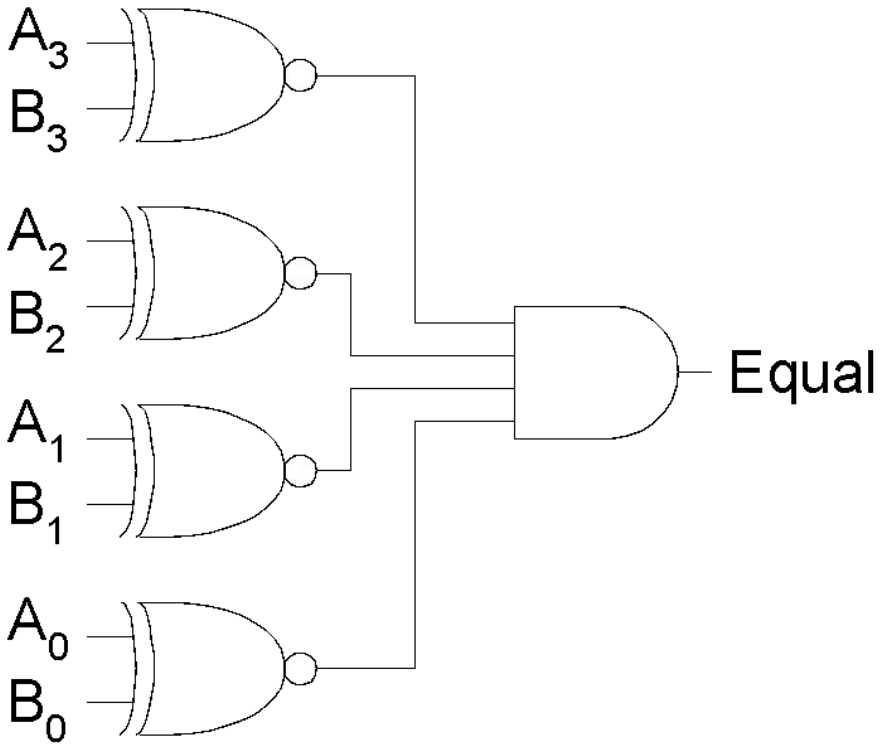


Компаратор: Сравнение на равенство

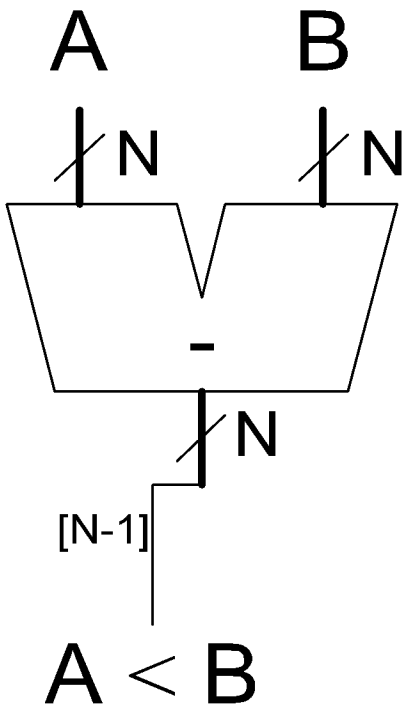
Symbol



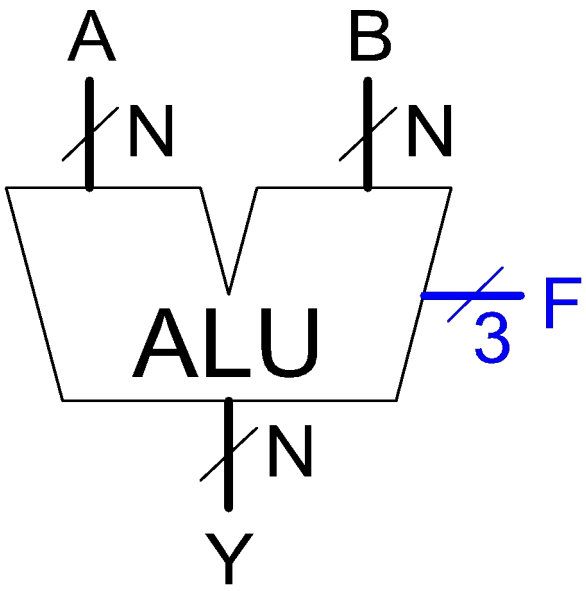
Implementation



Компаратор: Меньше, чем

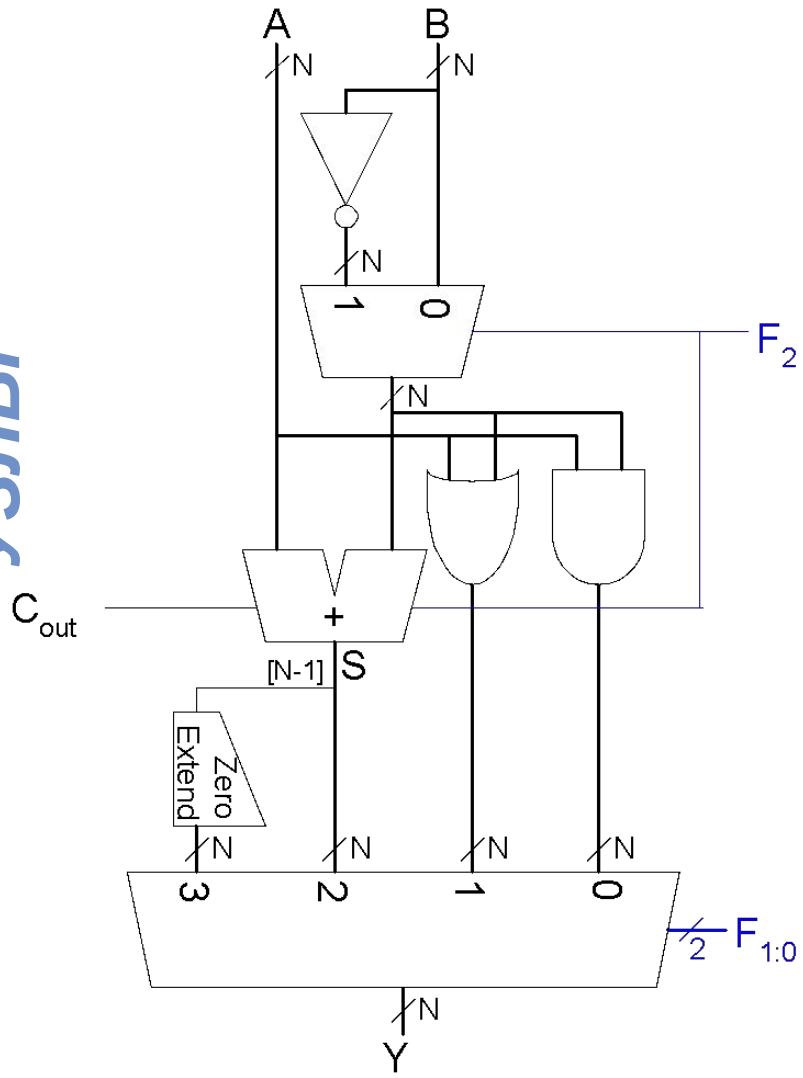


Арифметико-логическое устройство (АЛУ)



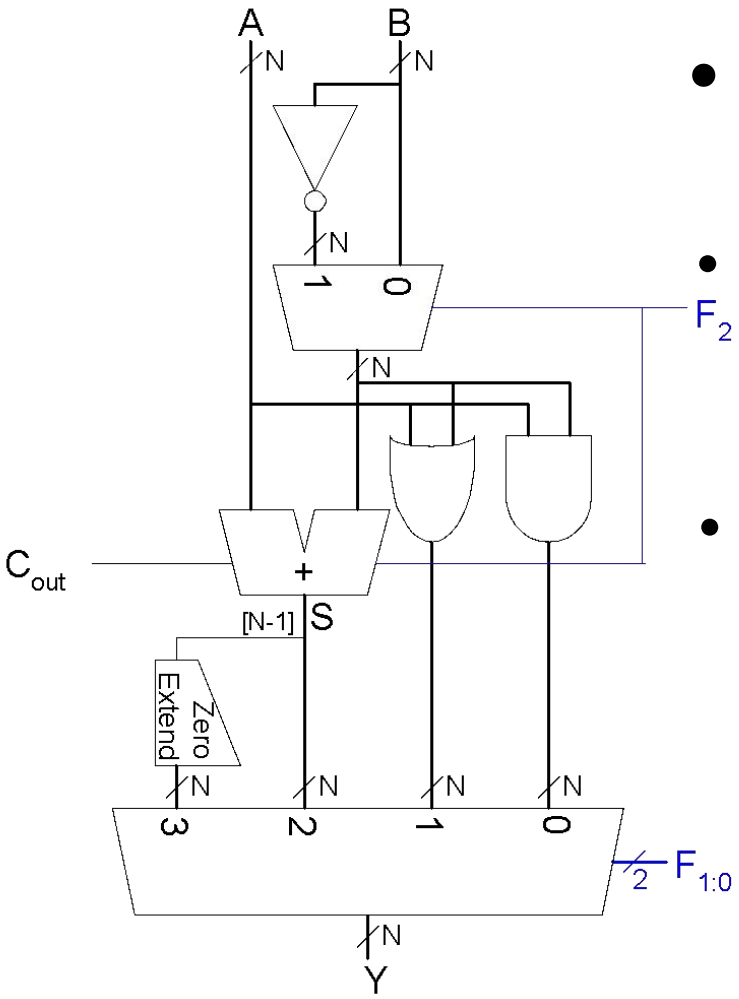
$F_{2:0}$	Функция
000	A & B
001	A | B
010	A + B
011	Не используется
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Схема АЛУ



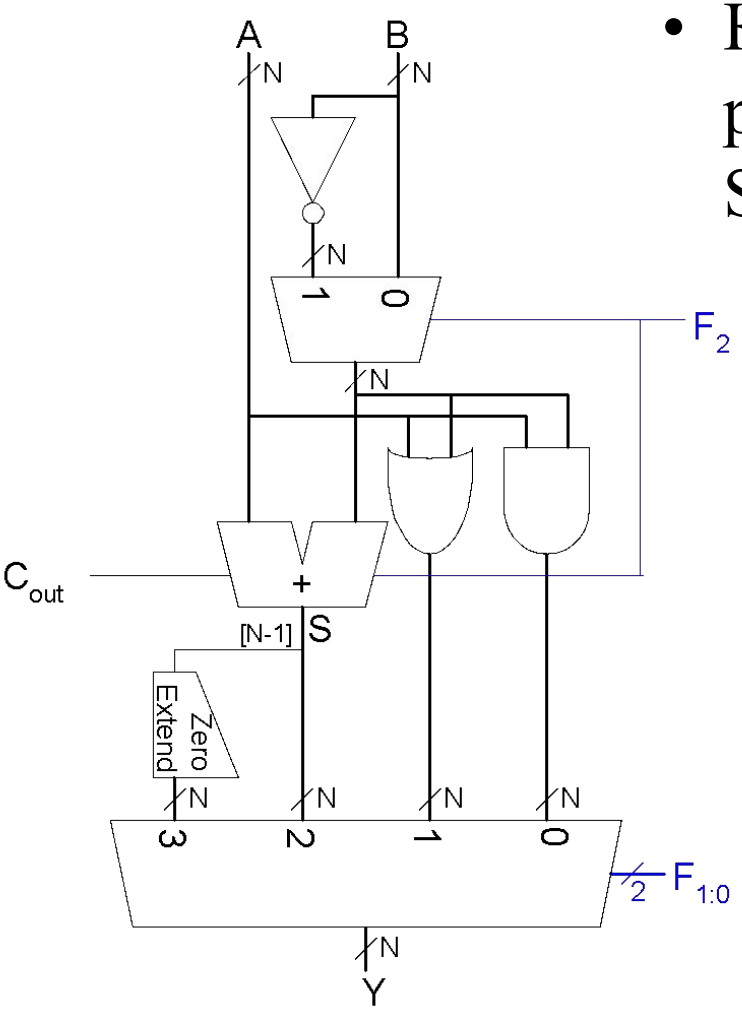
$F_{2:0}$	Функция
000	A & B
001	A | B
010	A + B
011	Не используется
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Сравнение «Меньше, чем». Пример



- Сравнение на «Меньше» (Set Less Than, SLT)
- Конфигурирование 32-разрядного АЛУ для операции SLT:
- $A = 25$ и $B = 32$

Сравнение «Меньше, чем». Пример



- Конфигурирование 32-разрядного АЛУ для операции SLT: $A = 25$ и $B = 32$
  - $A < B$, поэтому Y должен быть 32-разрядным представлением 1 (0x00000001)
  - $F_{2:0} = 111$
  - $F_2 = 1$ (сумматор работает как вычитатель): 25 - 32 = -7
  - -7 имеет 1 в старшем разряде ($S_{31} = 1$)
  - $F_{1:0} = 11$ мультиплексор выбирает $Y = S_{31}$ (дополнение нулями) = 0x00000001.

