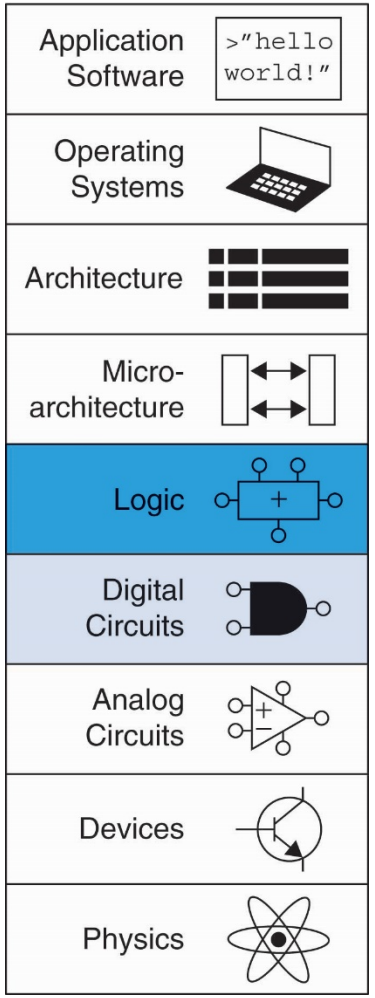
Розділ 5. Теми:

- Вступ
- Арифметичні схеми
- Подання чисел
- Послідовнісні функціональні блоки
- Матриці пам’яті
- Матириці логічних елементів

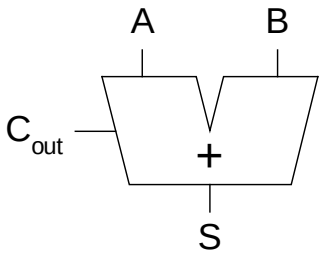


Вступ.

- **Цифрові функціональні блоки:**
  - Логічні елементи, мультиплексори, декодери, регістри, схеми арифметики, лічильники, матриці памя’ті і матриці логічних елементів
- **Функціональні блоки створюються за принципом ієрархічності, модульності і регулярності проектованих систем**
  - Ієрархія більш простих компонентів
  - Строго визначені інтерфейси і функції
  - Регулярна структура легко масштабується в системи різних розмірів
- **Подібні функціональні блоки будуть використовуватися в розділі 7 для проектування мікропроцесора**

Одноразрядний суматор.

Half Adder

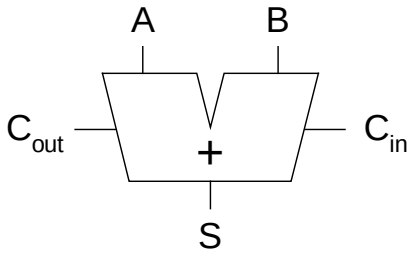


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = AB$

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

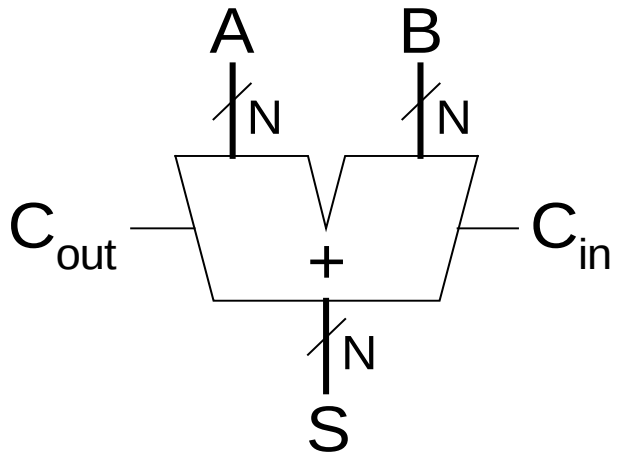
$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + AC_{in} + BC_{in}$

Багаторозрядні суматори

- Типи поширення перенесень:
  - Послідовний (повільний)
  - Прискорений груповий (швидкий)
  - Префіксний (самий швидкий)
- Два останніх типи використовуються в багаторозрядних суматорах, але їх реалізація вимагає додаткових апаратних витрат

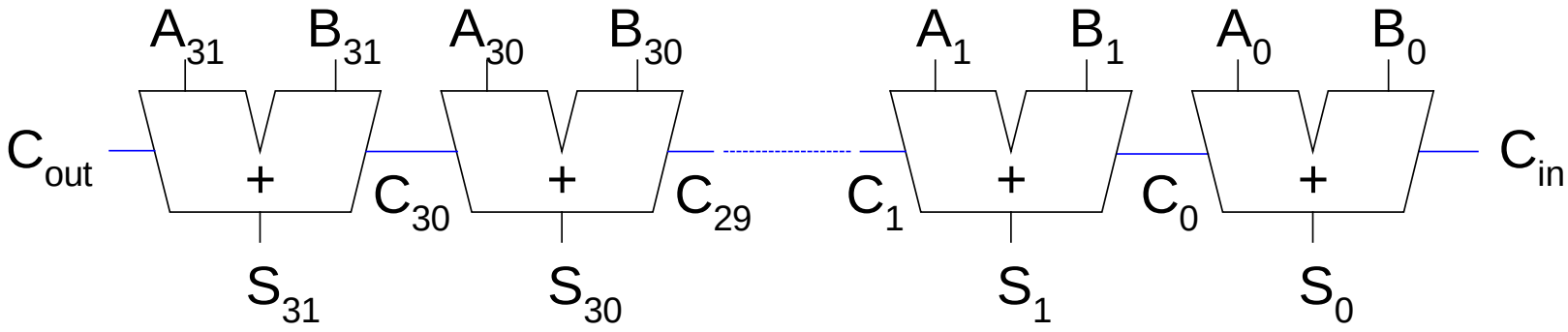
Умовне позначення



Суматор з послідовним перенесенням

- Ланцюг з одноразрядних суматорів
- Перенесення відбувається через увесь ланцюг
- Недолік: **поступове додавання**
- Затримка суматора

$$t = N \times t_{1\text{-го півсуматора}}$$



Суматор з прискореним груповим перенесенням

- Визначити значення перенесень (C_{out}) у кожному блоці *k*-розрядного суматора, використовуючи сигнали *generate i propagate*
- **Деякі визначення:**

  – Розряд *i* формує перенесення або шляхом його генерування (*generating*) або шляхом поширення (*propagating*) перенесення із свого відповідного входу на свій вихід

  – Генерування (G_i) і поширення (P_i) сигналів для кожного розряду:

  • Розряд *i* будет генерувати перенесення, якщо A_i і B_i (обидва) дорівнюють 1.

$$G_i = A_i B_i$$

  • Розряд *i* буде поширювати перенесення від відповідного входу до відповідного виходу, якщо A_i або B_i дорівнює 1.

$$P_i = A_i + B_i$$

  • Перенесення розряду *i* (C_i): $C_i = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1}$

Додавання з прискореним груповим перенесенням

- **Крок 1:** Обчислити G_i і P_i для всіх розрядів
- **Крок 2:** Обчислити G і P для всіх k-бітових блоків суматора
- **Крок 3:** Перенесення C_{in} поширюється через всі k-бітові блоки генерування/поширення

Суматор з прискореним груповим перенесенням.

- **Приклад:** 4-розрядні блоки ($G_{3:0}$ и $P_{3:0}$) :

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

$$P_{3:0} = P_3 P_2 P_1 P_0$$

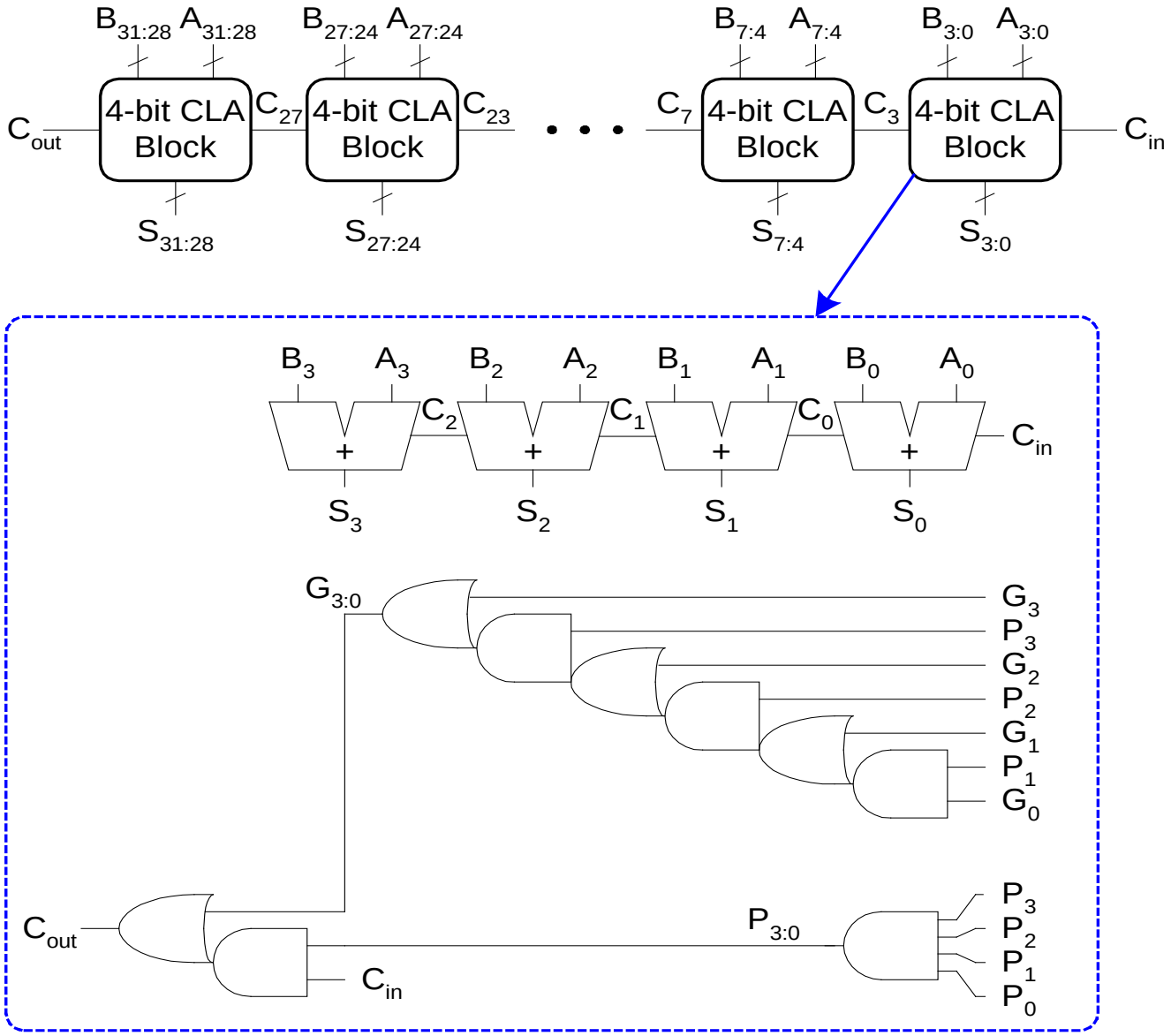
- **У загальному випадку,**

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

$$P_{i:j} = P_i P_{i-1} P_{i-2} P_j$$

$$C_i = G_{i:j} + P_{i:j} C_{i-1}$$

32-розрядний суматор з прискореним груповим перенесенням з 4-розрядними блоками



Затримки суматора з прискореним груповим перенесенням

Для N-розрядного суматора з прискореним груповим перенесенням з k-розрядними блоками:

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

– t_{pg} : затримка генерування всіх P_i, G_i

– t_{pg_block} : затримка генерування всіх $P_{i:j}$, $G_{i:j}$

– t_{AND_OR} :затримка тракту вхід C_{in} - вихід C_{out} з елементів І/АБО в k-розрядному блоці суматора з прискореним груповим перенесенням

N-розрядний суматор з прискореним груповим перенесенням практично завжди більш швидший, ніж суматор з послідовним перенесенням для $N > 16$

Префіксний суматор

- Обчислює перенесення на вході (C_{i-1}) кожного розряду, потім обчислює суму:

$$\boldsymbol{S_i = (A_i \oplus B_i) \oplus C_i}$$

- Обчислює G і P для 1-, 2-, 4-, 8-розрядів блоків, до тих пір, поки не стануть відомі всі переносення G_i (вхідні перенесення всіх розрядів)
- Кількість каскадів $\log_2 N$

Префіксний суматор

- Перенесення на вході або генеруються *для даного розряду* або поширюються від попереднього.
- Розряд -1 відповідає C_{in}, тоді

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

- Значення перенесення на вході розряду *і дорівнює значенню перенесення на виході розряду i-1*:

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

  $G_{i-1:-1}$: сигнал генерування блоку розрядів від *i*-1 до -1
- Вираз для суми:

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

- **Мета:** швидке обчислення $G_{0:-1}$, $G_{1:-1}$, $G_{2:-1}$, $G_{3:-1}$, $G_{4:-1}$, $G_{5:-1}$, … (які називаються ***префіксами***)

Префіксний суматор

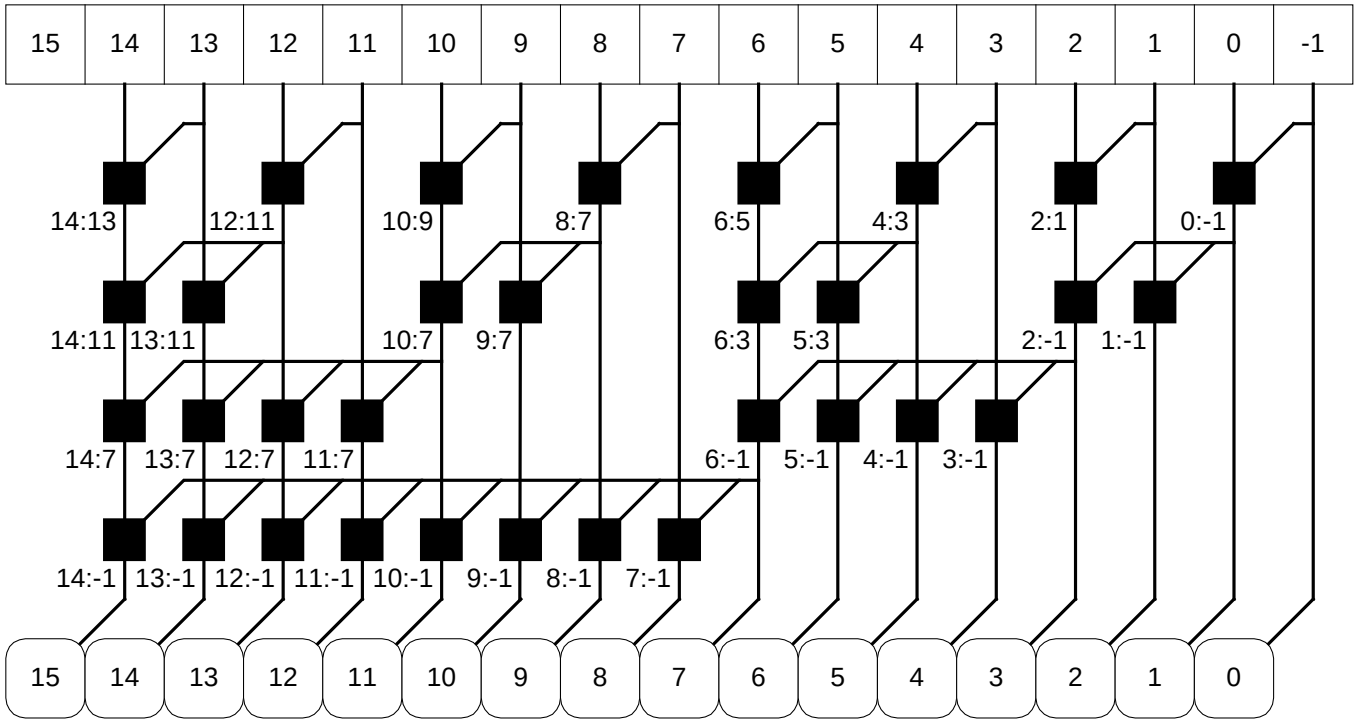
- Сигнали генерування і поширення блоку, охоплюючого розряду *i*:*j*:

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

$$P_{i:j} = P_{i:k} P_{k-1:j}$$

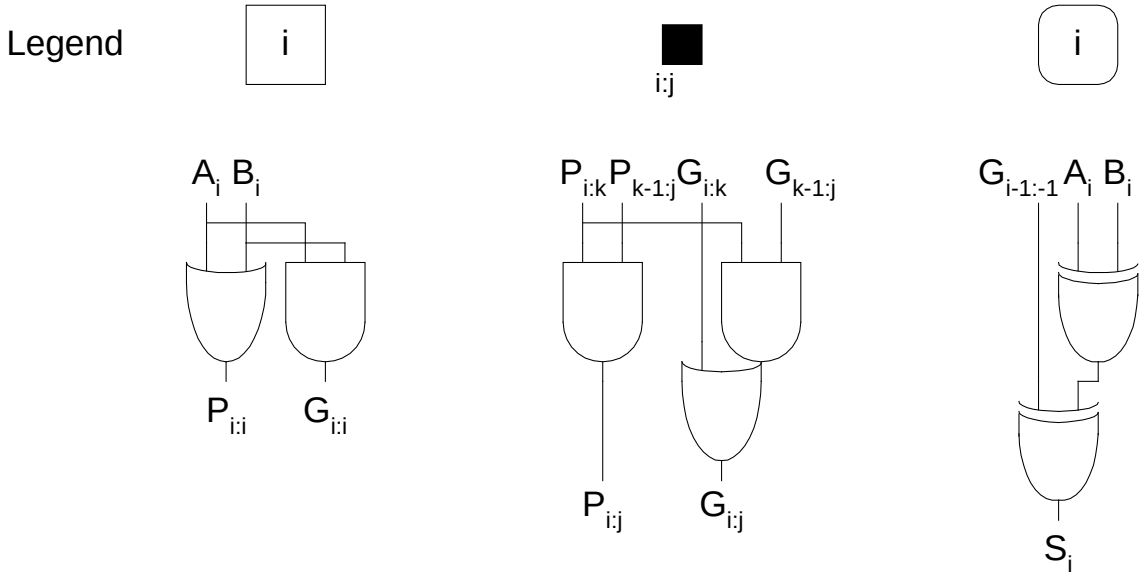
- Більш детально:
  - **Генерування:** блок *i*:*j* генерує перенесення, якщо:
    - старші розряди (*i*:*k*) генерують перенесення або
    - старші розряди поширюють перенесення, згенероване у молодших розрядах (*k*-1:*j*)
  - **Поширення:** блок *i*:*j* поширює перенесення, якщо і старші і молодші розряди *поширюють перенесення*

Схема префіксного суматора



Затримка префіксного суматора

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



t_{pg}: затримка формування P_i G_i (елементи І або АБО)

t_{pg_prefix}: затримка чорної префіксної комірки (елементи І-АБО)

Порівняння затримок суматорів

Порівняти затримки: 32-розрядний суматор з послідовним перенесенням, суматор з прискореним груповим перенесенням і префіксний суматор

- суматор з прискореним груповим перенесенням містить 4-розрядні блоки
- затримка 2-входового вентиля = 100 ps; затримка повного суматора = 300 ps

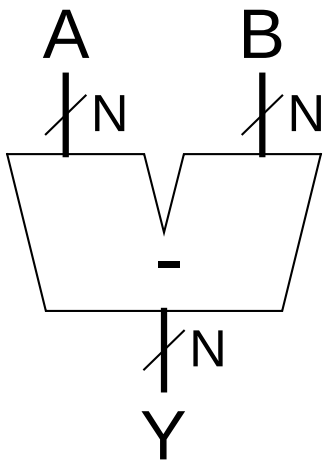
$$t_{\text{ripple}} = Nt_{FA} = 32(300\text{ ps})$$
$$= \mathbf{9.6\ ns}$$