Устройство сдвига (Shifter)

- **Логическое устройство сдвига:** смещает сдвигаемое значение влево или вправо и заполняет пустые разряды нулями «0»
  Пример: 11001 >> 2 =
  Пример 11001 << 2 =

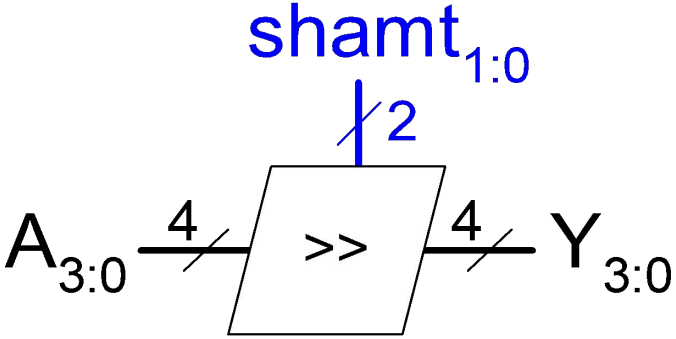
- **Арифметическое устройство сдвига:** при сдвиге влево работает так же, как и логическое, а при сдвиге вправо заполняет пустые разряды значением старшего бита (most significant bit, msb).
  Пример 11001 >>> 2 =
  Пример : 11001 <<< 2 =

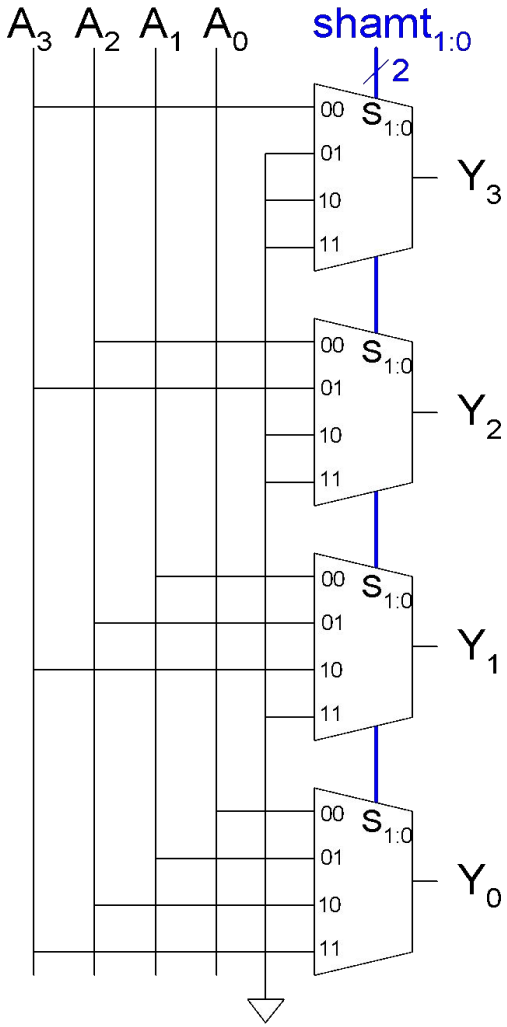
- **Циклический сдвиг:** сдвигает биты по кругу, таким образом, что уходящий бит появляется на месте появившегося свободного разряда на другом конце числа
  Пример : 11001 ROR 2 =
  Пример : 11001 ROL 2 =

Устройства сдвига

- **Логическое устройство сдвига:**
  - Ex: 11001 >> 2 = 00110
  - Ex: 11001 << 2 = 00100
- **Арифметическое устройство сдвига:**
  - Ex: 11001 >>> 2 = **11**110
  - Ex: 11001 <<< 2 = 00100
- **Циклическое устройство сдвига:**
  - Ex: 11001 ROR 2 = 01110
  - Ex: 11001 ROL 2 = 00111

Схема устройства сдвига





Устройство сдвига как умножитель и делитель

- $A << N = A \times 2^N$
  - **Пример:** 00001 << 2 = 00100 $(1 \times 2^2 = 4)$
  - **Пример :** 11101 << 2 = 10100 $(-3 \times 2^2 = -12)$
- $A >>> N = A \div 2^N$
  - **Пример :** 01000 >>> 2 = 00010 $(8 \div 2^2 = 2)$
  - **Пример :** 10000 >>> 2 = 11100 $(-16 \div 2^2 = -4)$

Устройство умножения

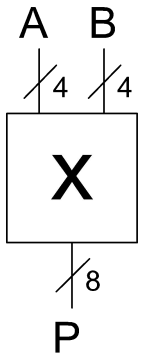
- **Частичное произведение,** формируемое путем умножения текущего разряда множителя на все разряды множимого
- **Сдвинутые частичные произведения,** суммированные для формирования результата

Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
9660		0101
		+ 0000
	result	0100011

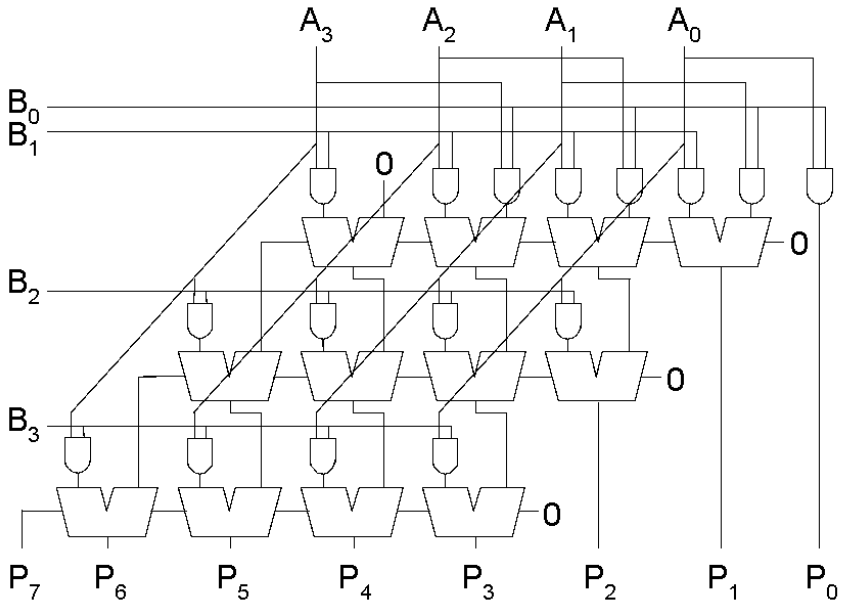
230 x 42 = 9660

5 x 7 = 35

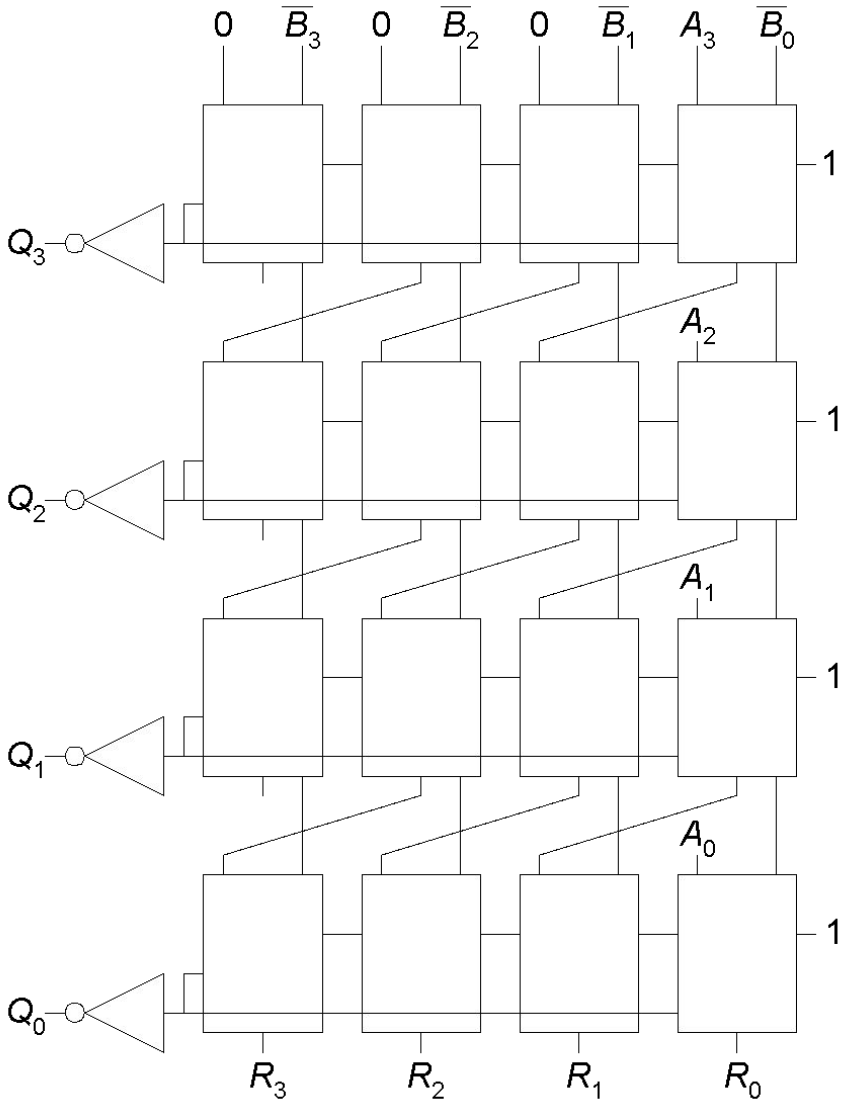
Умножитель 4 x 4

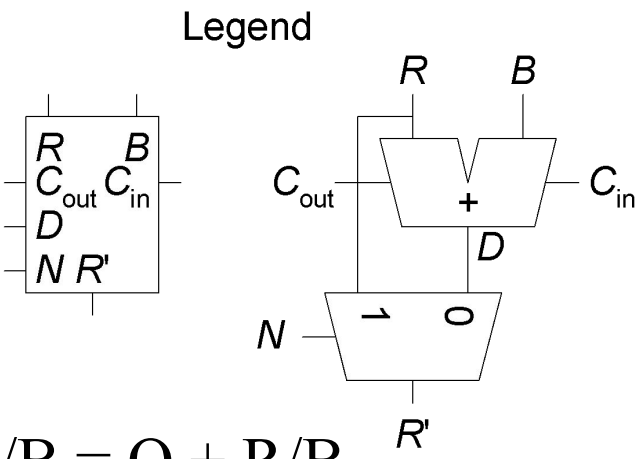


$$
\begin{array}{rcccccccc}
 & & & A_3 & A_2 & A_1 & A_0 & & \\
 & x & & B_3 & B_2 & B_1 & B_0 & & \\
\hline
 & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & & & \\
 & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & & & & \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & & & & \\
+ \; A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$



Делитель 4 x 4





$A/B = Q + R/B$

Алгоритм:

$R' = 0$

for i = N-1 to 0

  $R = \{R' << 1. A_i\}$

  $D = R - B$

  if $D < 0$, Q_i=0, $R' = R$

  else Q_i=1, $R' = D$

$R' = R$

Системы счисления

- Числа можно представить с помощью двоичного представления
  - **Положительные числа**
    - Беззнаковое двоичное
  - **Отрицательные числа**
    - Дополнительный код
    - Прямой код
- А как же дробные числа?

Дробные числа

- Две основных способа представления:
  - **Fixed-point:** двоичное число с фиксированной точкой
  - **Floating-point:** двоичное число с плавающей точкой – двоичная точка «плавает» между значащими цифрами

Числа с фиксированной точкой

- 6.75 содержит 4 разряда целой части и 4 разряда дробной части :

$$01101100$$

$$0110.1100$$

$$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$$

- В определенном месте подразумевается наличие двоичной точки
- Количество разрядов целой и дробной части должно быть определено заранее

Пример числа с фиксированной точкой

- Представить 7.5_{10} используя 4 целых бита и 4 дробных.

Пример числа с фиксированной точкой

- Представить 7.5_{10} используя 4 целых бита и 4 дробных.

01111000

Знаковые числа с фиксированной точкой

- **Представление:**
  - Прямой код (знак/величина)
  - Дополнительный код (дополнение до основания системы счисления)
- **Пример:** Представить -7.5_{10} используя 4 целых и 4 дробных бита
  - **Прямой код:**
  - **Дополнительный код:**

Знаковые числа с фиксированной точкой

- **Представление:**
  - Прямой код (знак/величина)
  - Дополнительный код (дополнение до основания системы счисления)
- **Пример:** Представить -7.5_{10} используя 4 целых и 4 дробных бита
  - **Прямой код:**

    11111000
  - **Дополнительный код:**

    1. +7.5: 01111000
    2. Инвертировать значения разрядов: 10000111
    3. Добавить 1 к младшему разряду: + 1

    10001000

Числа с плавающей точкой

- Двоичная точка «плавает» между значащими цифрами
- Подобно десятичному представлению в экспоненциальном представлении
- Например, записать 273_{10} в экспоненциальном представлении :

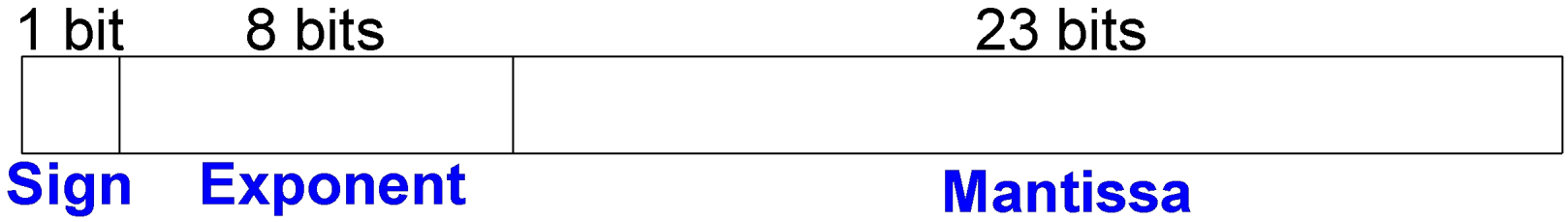
$$273 = 2.73 \times 10^2$$

- В общем виде, число записывается в экспоненциальном представлении как:

$$\pm M \times B^E$$

  - **M** = мантисса
  - **B** = основание показательной функции
  - **E** = порядок (экспонента)
  - Например, M = 2.73, B = 10, and E = 2

Числа с плавающей точкой



- **Пример:** представить число 228_{10} используя 32-битное представление с плавающей точкой

Рассмотрим 3 версии – последняя версия называется IEEE 754 floating-point standard