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

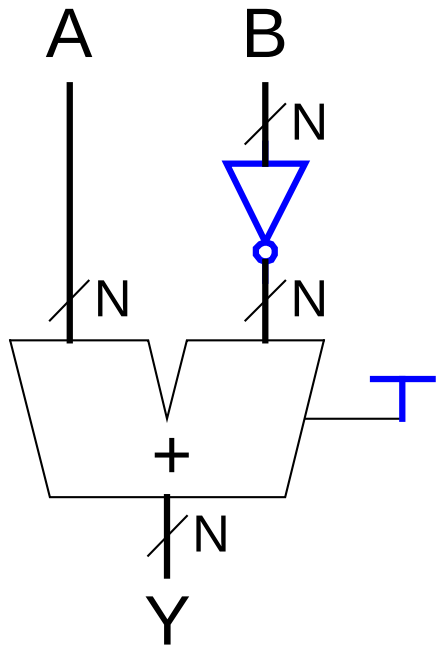
$$t_{PA} = t_{pg} + \log_2 N(t_{pg\_\text{prefix}}) + t_{\text{XOR}}$$
$$= [100 + \log_2 32(200) + 100]\text{ ps}$$
$$= \mathbf{1.2\ ns}$$

Пристрій віднімання

Symbol

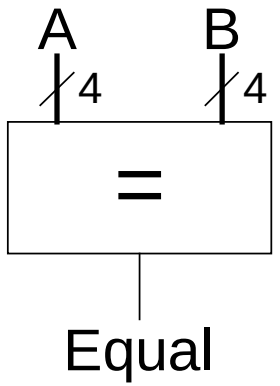


Implementation

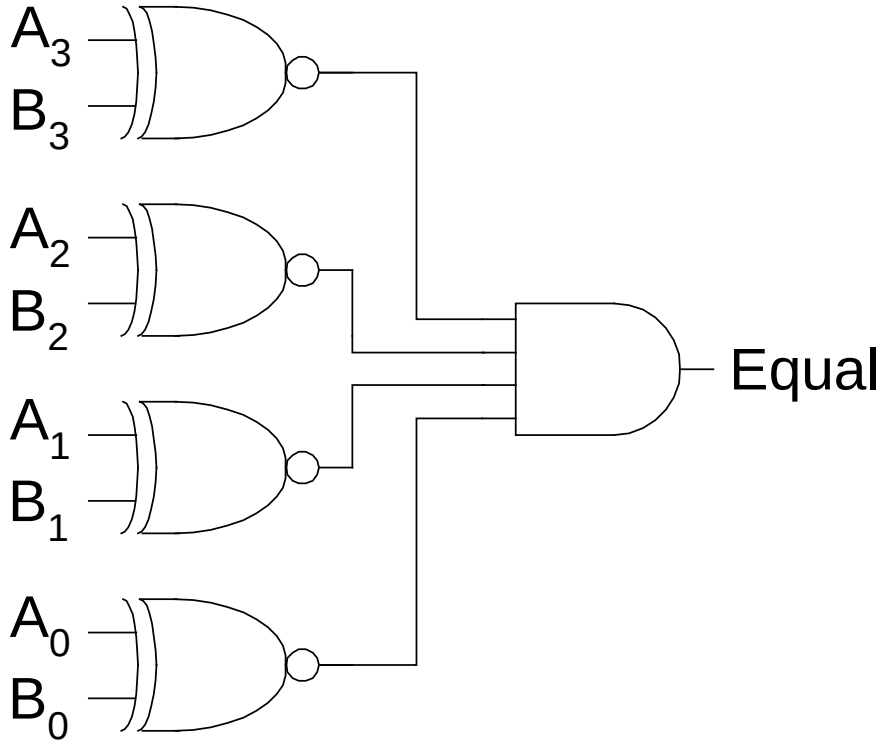


Компаратор. Порівняння на рівність

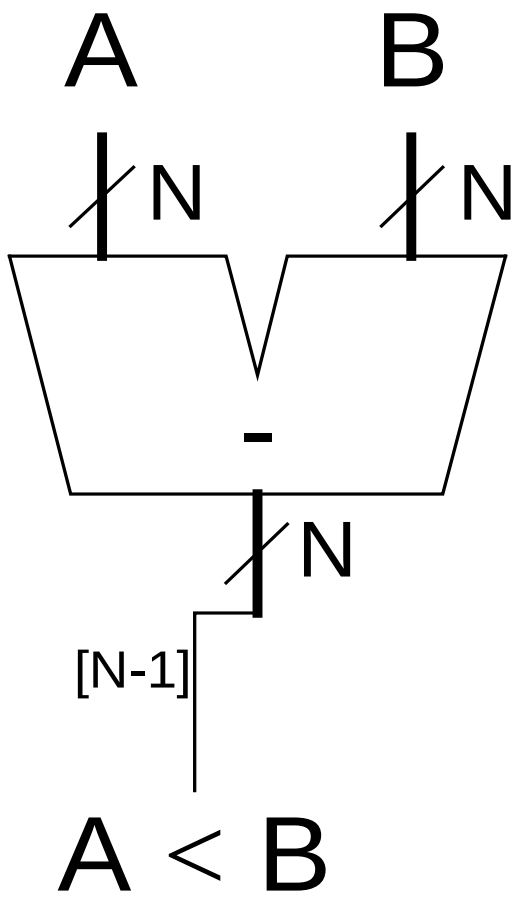
Symbol



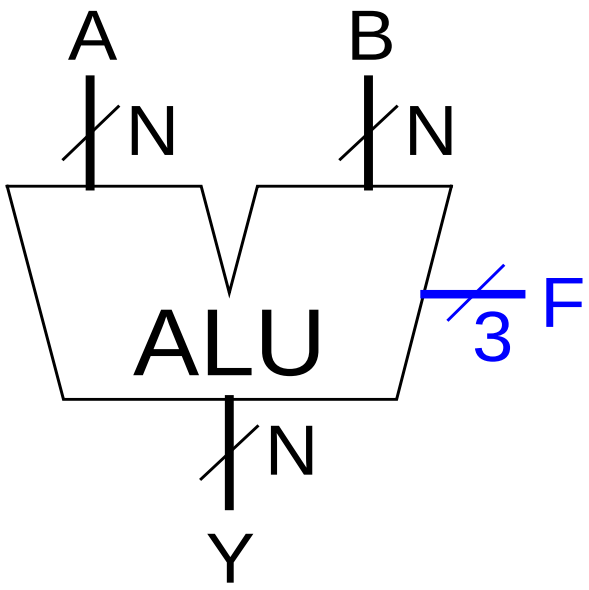
Implementation



Компаратор. Менше ніж.

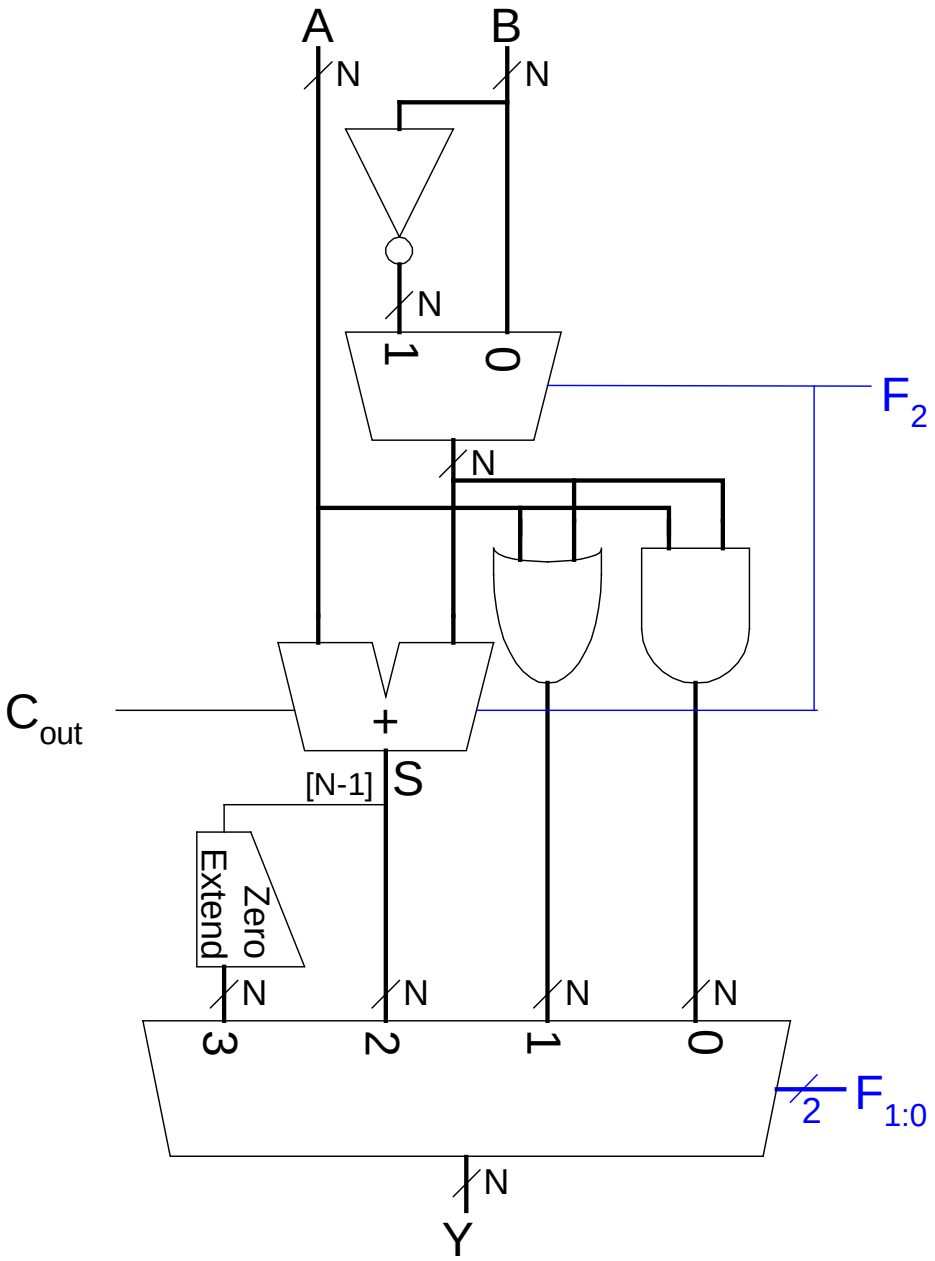


Арифметико-логічний пристрій



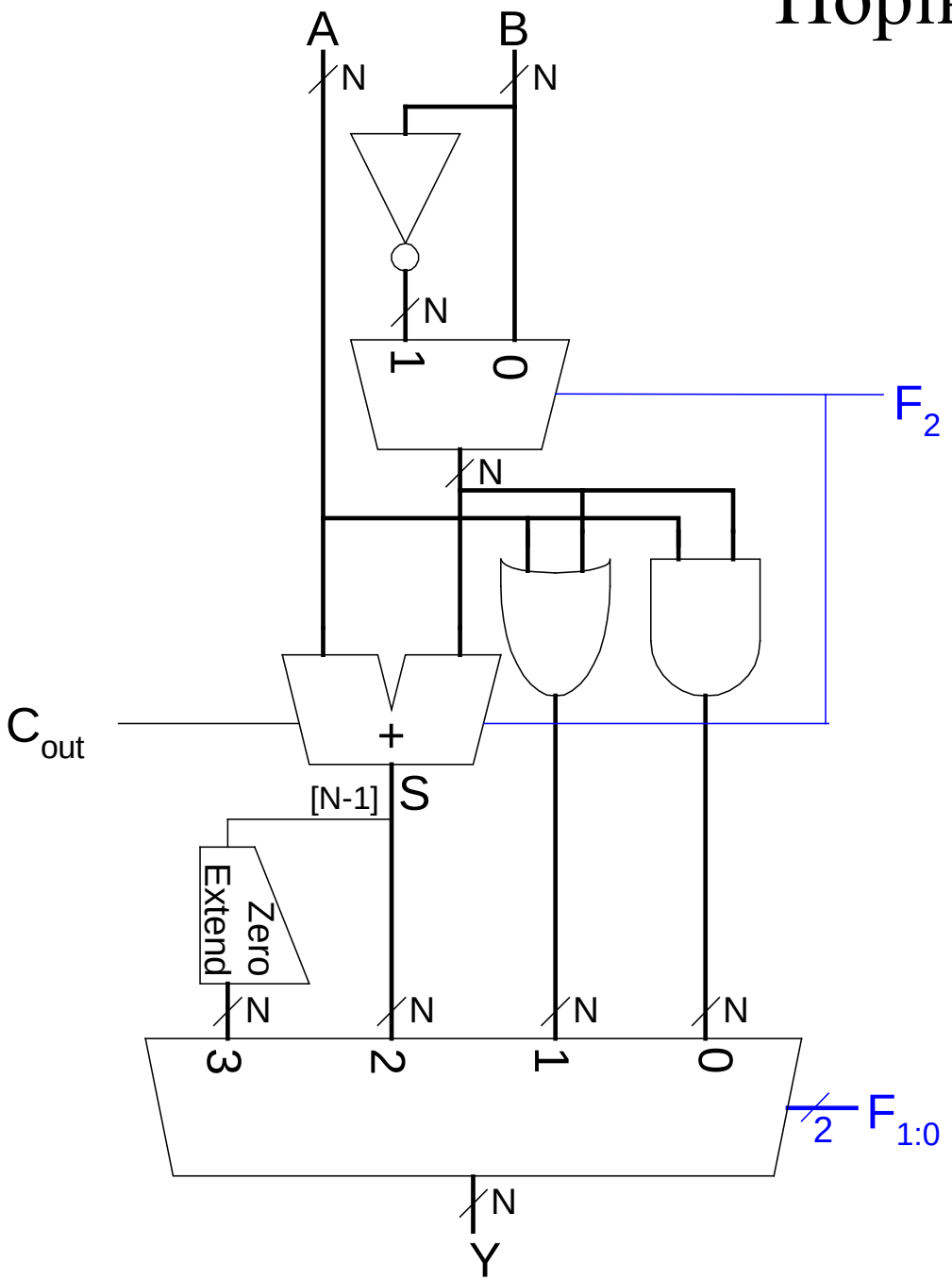
$F_{2:0}$	**Функція**
000	A & B
001	A | B
010	A + B
011	Не використовується
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Схема АЛП



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

Порівняння «менше ніж»



- Порівняння на «Менше ніж» (Set Less Than, SLT)
- Конфігурування 32-розрядного АЛУ для операції SLT: $A = 25$ и $B = 32$

- A < B, тому Y має бути 32-розрядним поданням 1 (0x00000001)
- F2:0 = 111
- F2 = 1 (суматор працює як віднімач): 25 - 32 = -7
- -7 має 1 в старшому розряді (S31 = 1)
- F1:0 = 11 мультиплексор вибирає Y = S31 (доповнення нулями) = 0x00000001.

Пристрій зсуву

- **Логічний зсув:** зміщує зсовуване значення вліво або вправо і заповнює порожні розряди нулями «0»

  Приклад: 11001 >> 2 = 00110

  Приклад: 11001 << 2 = 00100

- **Арифметичний зсув:** при зсуві вліво працює так як і логічнй зсув, а при зсуві вправо заповнює порожні розряди значеннями старшого біту (most significant bit, msb).

  Приклад: 11001 >>> 2 = **11**110

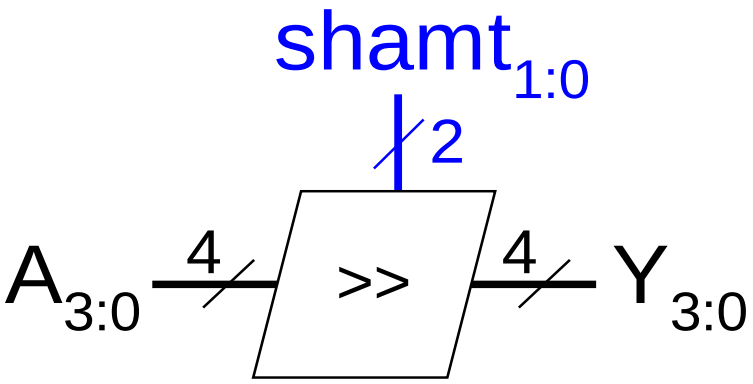
  Приклад: 11001 <<< 2 = 00100

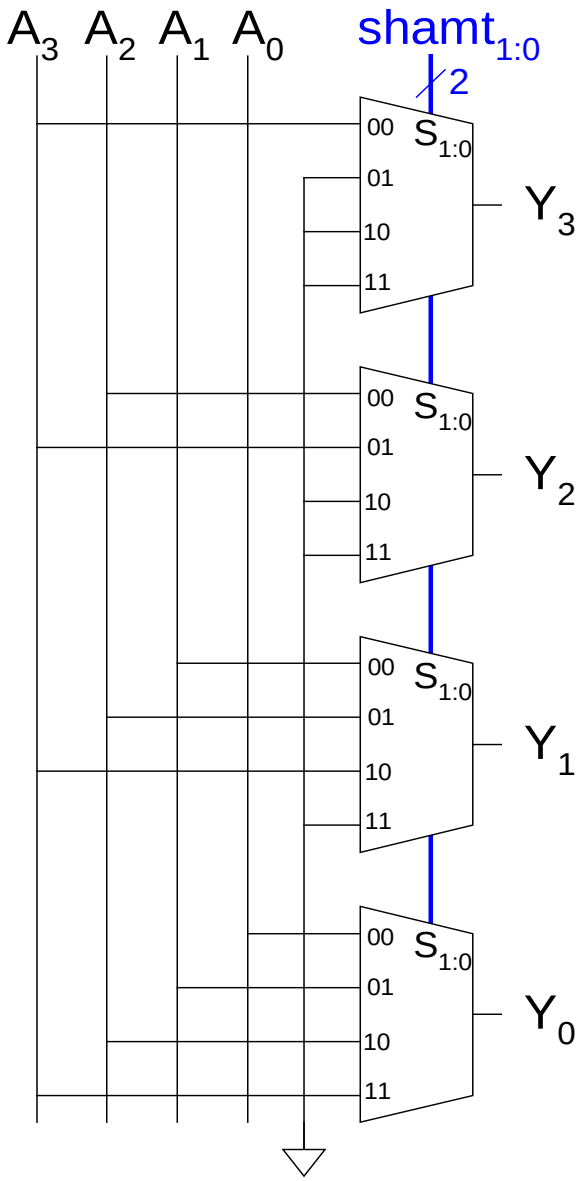
- **Циклічний зсув:** зсовує біти по колу, так, що біт який вийшов за межі розрядної сітки заповнює місце звільненого розряду на іншому кінці розрядної сітки

  Приклад : 11001 ROR 2 = 01110

  Приклад : 11001 ROL 2 = 00111

Схема пристрою зсуву





Пристрій зсуву як помножувач і поділювач

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

Пристрій множення

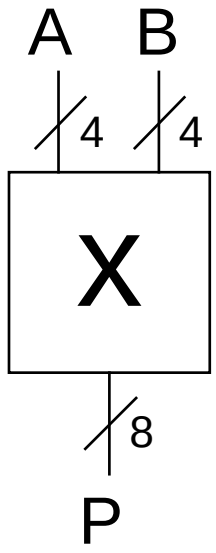
- **Часткові добутки,** формуються шляхом множення поточного розряду множника на всі розряди множеного
- **Зсунуті часткові добутки,** додаються для формування результату