Представление с плавающей точкой, 1

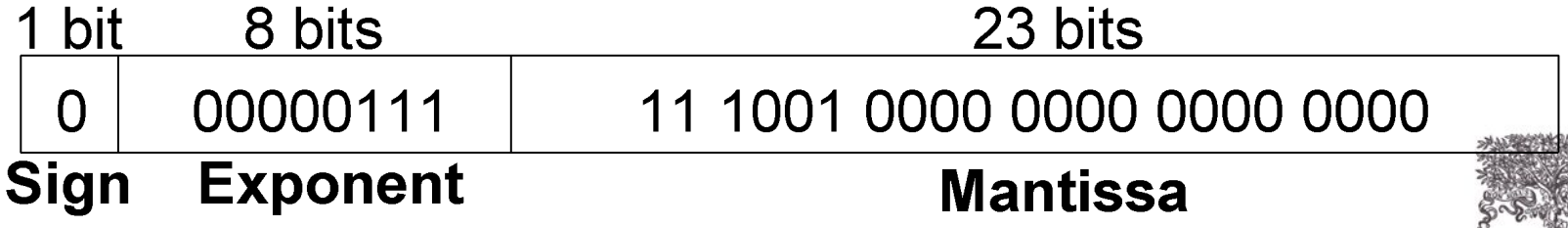
1. Преобразовать десятичное в двоичное (**не меняйте местами шаги 1 & 2!**):

$$228_{10} = 11100100_2$$

2. Записать число в двоичной системе и экспоненциальном представлении:

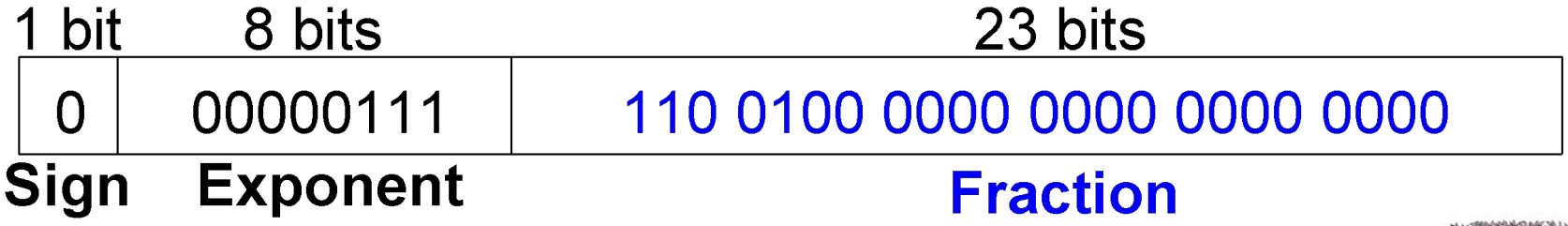
$$11100100_2 = 1.11001_2 \times 2^7$$

3. Заполнить каждое поле 32-битное числа с плавающей точкой:
   - Знак – положительный, знаковый бит – (0)
   - 8 разрядов порядка представляют значение 7
   - Остальные 23 разряда – мантисса



Представление с плавающей точкой 2

- Первый разряд мантиссы – всегда 1:
  - $228_{10} = 11100100_2 = \mathbf{1}.11001 \times 2^7$
- Следовательно, нет необходимости хранить его: *неявная ведущая 1*
- Хранить только дробные разряды мантиссы в 23-разрядном поле

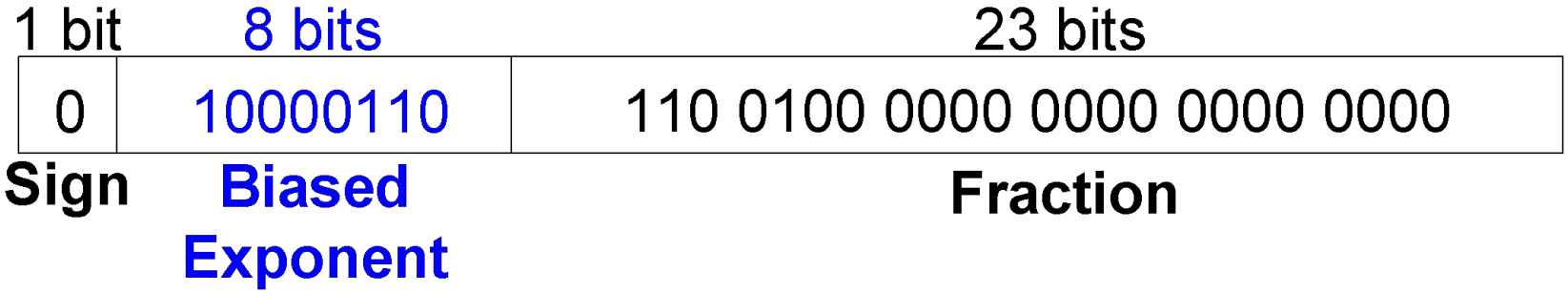


Представление с плавающей точкой 2

- *Смещенный порядок*: смещение = 127 (01111111_2)
  - Смещенный порядок = смещение + порядок
  - Порядок 7 хранится как:

$$127 + 7 = 134 = 0x10000110_2$$

- **32-разрядное (IEEE 754) представление с плавающей точкой** числа 228_{10}



в 16-ричном коде: **0x43640000**

Представление с плавающей точкой. Пример

Записать число -58.25_{10} с плавающей точкой (IEEE 754)

Представление с плавающей точкой. Пример

Записать число -58.25_{10} с плавающей точкой (IEEE 754)

$58.25_{10} = \mathbf{111010.01_2}$

1. Записать в двоичной системе и экспоненциальном представлении:

   $\mathbf{1.1101001 \times 2^5}$

3. Заполнить поля:

   Знаковый разряд: 1 (отрицательный)
   8 разрядов порядка: (127 + 5) = 132 = $\mathbf{10000100_2}$
   23 разряда мантиссы: 110 1001 0000 0000 0000 0000

1 bit	8 bits	23 bits
1	100 0010 0	110 1001 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

в 16-ричном коде: **0xC2690000**

Плавающая точка: Особые

Число	Знак	Порядок	Мантисса
0	X	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
- ∞	1	11111111	00000000000000000000000
NaN	X	11111111	Не нуль

Плавающая точка: точность

- **Одинарная точность:**
  - 32-разрядное
  - 1 знаковый разряд, 8 разрядов порядка, 23 разряда мантиссы
  - Смещение = 127

- **Двойная точность:**
  - 64- разрядное
  - 1 знаковый разряд, 11 разрядов порядка, 52 разряда мантиссы
  - Смещение = 1023

Плавающая точка:

- **Переполнение:** число слишком большое для представления
- **Потеря точности:** число слишком маленькое для представления
- **Режимы округления:**
  - Вниз – к меньшему
  - Вверх – к большему
  - К нулю (к меньшему по модулю)
  - К ближайшему
- **Пример:** округлить 1.100101 (1.578125) к числу с 3 дробными разрядами
  - Вниз: 1.100
  - Вверх: 1.101
  - К нулю: 1.100

Плавающая точка:

1. Выделить порядок числа и мантиссу
2. Присоединить ведущую 1 к мантиссе
3. Сравнить порядки
4. Выполнить сдвиг меньшей мантиссы при необходимости
5. Сложить мантиссы
6. Нормализовать мантиссы и подобрать порядки при необходимости
7. Округлить результат
8. Выполнить сборку порядка и мантиссы обратно в формат с плавающей точкой

Суммирование с плавающей точкой. Пример

Сложить числа с плавающей точкой:

0x3FC00000

0x40500000

Суммирование с плавающей точкой. Пример

1. Извлечь порядок и мантиссу

1 bit	8 bits	23 bits
0	01111111	100 0000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

1 bit	8 bits	23 bits
0	10000000	101 0000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

Для первого числа (N1): S = 0, E = 127, F = .1

Для второго числа (N2): S = 0, E = 128, F = .101

2. Присоединить ведущую 1 к мантиссе

N1: 1.1

N2: 1.101

Суммирование с плавающей точкой. Пример

3. **Сравнить порядки**

   127 – 128 = -1, поэтому сдвиг N1 вправо на 1 разряд

4. **Сдвиг меньшей мантиссы при необходимости**

   shift N1's mantissa: 1.1 >> 1 = 0.11 $(\times 2^1)$

5. **Сложить мантиссы**

$$\begin{array}{r} 0.11 \times 2^1 \\ +\ 1.101 \times 2^1 \\ \hline 10.011 \times 2^1 \end{array}$$

Суммирование с плавающей точкой. Пример

6. **Нормализовать мантиссы и подобрать порядки при необходимости**

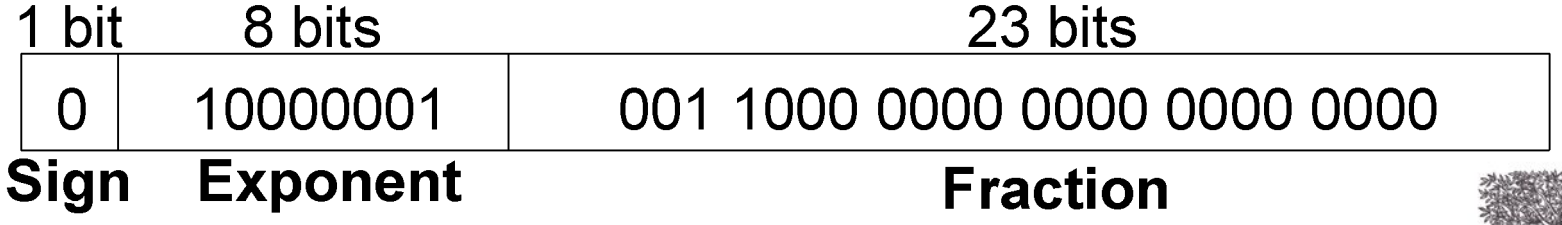
   $10.011 \times 2^1 = 1.0011 \times 2^2$

7. **Округлить результат**

   нет необходимости (уложились в 23 разряда)

8. **Выполнить сборку порядка и мантиссы обратно в формат с плавающей точкой**

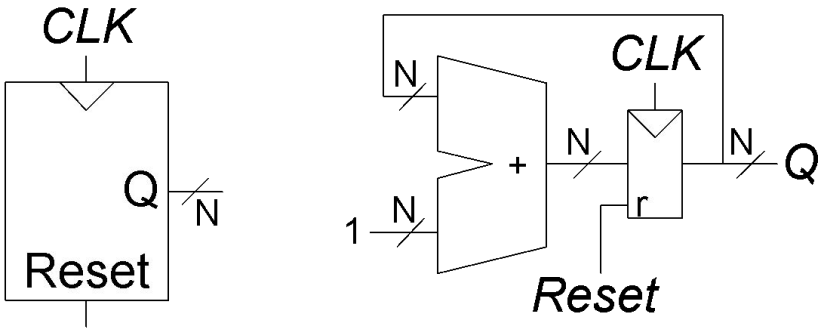
   $S = 0, E = 2 + 127 = 129 = 10000001_2, F = 001100..$



Счетчики

- Инкремент на каждом переднем фронте
- Используется в цикле для перебора всех чисел. Например,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- В примере использованы:
  - Отображение цифровых часов
  - Программный счетчик: отслеживает выполнение текущей команды

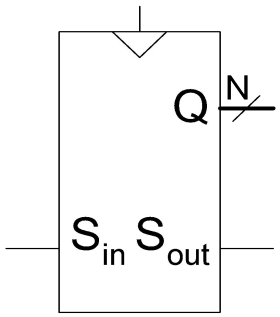
Symbol **Implementation**



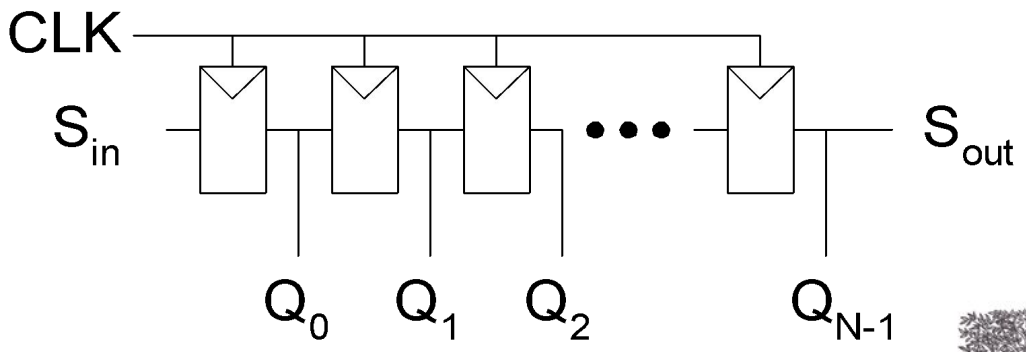
Сдвигающий регистр

- Вдвигается новый бит по переднему фронту тактового сигнала
- Выдвигается бит по переднему фронту тактового сигнала
- *Последовательно-параллельный преобразователь*: преобразует последовательный вход (S_{in}) в параллельный выход ($Q_{0:N-1}$)

Обозначение

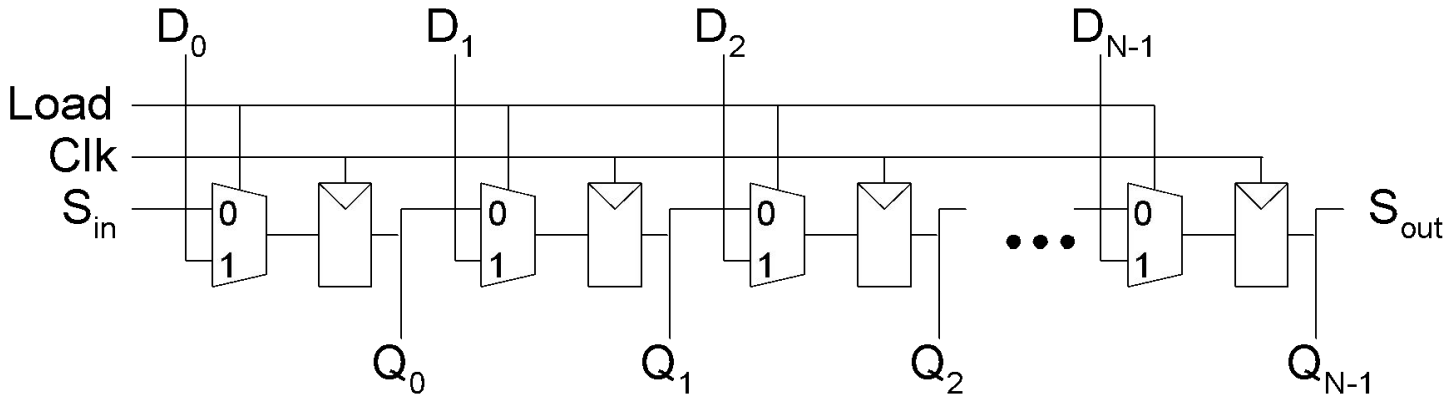


Реализация



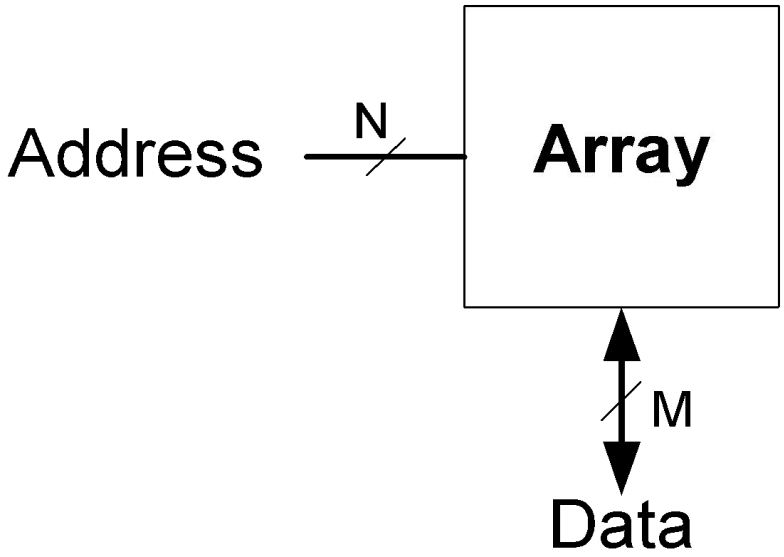
Сдвигающий регистр с параллельной загрузкой

- Когда *Load* = 1, работает как обычный *N*-разрядный регистр
- Когда *Load* = 0, работает как регистр сдвига
- Может работать как последовательно-параллельный преобразователь ($S_{in} \rightarrow Q_{0:N-1}$) или *параллельно-последовательный преобразователь* ($D_{0:N-1} \rightarrow S_{out}$)



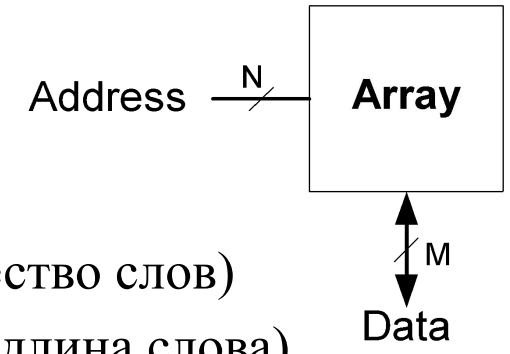
Матрицы памяти

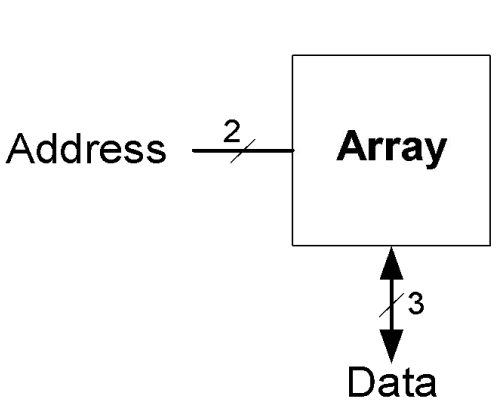
- Эффективно хранят большие объемы данных
- 3 основных типа:
  - Динамическое оперативное запоминающее устройство (ОЗУ) (DRAM)
  - Статическое оперативное запоминающее устройство (ОЗУ) (SRAM)
  - Постоянное запоминающее устройство (ПЗУ), память только для чтения (ROM)
- *M*-разрядное значение данных считывается/записывается по уникальному *N*-разрядному адресу

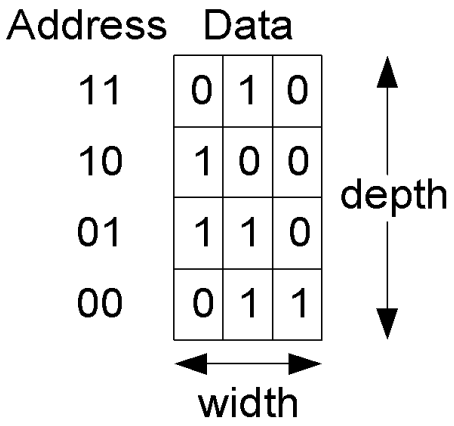


Матрицы памяти

- 2-мерная матрица битовых ячеек
- Каждая битовая ячейка хранит 1 бит
- *N* адресных битов и *M* битов данных:
  - 2^N строк и *M* столбцов
  - **Глубина (Depth):** количество строк (количество слов)
  - **Ширина (Width):** число столбцов (размер, длина слова)
  - **Размер матрица:** depth × width = $2^N \times M$

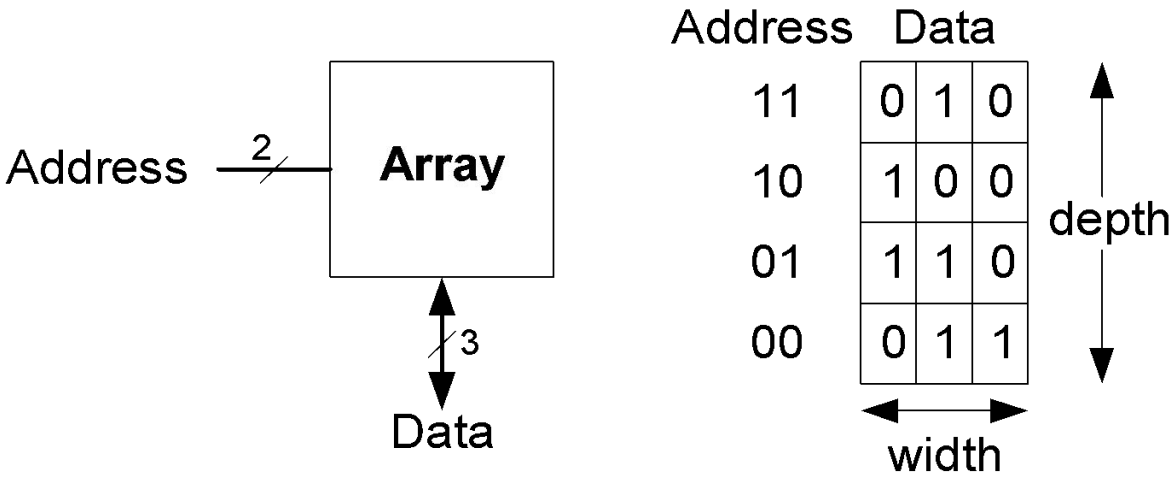




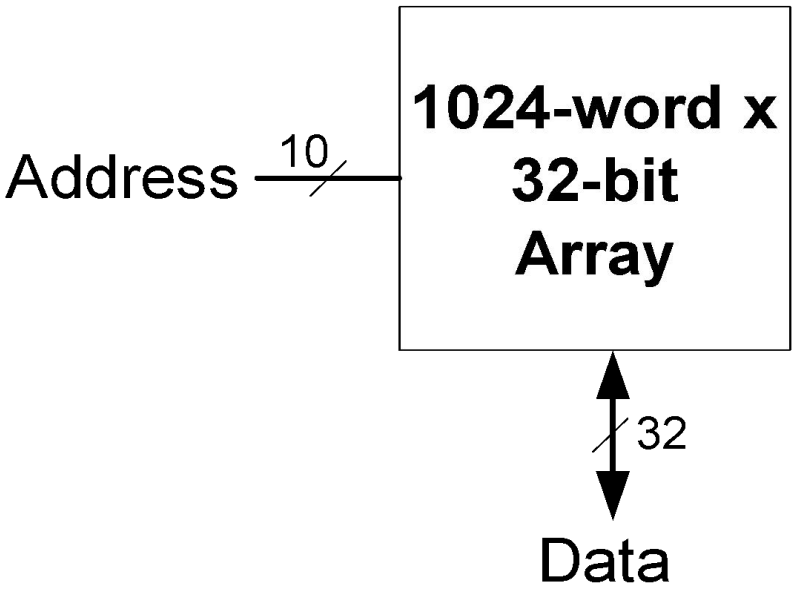


Пример матрицы памяти

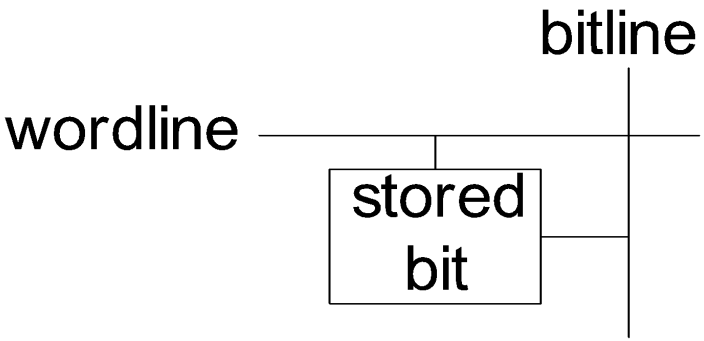
- $2^2 \times 3$-битовая матрица
- Количество слов: 4
- Длина слова: 3 бита
- Например, 3-битовое слово 100 хранится по адресу 10