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

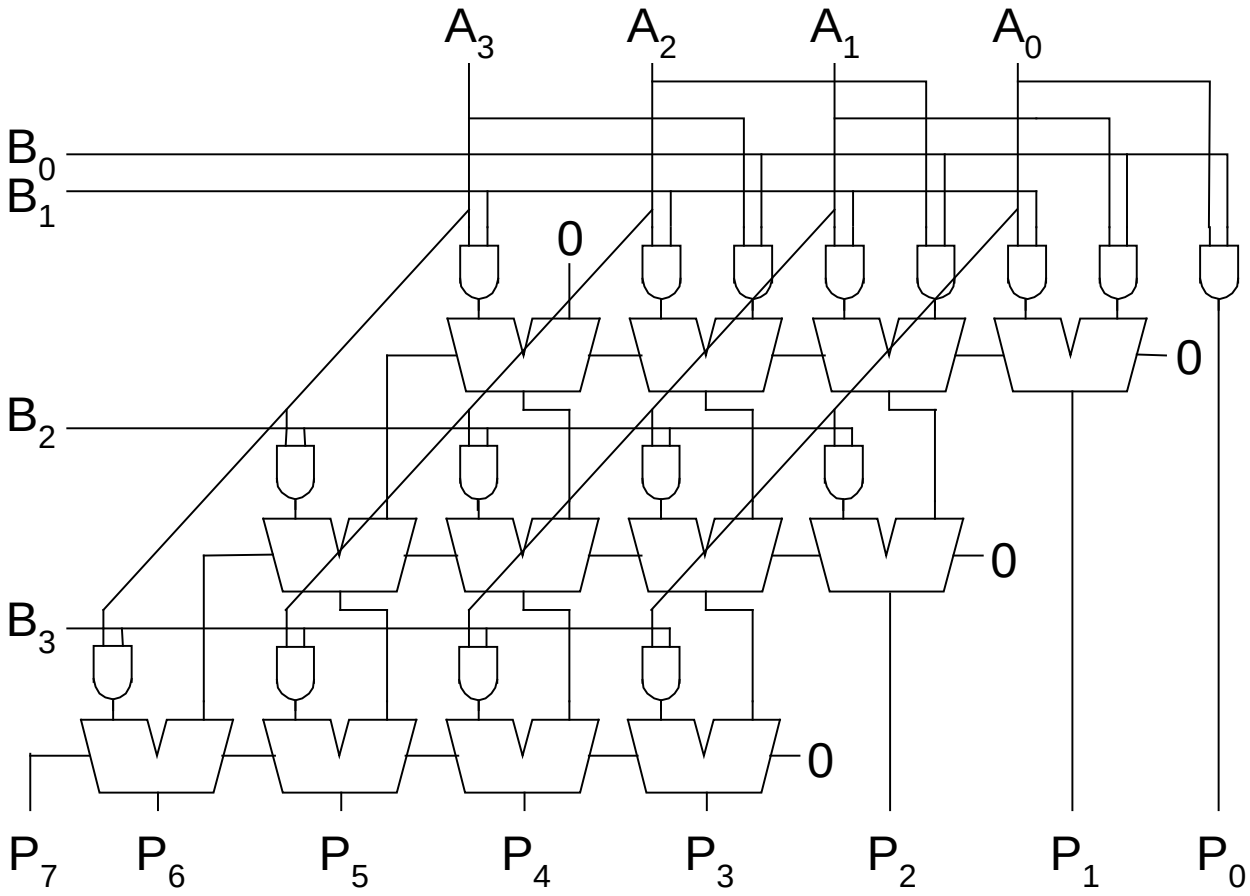
230 x 42 = 9660

5 x 7 = 35

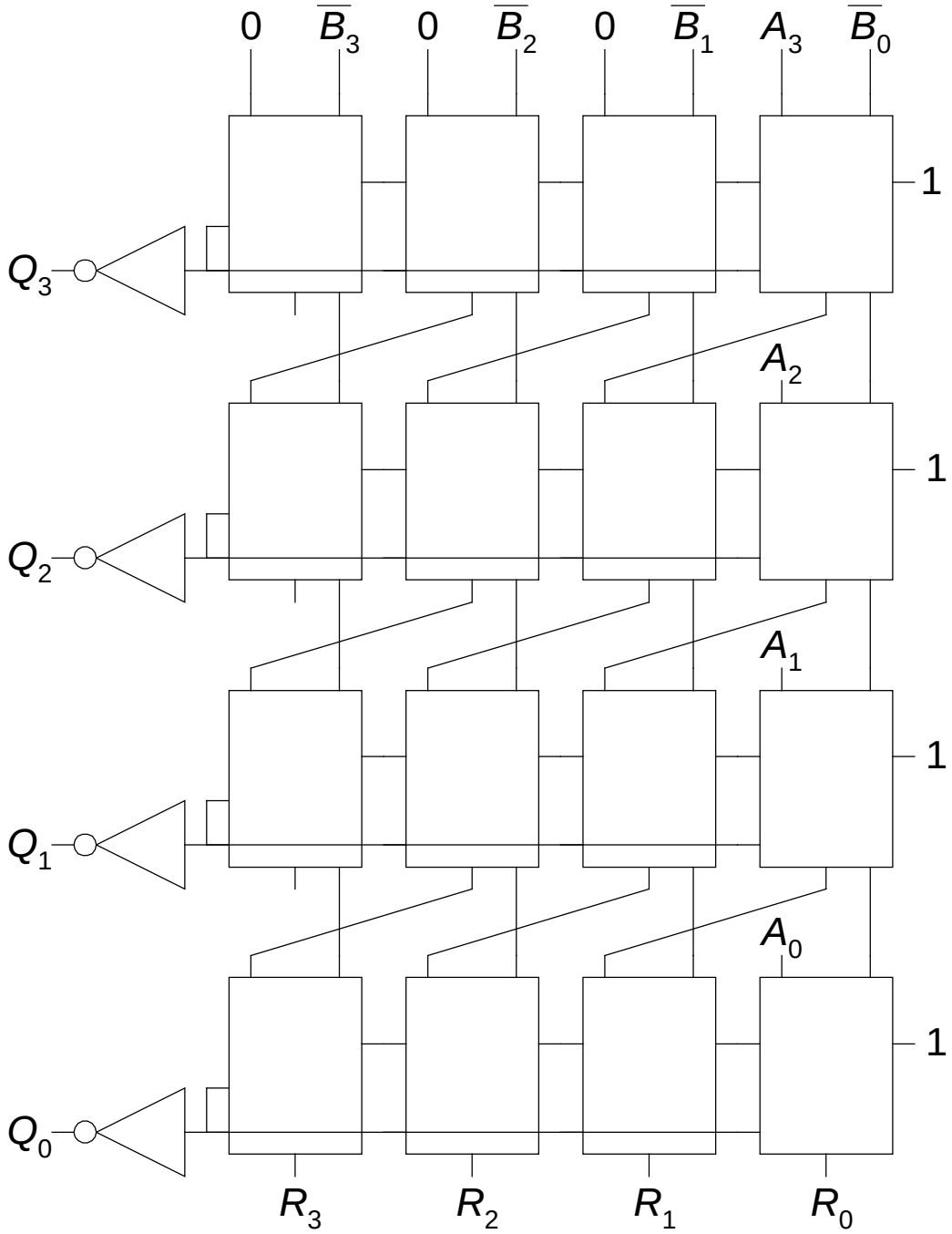
Помножувач 4 × 4

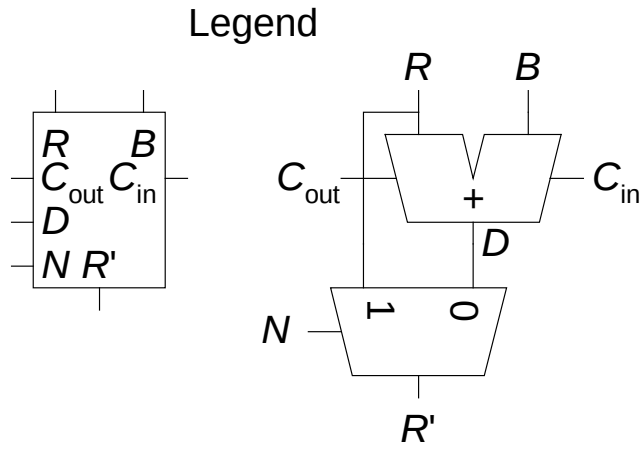


$$
\begin{array}{cccccccc}
 & & & A_3 & A_2 & A_1 & A_0 & \\
 & x & & B_3 & B_2 & B_1 & B_0 & \\
\hline
 & & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & \\
 & & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & & \\
 & A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & & \\
+ & A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & \\
\hline
P_7 &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



Подільювач 4x4





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

Системи числення

- Числа задаються у двійковій системі
- Ціле число:
  - **Додатне число**
    - прямий код 3 => 0011
  - **Від’ємне число**
    - прямий код -3 => 1011
    - інверсний код -3 => 1100
    - доповняльний код -3 => 1101
- Дробові числа:
  - **Fixed-point:** двійкове число з фіксованою крапкою
  - **Floating-point:** двійкове число з плаваючою крапкою

Числа з фіксованою крапкою

Число, яке містить задану наперед фіксовану кількість розрядів до і після крапки. Місце крапки уявне. Окремий розряд під знак. Перетворення числа -45.2 у двійкове із знаком.

```
45:2=22, залишок 1          0.2*2
22:2=11, залишок  0       0|0.4*2
11:2=5,  залишок 1        0|0.8*2
 5:2=2,  залишок 1        1|1.6*2
 2:2=1,  залишок 0        1|1.2*2
 1:2=0,  залишок 1        0|0.4
45=> 0010_1101            0.2 => .0011
```

або

```
45:16=2, залишок 13=>D
45 => 2D => 0010_1101
```

0010_1101 => прямий код
1010_1101 => знак
1101_0010 => інверсний код
\+ 1
1101_0011 => доповняльний код

Числа з плаваючою крапкою

- Двійкова крапка «плаває» між значущими цифрами
- Подібно до десяткового подання в експонентній формі
- Наприклад, 273_{10} в експонентній формі:

$$\mathbf{273 = 2.73 \times 10^2}$$

- У загальному вигляді, число записується в експонентній формі як:

$$\mathbf{\pm M \times B^E}$$

  - **M** = нормалізована мантиса (значення від 1 до 9)
  - **B** = основа показникової функції
  - **E** = порядок (експонента)
  - Например, M = 2.73, B = 10, and E = 2

Подання числа з плаваючою крапкою. 1

Задати число 228_{10} у 32-бітному поданні з плаваючою крапкою

1. Перетворення десяткового числа в двійкове (**не міняти порядком кроки 1 і 2**):

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

2. Записати число в двійковій системі і експонентній формі:

   $$\mathbf{11100100_2 = 1.11001_2 \times 2^7}$$

3. Заповнити кожне поле 32-бітного числа з плаваючою крапкою:
   - знак – позитивний, знаковий біт – (0)
   - 8 розрядів порядку задає значення 7
   - решта 23 розряди – мантиса