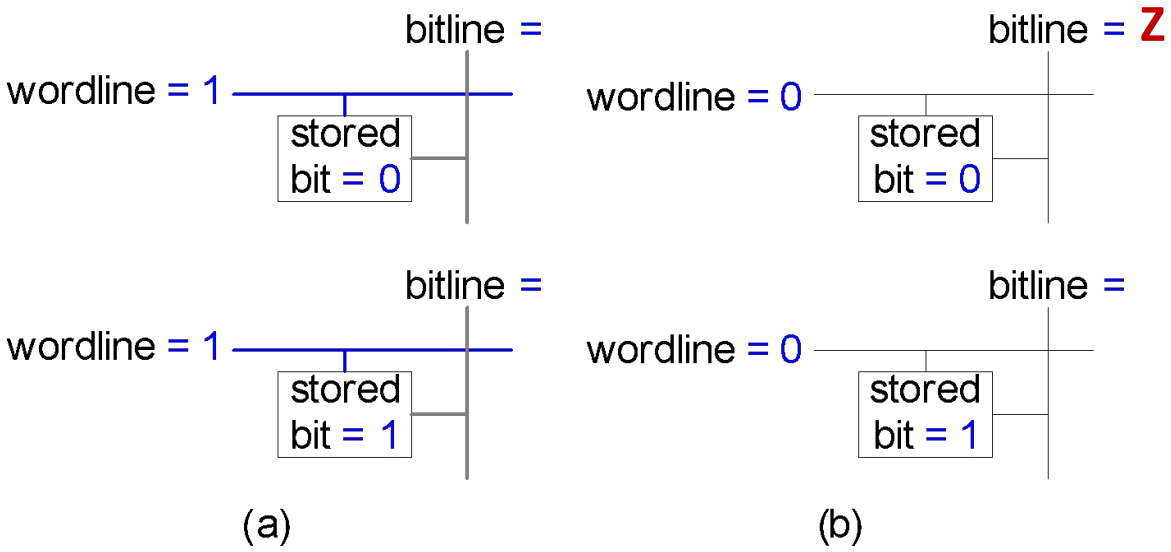


Матрицы памяти

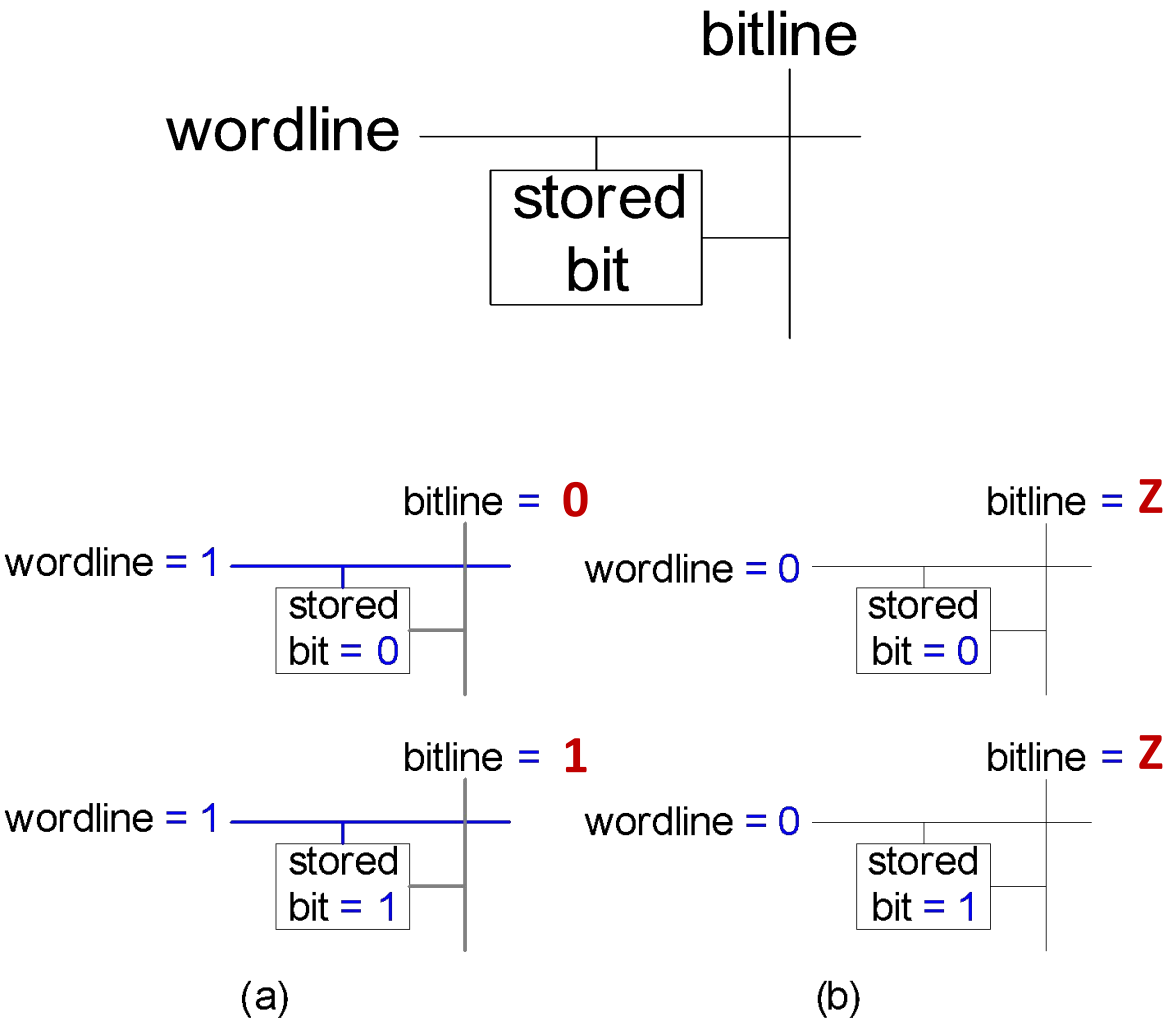


Запоминающие элементы матрицы памяти



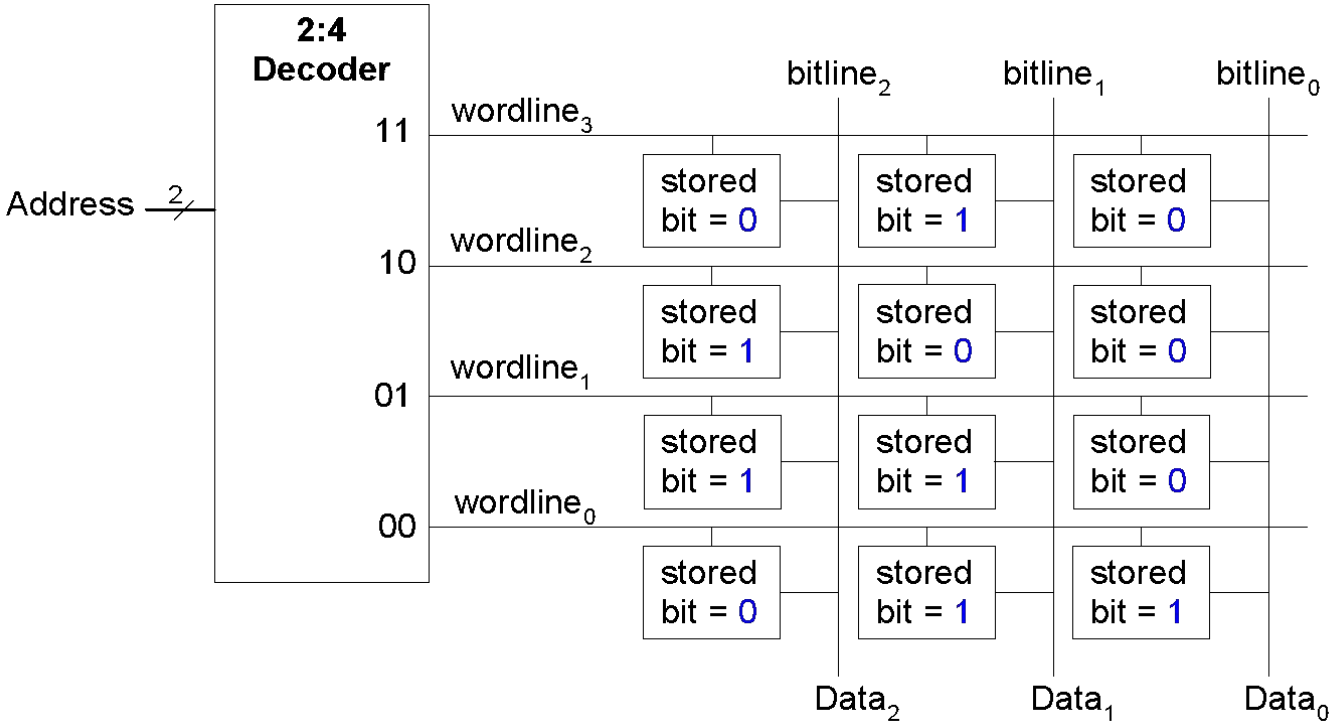


Запоминающие элементы матрицы памяти



Матрица памяти

- **Линия выборки слов (worldline):**
  - формирует сигнал разрешения для выбора строки
  - в матрице памяти только одна строка может читаться/записываться
  - соответствует уникальному адресу
  - только одна линия выборки слов может быть активна



Типы памяти

- с произвольным доступом, оперативная память (RAM, ОЗУ): энергозависимая (**volatile)**
- Память только для чтения (ROM, ПЗУ): энергонезависимая (**nonvolatile)**

RAM, ОЗУ: оперативная

- **Энергозависимая:** содержимое памяти теряется при отключении электропитания
- Быстрые чтение и запись
- Основная память в компьютере – RAM (DRAM)

Исторически сложилось название «память с произвольным доступом», так как в ней доступ к любому слову данных для чтения или записи осуществляется всегда за одно и то же время (в отличие от памяти с последовательным доступом, такой, например, как магнитная лента)

ROM, ПЗУ: Память только для чтения

- **Энергонезависимая:** содержимое памяти сохраняется при отключении электропитания
- Чтение быстрое, но запись невозможна или медленная
- Флэш память в видеокамерах, флэш-накопителях – ROM

Исторически сложилось название «память только для чтения,ROM», поскольку информация в нее могла быть записана только во время ее производства или путем пережигания плавких перемычек. После того, как память была сконфигурирована, она не могла быть записана снова. Теперь это уже не так.

Типы RAM

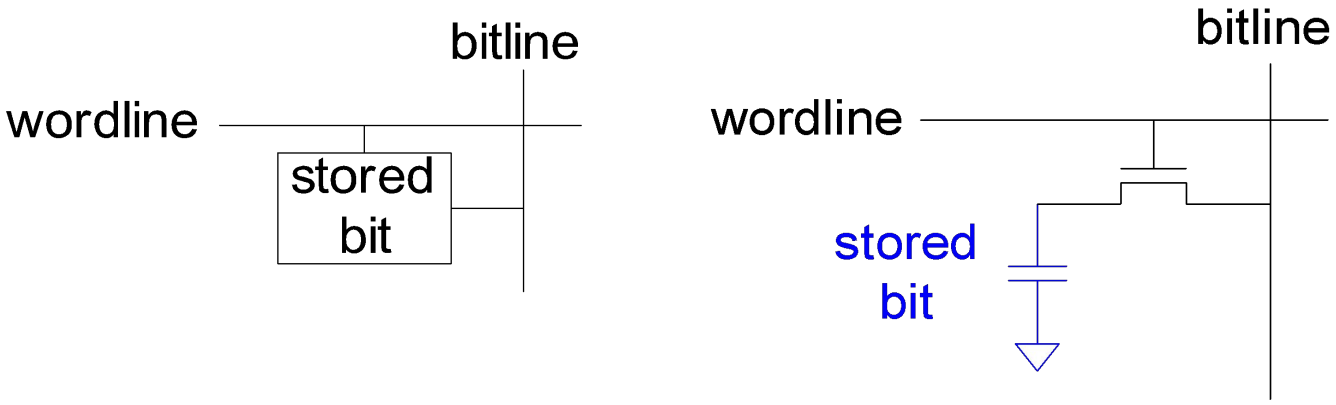
- **DRAM** (динамическое ОЗУ - Dynamic random access memory)
- **SRAM** (Статическое ОЗУ - Static random access memory)
- Отличаются способом хранения данных:
  - DRAM использует конденсатор
  - SRAM использует инверторы с перекрёстными обратными связями

Роберт Деннард, 1932 -

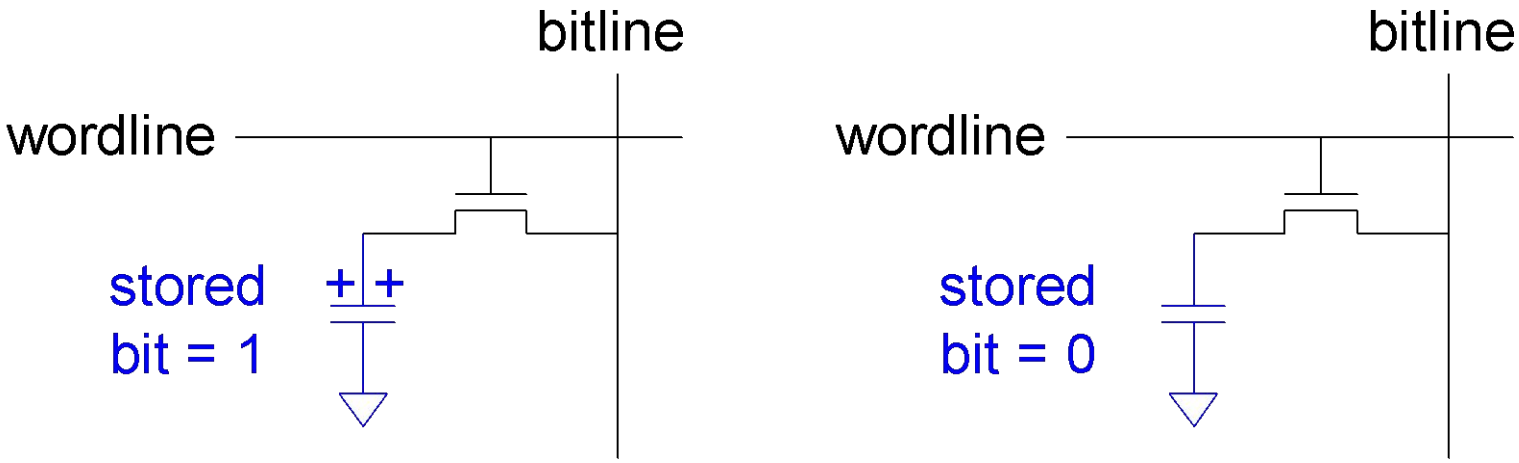
- Изобрел DRAM в 1966, IBM
- Многие были настроены скептически к работоспособности его идеи
- С середины 1970-х DRAM используется практически во всех компьютерах

DRAM

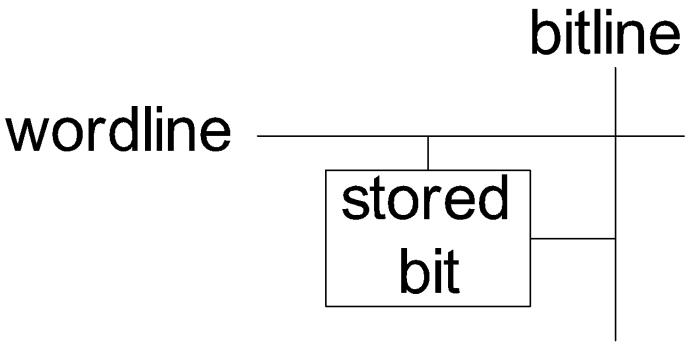
- Биты данных сохраняются в конденсаторах
- *Динамическая,* потому что значение должно быть обновлено (перезаписано) как периодически, так и после считывания:
  - Утечка заряда конденсатора разрушает значение
  - Чтение уничтожает сохраненное значение

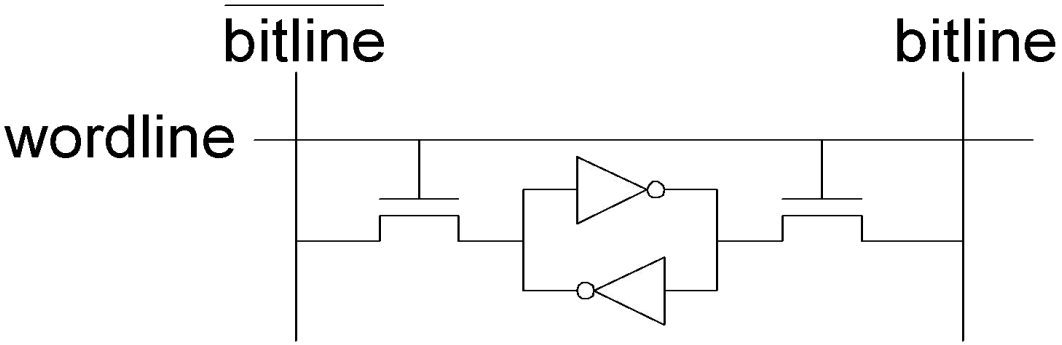


DRAM

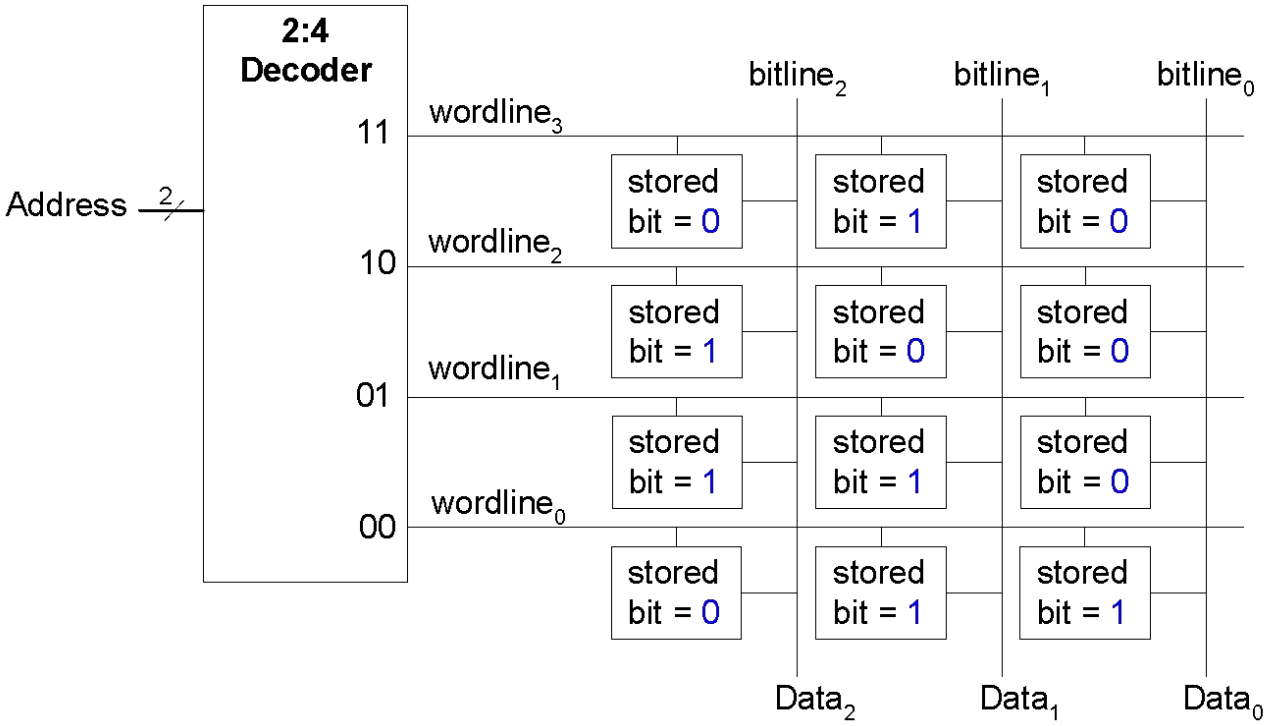

bitline
wordline
stored
bit = 1
+ +
bitline
wordline
stored
bit = 0

SRAM

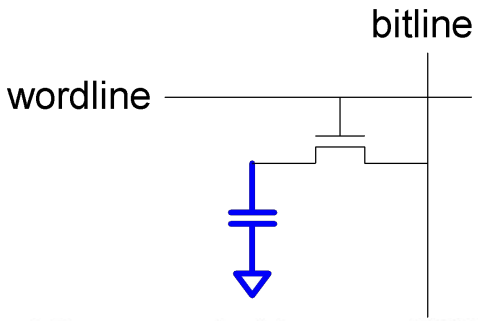




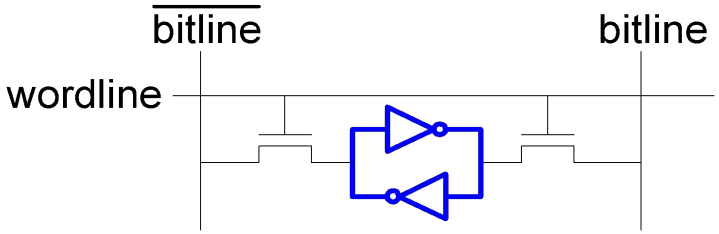
Матрицы памяти. Обзор



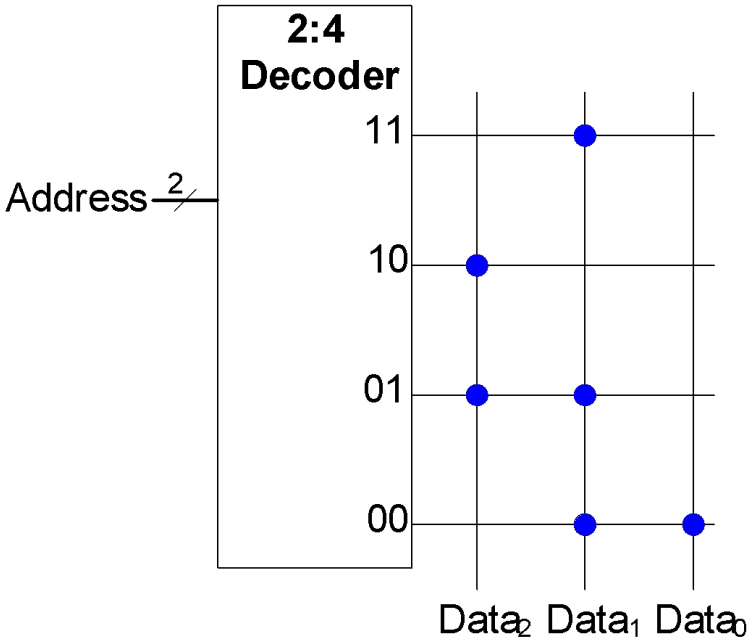
DRAM bit cell:

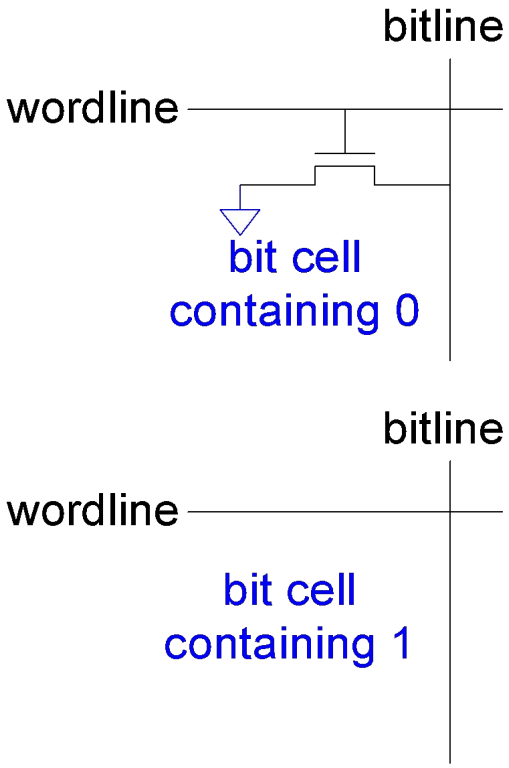


SRAM bit cell:



ПЗУ (ROM): Точечная нотация

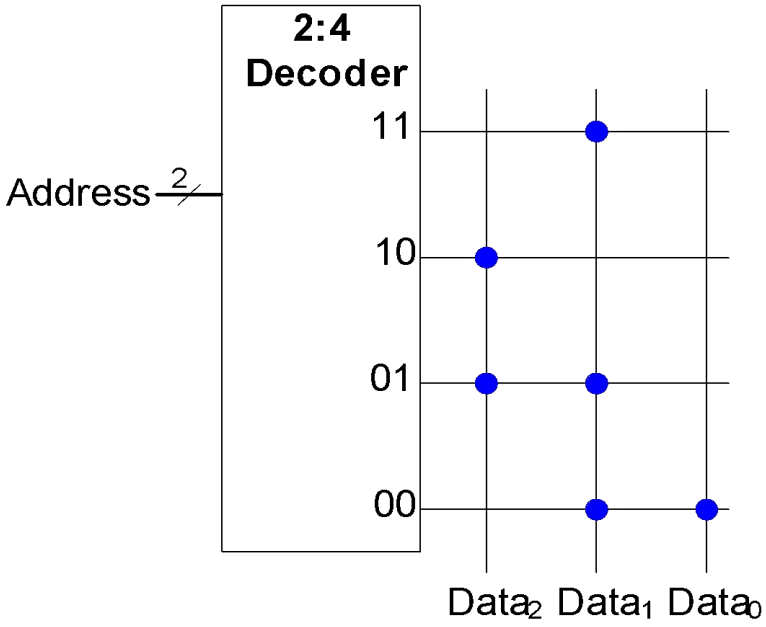




Фуджио Масуока, 1944 -

- Разрабатывал память и быстродействующие схемы для Toshiba, 1971-1994
- Изобрел флэш-память в процессе самостоятельной работы, проводимой по ночам и в выходные дни в конце 1970-х
- Процесс стирания памяти напомнил ему о вспышке камеры
- Toshiba медленно коммерциализировала идею; Intel была первой на рынке в 1988
- Рынок флэш-памяти растет на $ 25 млрд в год

Хранение данных в ПЗУ (ROM)

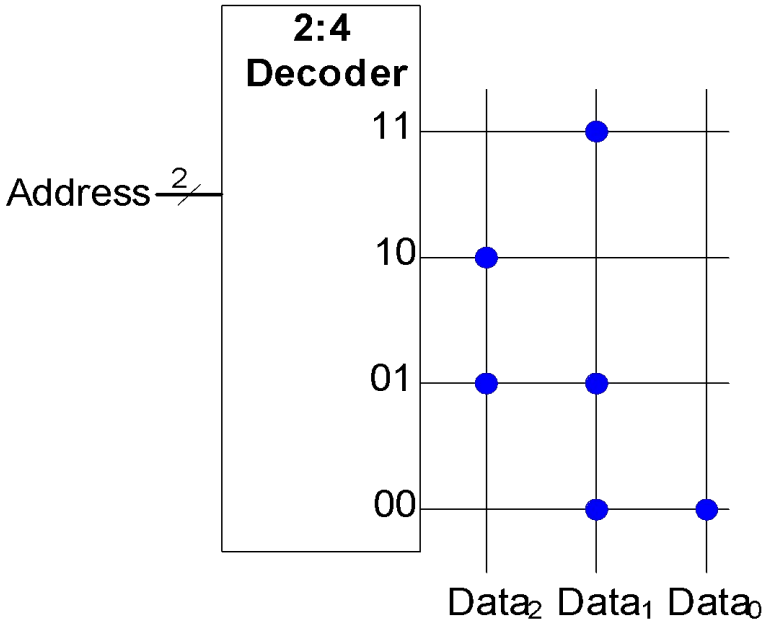


Address	Data		
11	0	1	0
10	1	0	0
01	1	1	0
00	0	1	1

depth

width

Логические функции и ПЗУ



$$Data_2 = A_1 \oplus A_0$$

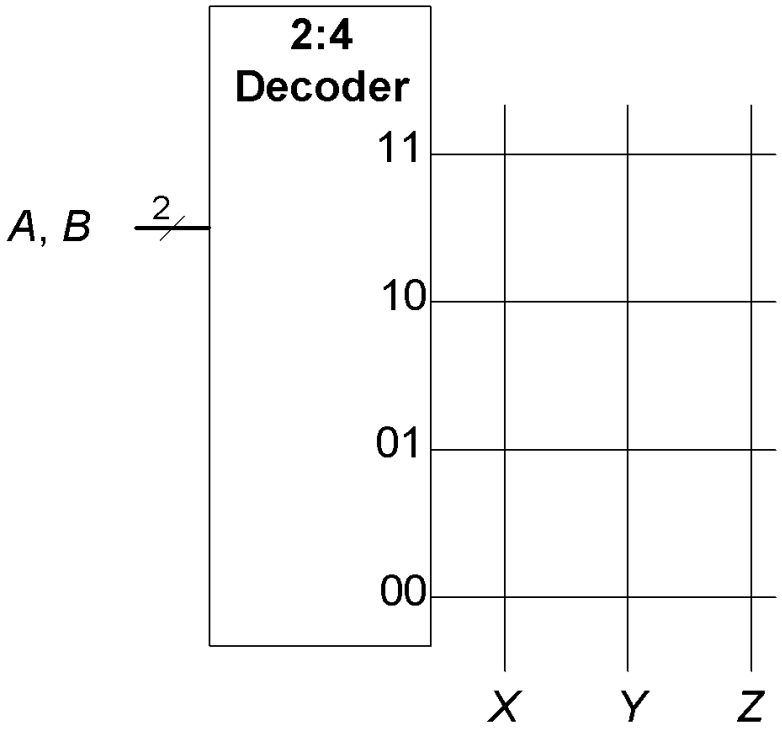
$$Data_1 = \overline{A}_1 + A_0$$

$$Data_0 = \overline{A}_1 \overline{A}_0$$

Пример: Логика на основе ПЗУ

Реализовать следующие логические функции, используя ПЗУ $2^2 \times 3$-бит:

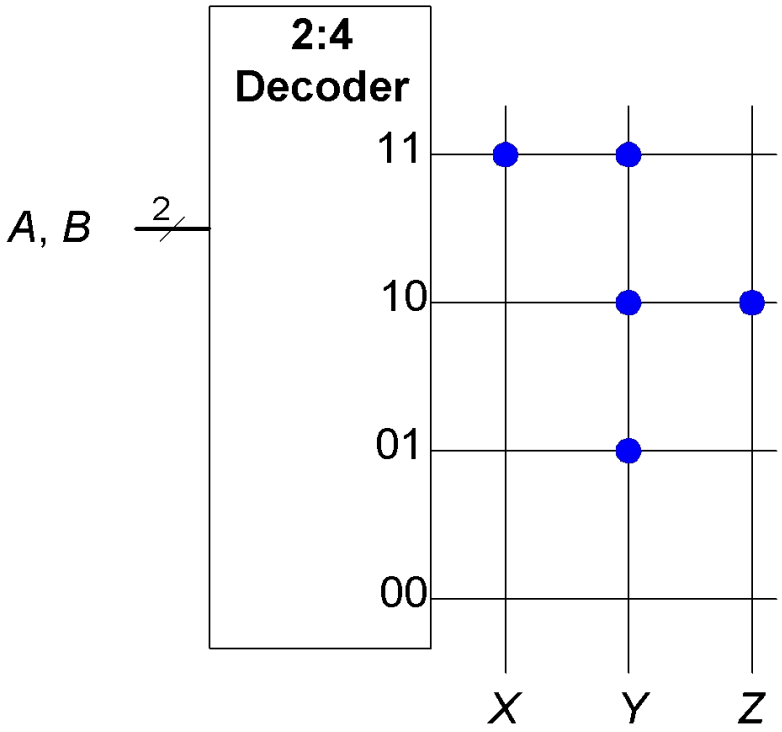
- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$