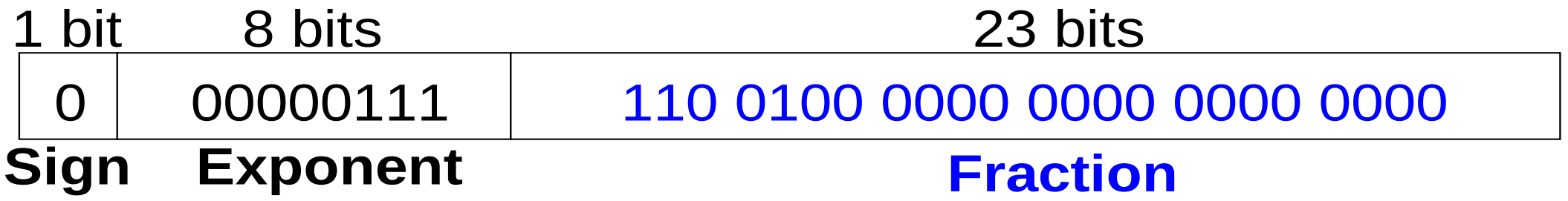
1 bit	8 bits	23 bits
0	00000111	11 1001 0000 0000 0000 0000
Sign	**Exponent**	**Mantissa**

Подання числа з плаваючою крапкою. 2

- Перший розряд мантиси завжди 1:

  $228_{10} = 11100100_2 = \mathbf{1}.11001 \times 2^7$

- Тому, не має необхідності йог збегігати: *неявна ведуча 1*
- Зберігаються тільки дробові розряди мантиси в 23-розрядному полі

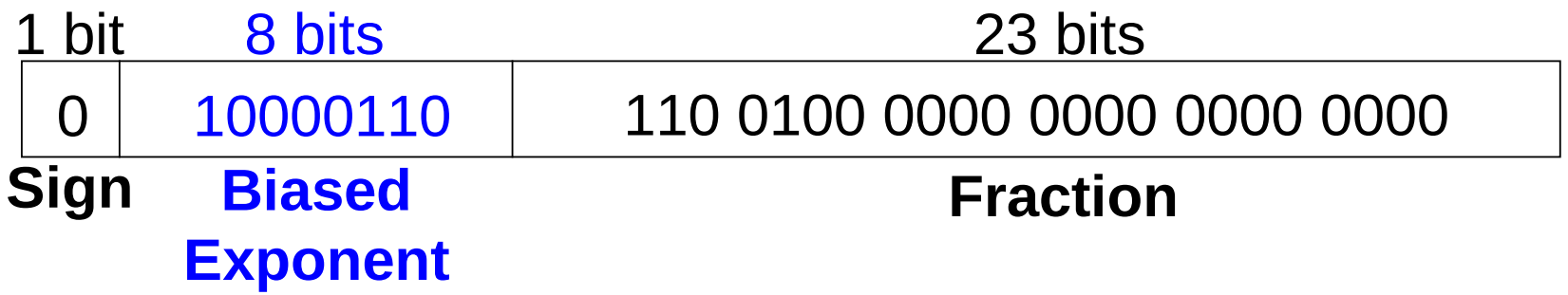


Подання числа з плаваючою крапкою. 3

- *Зміщений порядок*: зміщення = 127 (01111111_2)
  - Зміщений порядок = зміщення + порядок
  - Порядок 7 зберігається як:

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

- **Двійкове 32-розрядне (IEEE 754) подання з плаваючою крапкою** числа 228_{10}



- 16-вий код: **0x43640000**

Подання числа з плаваючою крапкою. Приклад

Записати число -58.25_{10} з плаваючою крапкою (IEEE 754)

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

1. Записати в двійковій системі і експонентній формі:

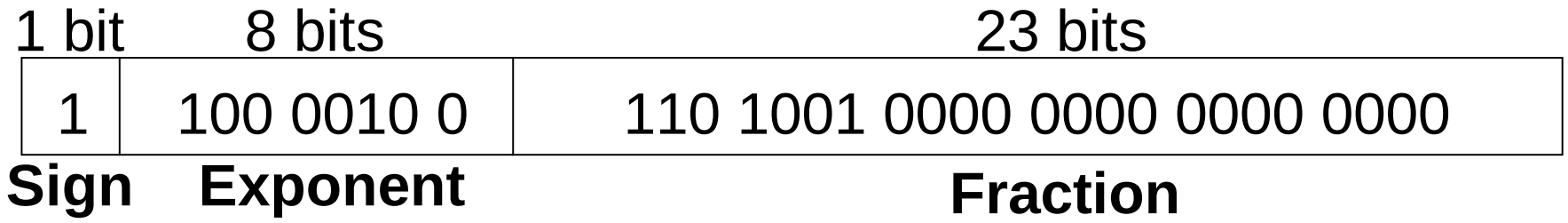
   $\mathbf{1.1101001 \times 2^5}$

3. Заповнити поля:

   Знаковий разряд: 1 (негативний)

   8 розрядів порядку: (127 + 5) = 132 = $\mathbf{10000100_2}$

   23 разряди мантиси: 110 1001 0000 0000 0000 0000



- 16-вий код: **0xC2690000**

Плаваюча крапка. Особливі випадки

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

Плаваюча крапка. Точність.

- **Одинарна точність:**
  - 32-разряди
  - 1 знаковий розряд, 8 розрядів порядку, 23 розряди мантиси
  - Зміщення = 127

- **Подвійна точність:**
  - 64- разряди
  - 1 знаковый разряд, 11 розрядів порядку, 52 розряди мантиси
  - Зміщення = 1023

Плаваюча крапка. Округлення

- **Переповнення:** число занадто велике для подання
- **Втрата точності:** число занадто мале для подання
- **Режими округлення:**
  - Вниз – до меншого
  - Вверх – до більшого
  - До нуля (до меншого по модулю)
  - До найближчого
- **Приклад:** округлити 1.100101 (1.578125) до числа з 3 дробовими розрядами
  - Вниз: 1.100
  - Вверх: 1.101
  - До нуля: 1.100
  - До найближчого: 1.101 (1.625 ближче до 1.578125, ніж 1.5)

Плаваюча крапка. Додавання.

1. Виділити порядок числа і мантису
2. Приєднати ведучу 1 до мантиси
3. Порівняти порядки
4. Виконати зсув меншої мантиси при необхідності
5. Додати мантиси
6. Нормалізувати мантиси і підібрати порядки при необхідності
7. Округлити результат
8. Зібрати порядок і мантису назад у формат з плаваючою крапкою

Додавання чисел з плаваючою крапкою. Приклад

Додати два числа с плаваючою крапкою: 0x3FC00000 + 0x40500000

1. Видобути порядок і мантису

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

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

2. Приєднати ведучу 1 до мантиси

N1: 1.1

N2: 1.101

Додавання чисел з плаваючою крапкою. Приклад

3. **Порівняти порядки**

   127 – 128 = -1, тому зсув N1 вправо на 1 розряд

4. **Зсув меншої мантиси при необхідності**

   зсув мантиси N1: 1.1 >> 1 = 0.11 $(\times 2^1)$

5. **Додати мантиси**

$$\begin{aligned} 0.11 \quad & \times 2^1 \\ + \; 1.101 & \times 2^1 \\ 10.011 & \times 2^1 \end{aligned}$$

Додавання чисел з плаваючою крапкою. Приклад

6. **Нормалізувати мантиси і підібрати порядки при необхідності**

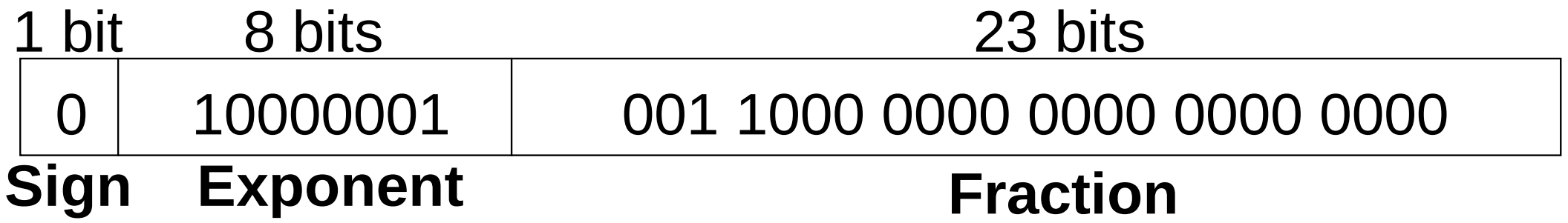
   $10.011 \times 2^1 = 1.0011 \times 2^2$

7. **Округлити результат**

   не потрібно (вклалися у 23 розряди)

8. **Зібрати порядок і мантису назад у формат з плаваючою крапкою**

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

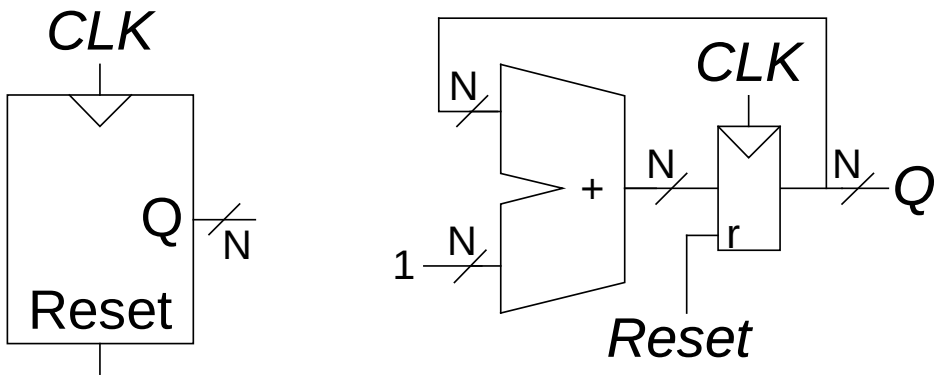


16-вий код: **0x40980000**

Лічильники

- Інкремент за кожним переднім фронтом
- Використовується у циклі для перебору всіх чисел. Наприклад,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- В прикладі:
  - відображення цифрового годинника
  - програмний лічильник: відслідковує виконання поточної команди