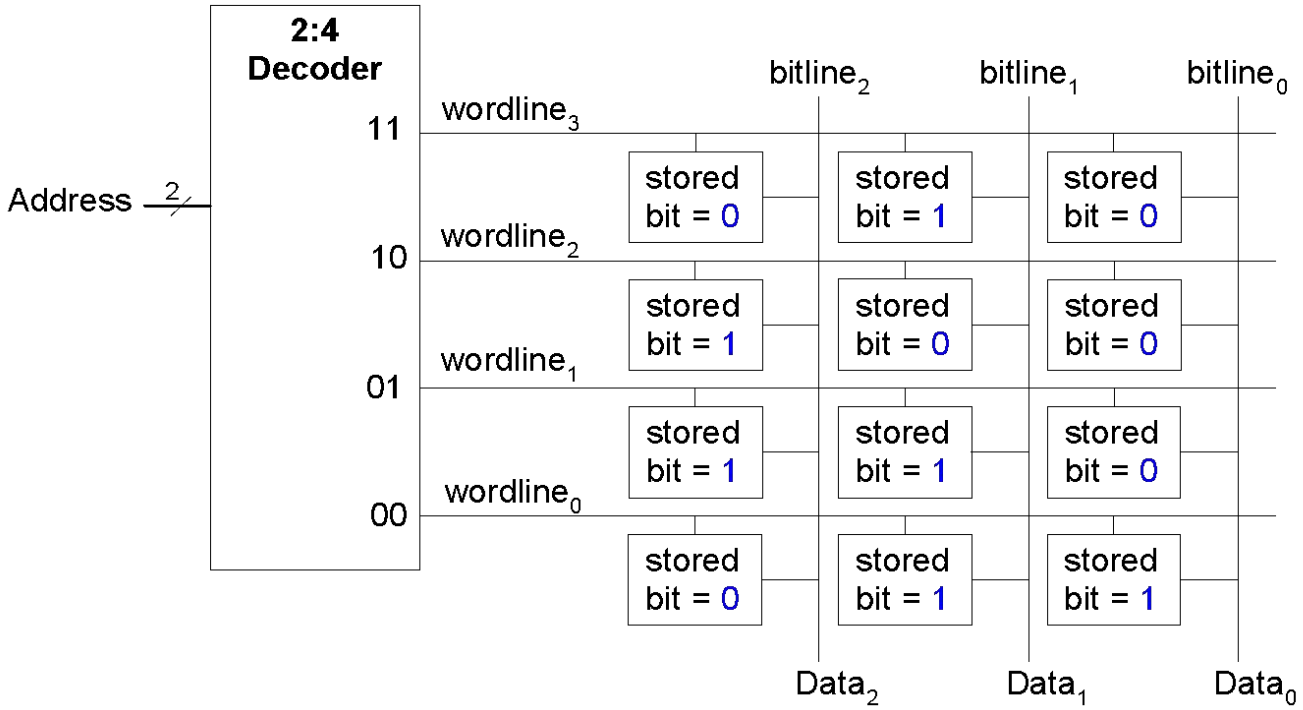
Пример: Логика на основе

Реализовать следующие логические функции, используя ПЗУ $2^2 \times 3$-бит:

- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$



Логика на основе любой матрицы памяти



$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = \overline{A}_1 + A_0$$

$$Data_0 = \overline{A}_1 \overline{A}_0$$

Логика на основе матрицы памяти

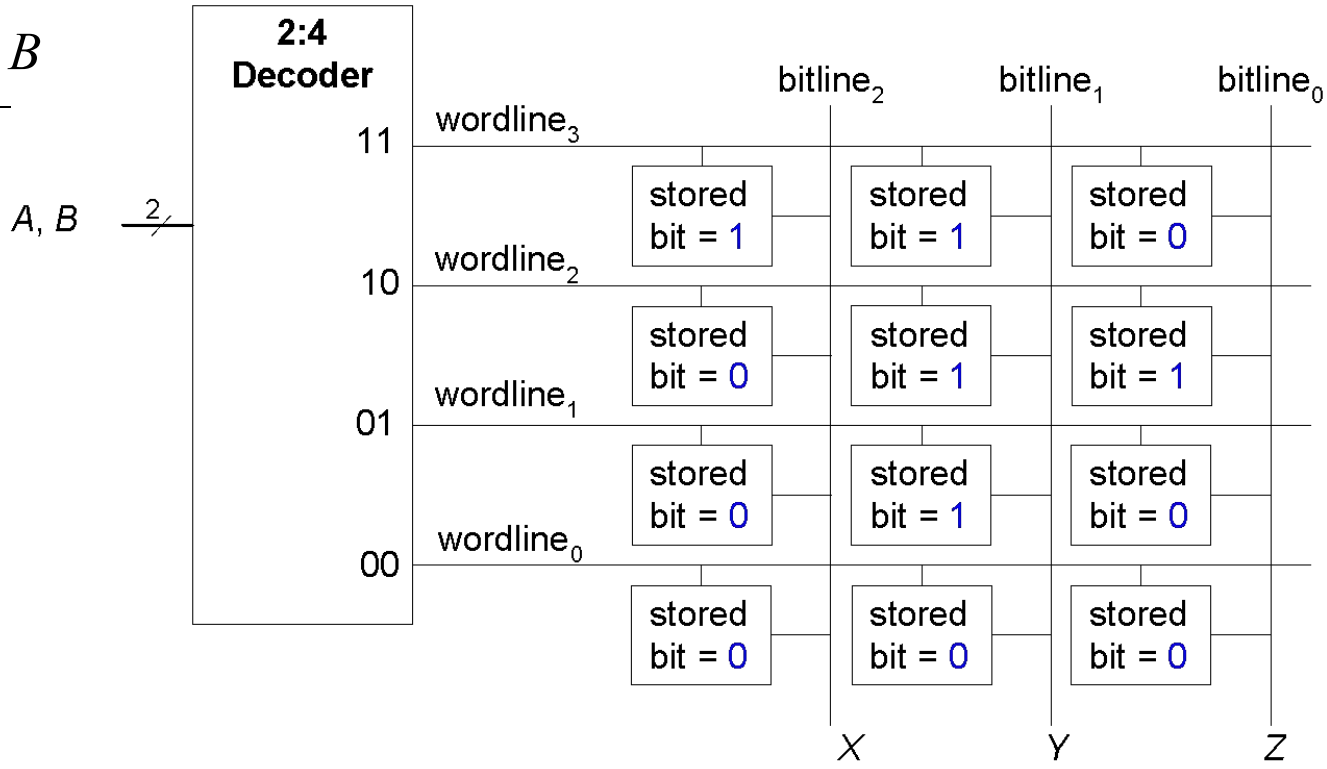
Реализовать следующие логические функции, используя $2^2 \times 3$-битовую матрицу памяти:

- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$

Логика на основе матрицы памяти

Реализовать следующие логические функции, используя $2^2 \times 3$-битовую матрицу памяти:

- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$

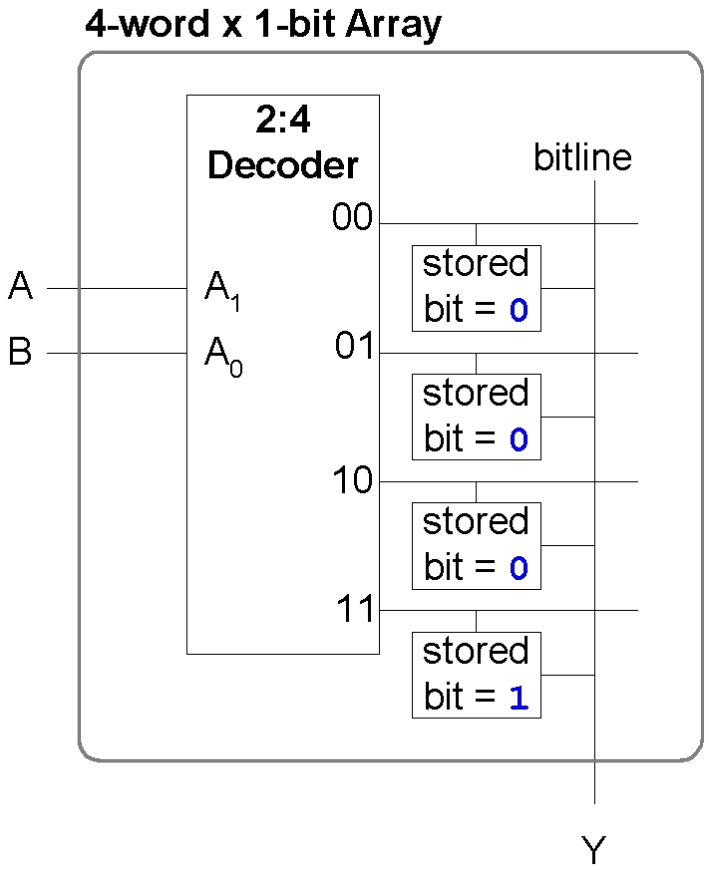


Логика на основе матрицы памяти

Таблицы преобразования (*lookup tables,* LUTs), каждой входной комбинации соответствует некоторое состояние выхода

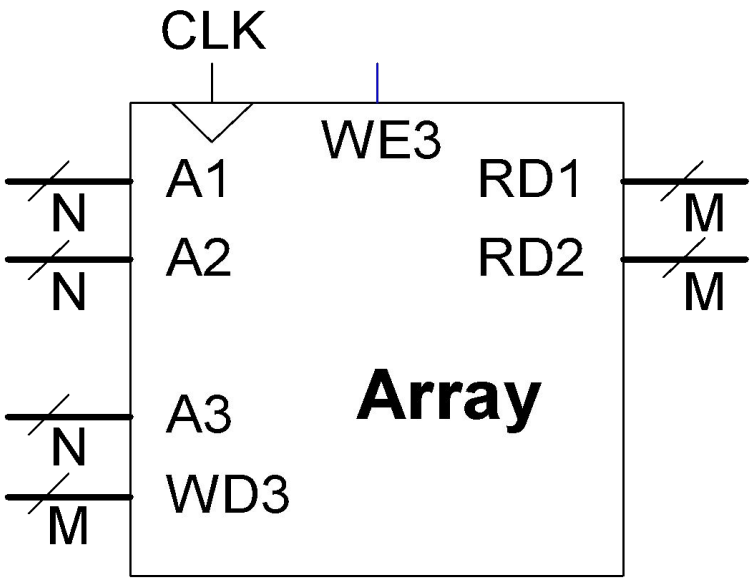
Truth Table

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



Многопортовая память

- **Порт:** пара адрес/данные
- 3-портовая память
  - 2 порта чтения (A1/RD1, A2/RD2)
  - 1 порт записи (A3/WD3, WE3 разрешает запись)
- **Регистровый файл:** малая многопортовая память



SystemVerilog: матрицы памяти

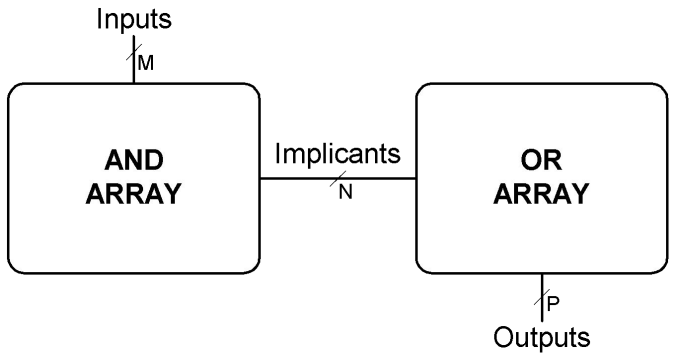
```
// 256 x 3 модуль памяти с одним портом чтения/записи
module dmem( input  logic       clk, we,
             input  logic[7:0]  a
       input  logic [2:0] wd,
             output logic [2:0] rd);

  logic  [2:0] RAM[255:0];

  assign rd = RAM[a];

  always @(posedge clk)
    if (we)
      RAM[a] <= wd;
endmodule
```

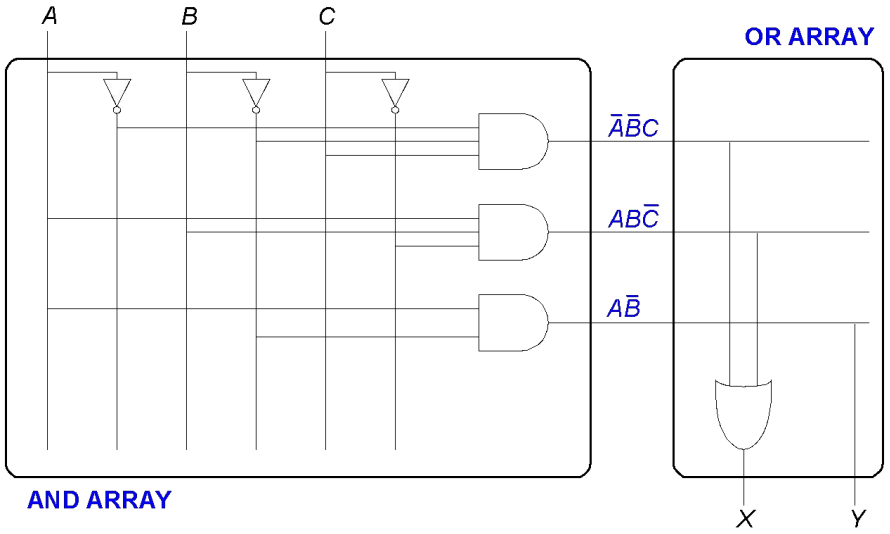
Матрицы логических

- **PLAs** (PLM, Programmable logic arrays) – программируемая логическая матрица
  - AND матрица, затем OR матрица
  - Только комбинационная логика
  - Фиксированные внутренние соединения
- **FPGAs** (Field programmable gate arrays) – Программируемая пользователем матрица логических элементов
  - Массив конфигурируемых логических блоков CLB
  - Комбинационная и последовательностная логика
  - Программируемые внутренние соединения