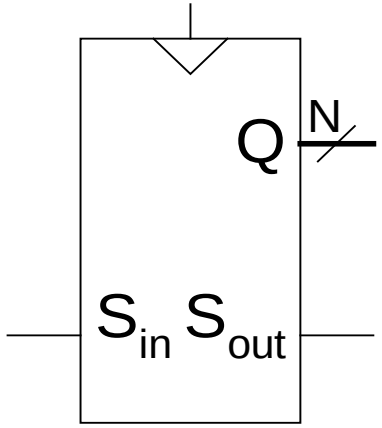
Symbol **Implementation**



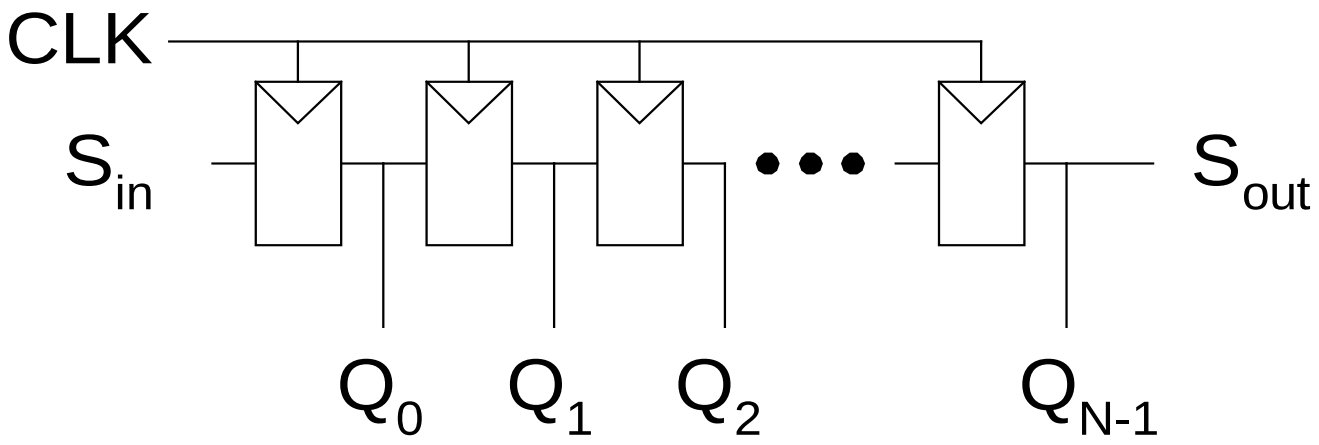
Зсувний регістр

- Всовується новий біт за переднім фронтом тактового сигналу
- Висовується біт за переднім фронтом тактового сигнала
- *Послідовно-паралельний перетворювач*: перетворює послідовний вхід (S_{in}) в паралельний вихід ($Q_{0:N-1}$)

Позначення

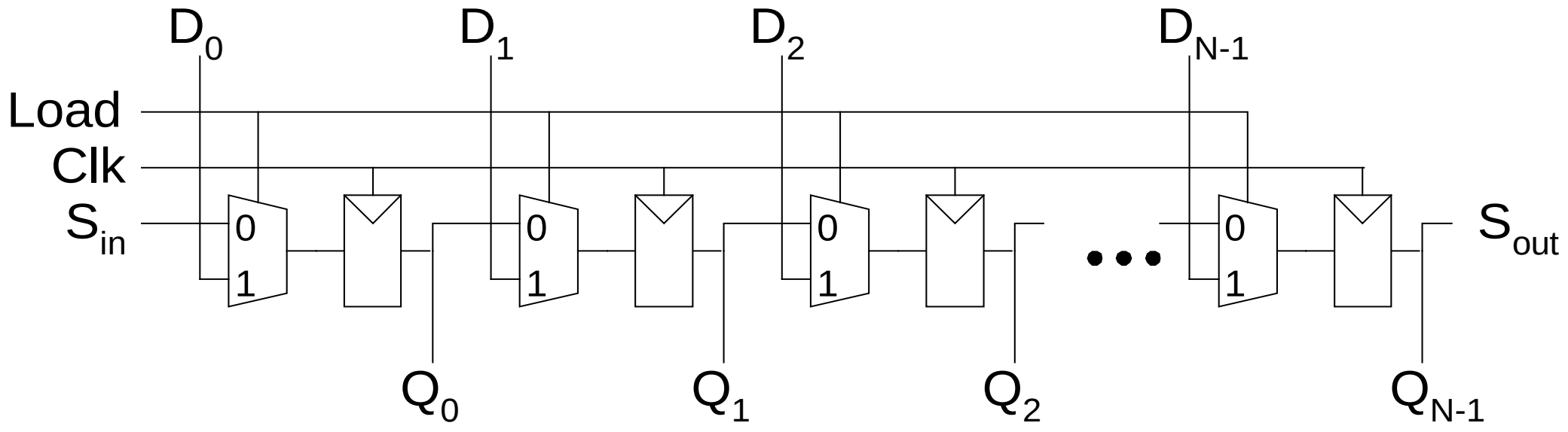


Реалізація



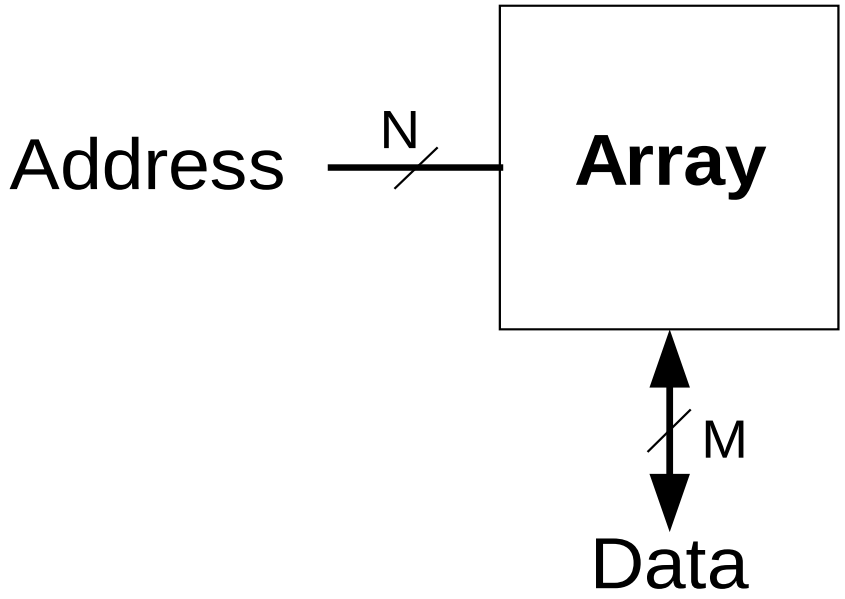
Зсувний регістр з паралельним завантаженням

- Коли *Load* = 1, працює як звичайний *N*-розрядний регістр
- Коли *Load* = 0, працює як регістр зсуву
- Може працювати як послідовно-паралельний перетворювач ($S_{in} \rightarrow Q_{0:N-1}$) або *паралельно-послідовний перетворювач* ($D_{0:N-1} \rightarrow S_{out}$)



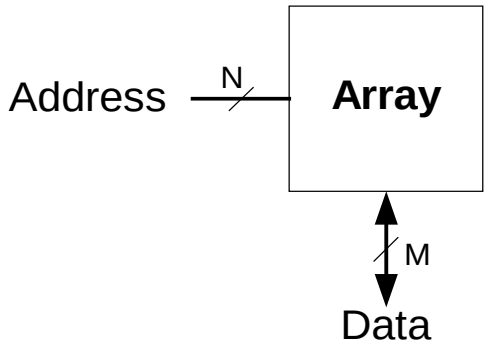
Матриці пам’яті

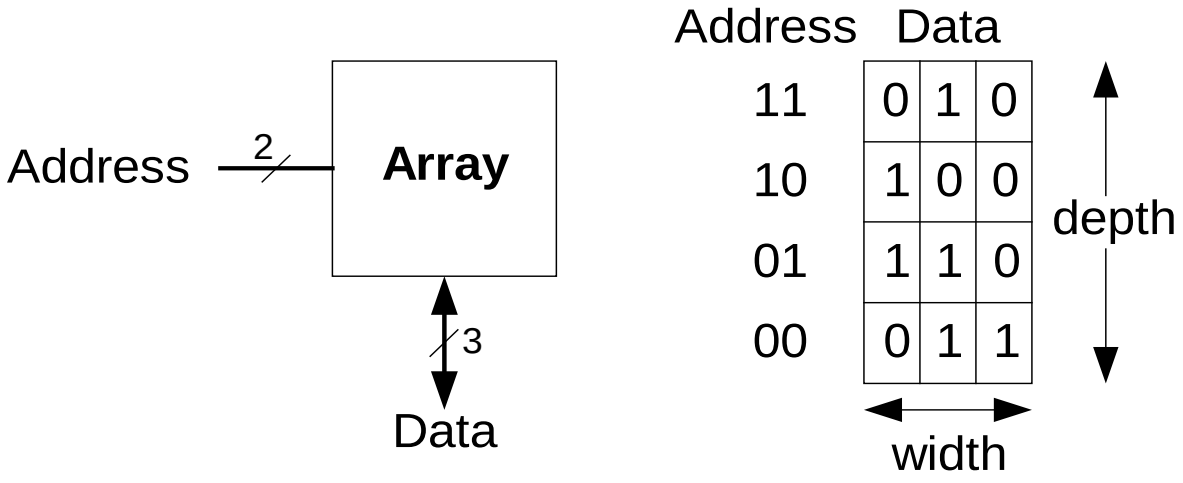
- Ефективно зберігають великі обсяги даних
- 3 основних типи:
  - динамічний оперативний запам’ятовуючий пристрій (ОЗП) (DRAM)
  - статичний оперативний запам’ятовуючий пристрій (ОЗУ) (SRAM)
  - постійний запам’ятовуючий пристрій (ПЗП), пам’ять тільки для читання (ROM)
- *M*-розрядне значення даних зчитується/записується за унікальною *N*-розрядною адресою