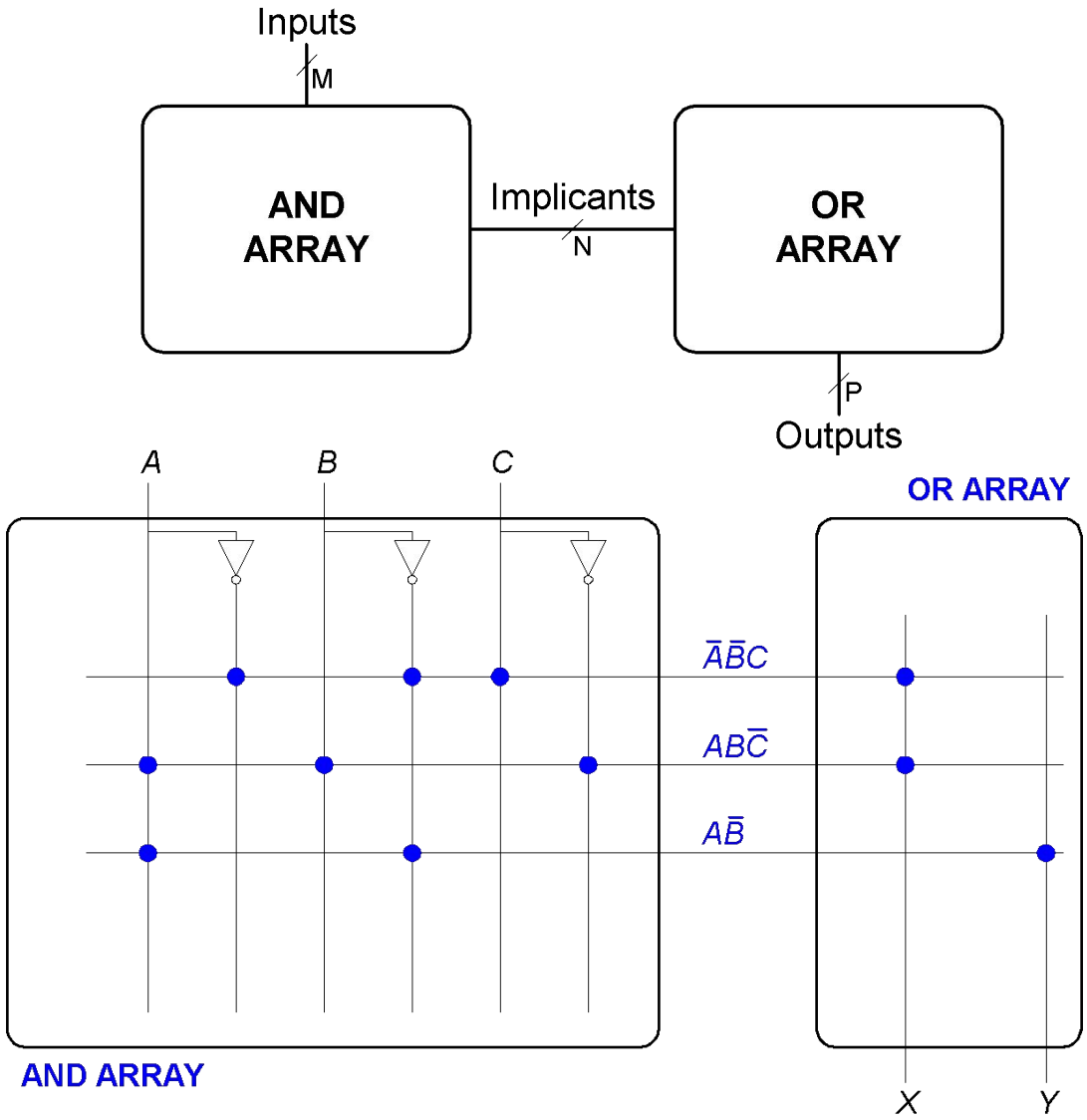
Программируемые логические матрицы

- $X = \bar{A}\bar{B}C + AB\bar{C}$
- $Y = A\bar{B}$





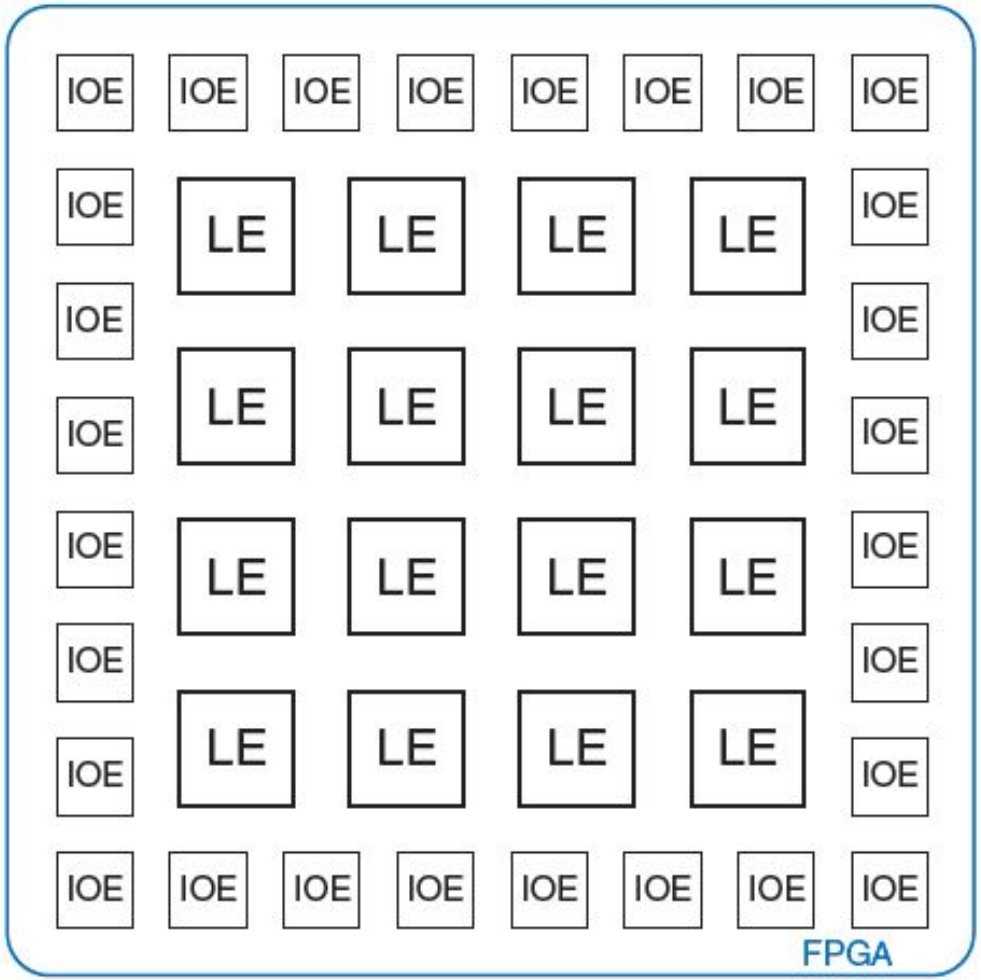
PLA (ПЛМ): Точечная нотация



FPGA: Программируемая пользователем матрица логических элементов

- Состоит из:
  - **LE** (логических элементов): реализуют логику
  - **IOE** (Элементов ввода/вывода): интерфейс с внешним миром
  - **Programmable interconnection:** программируемые соединения связывают логические элементы с элементами ввода/вывода
  - Некоторые FPGAs включают и другие блоки, такие как умножители и память RAMs

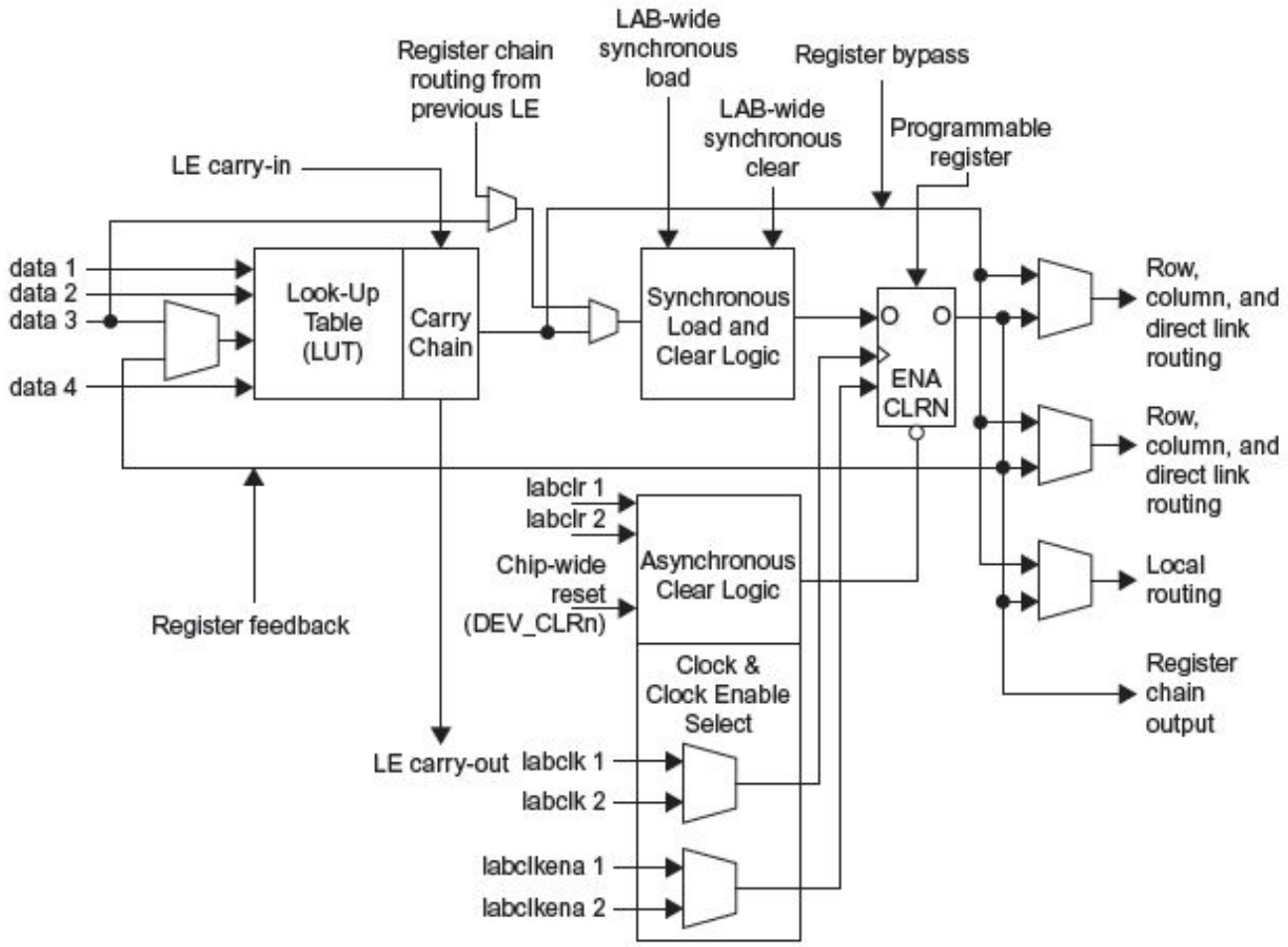
Обобщенная структура FPGA



LE: Логические элементы

- Состоят из:
  - **Таблиц преобразования (LUT)**: реализуют комбинационную логику
  - **Триггеров:** реализуют последовательностную логику
  - **Мультиплексоров:** соединяют таблицы преобразования (LUT) и триггеры

Altera Cyclone IV LE



Altera Cyclone IV LE

- Конфигурируемые логические блоки (CLB) Spartan имеют:
  - 1 четырехвходовую LUT
  - 1 регистровый выход
  - 1 комбинационный выход

Конфигурация ЛБ. Пример

Покажите, как сконфигурировать логические блоки Cyclone IV для выполнения следующих функций:

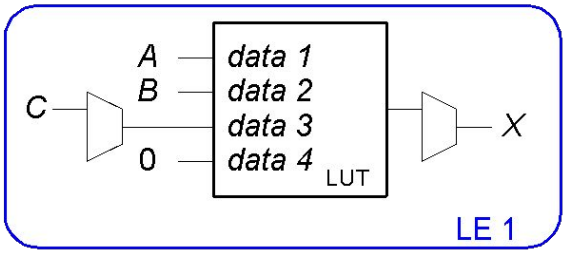
- $X = \overline{A}B\overline{C} + AB\overline{C}$
- $Y = A\overline{B}$

Конфигурация ЛБ. Пример

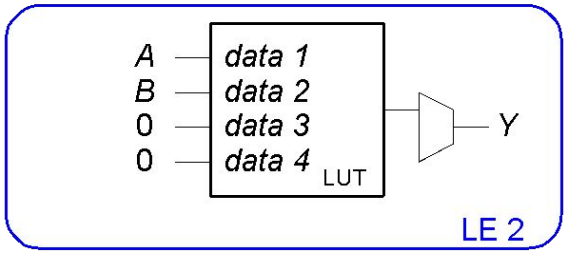
Покажите, как сконфигурировать ЛБ Cyclone IV для выполнения следующих функций:

- $X = \overline{A}\,\overline{B}C + AB\overline{C}$
- $Y = A\overline{B}$

(A) data 1	(B) data 2	(C) data 3	data 4	(X) LUT output
0	0	0	X	0
0	0	1	X	1
0	1	0	X	0
0	1	1	X	0
1	0	0	X	0
1	0	1	X	0
1	1	0	X	1
1	1	1	X	0



(A) data 1	(B) data 2	data 3	data 4	(Y) LUT output
0	0	X	X	0
0	1	X	X	0
1	0	X	X	1
1	1	X	X	0



FPGA. Последовательность проектирования

Используя средства САПР (Altera's Quartus II):

- **Описать проект** с использованием редактора принципиальных схем или HDL
- **Выполнить моделирование** проекта
- **Выполнить синтез** проекта и его имплементацию в FPGA
- **Загрузить конфигурацию** в FPGA
- **Протестировать** проект