Матриці пам’яті

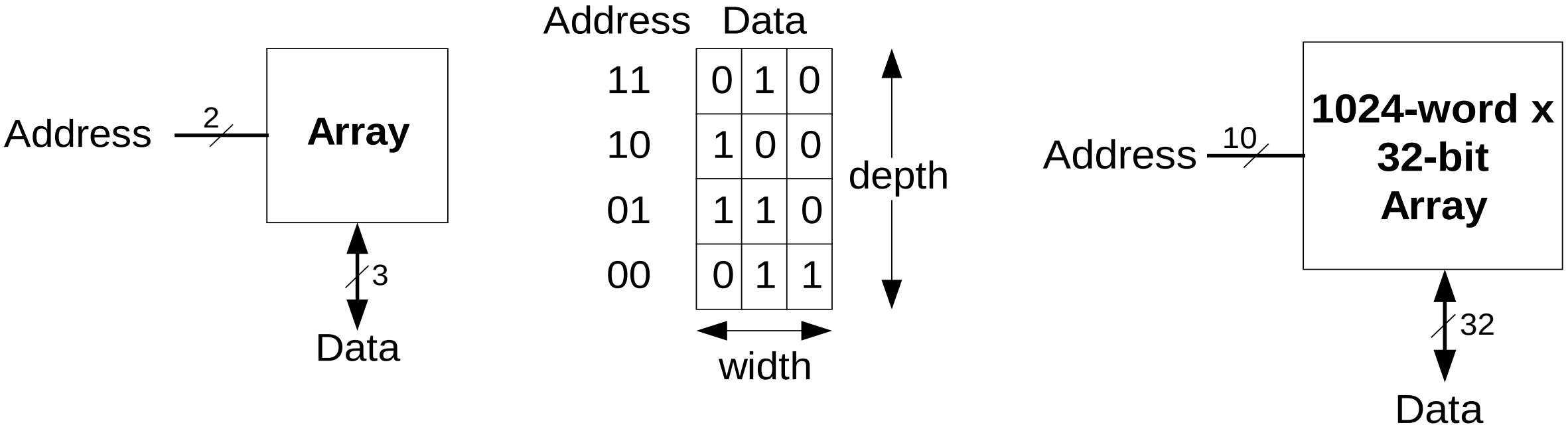
- 2-вимірна матриця бітових комірок
- Кожна бітова комірка зберігає 1 біт
- N адресних бітів і M бітов даних:
  - 2^N рядків і M стовпців
  - **Глибина (Depth):** кількість рядків (кількість слів)
  - **Ширина (Width):** число стовпців (розмір, довжина слова)
  - **Розмір матриці:** depth × width = $2^N \times M$



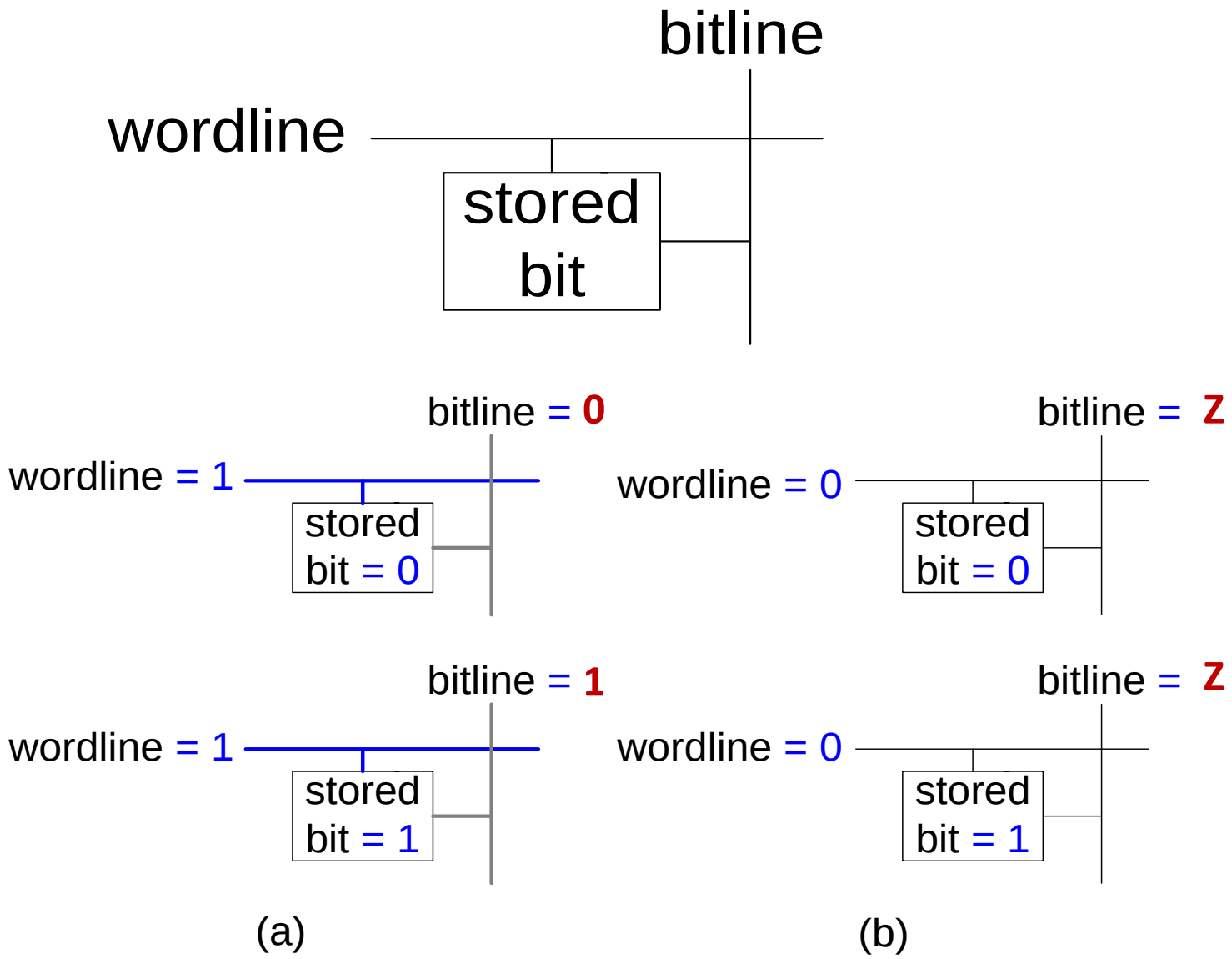


Приклад матриці пам’яті

- $2^2 \times 3$-бітова матриця
- Кількість слів: 4
- Довжина слова: 3 біти
- Наприклад, 3-бітове слово 100 зберігається за адресою 10

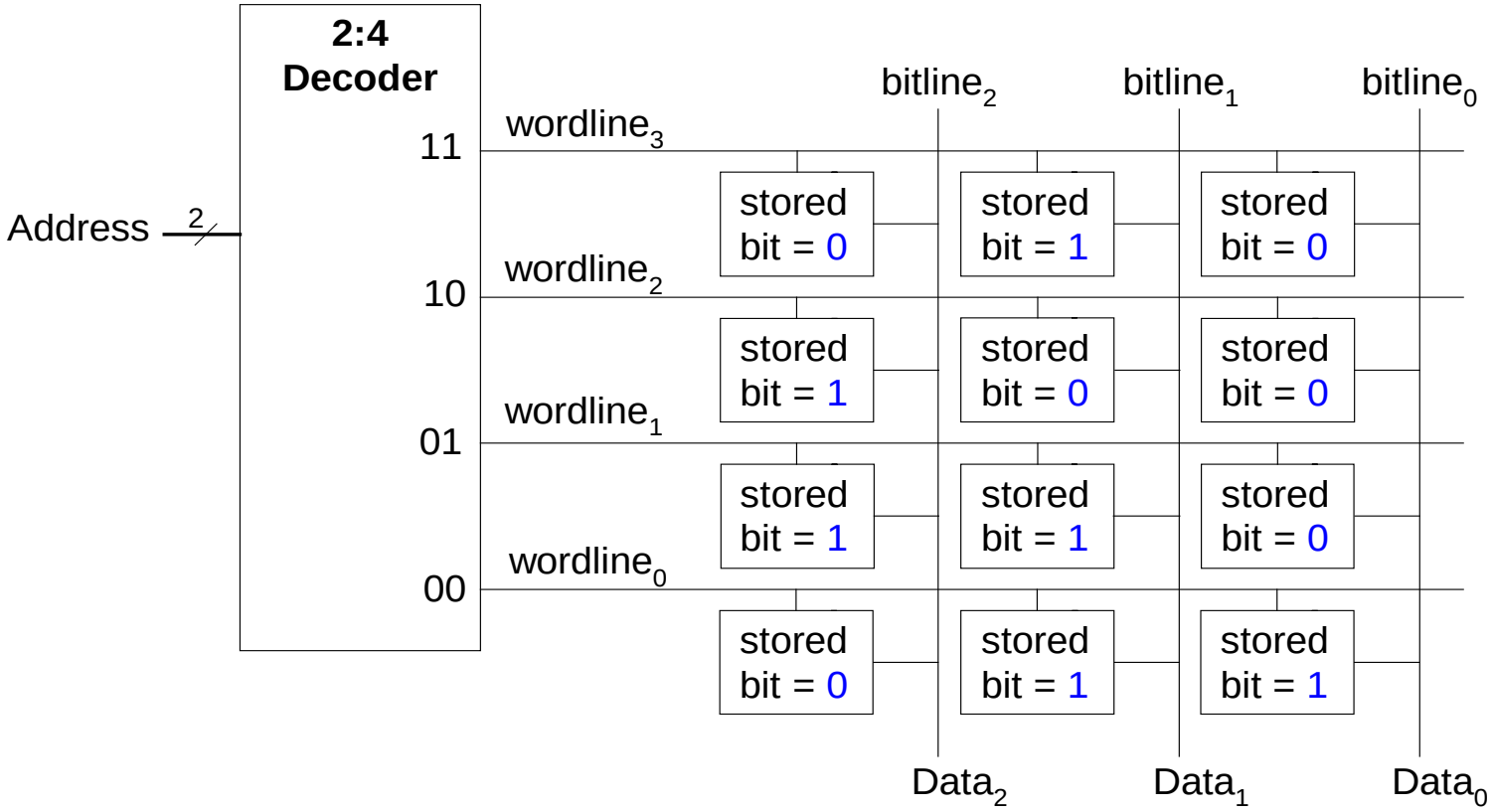


Запам’ятовуючі елементи матриці пам’яті



Матриця пам’яті

- **Линія вибірки слів (worldline):**
  - формує сигнал дозволу для вибору рядка
  - в матриці пам’яті тільки один рядок може читатися/записуватися
  - Відповідає унікальній адресі
  - тільки одна лінія вибірки слів може бути активною



Типи пам’яті

- з довільним доступом, оперативна пам’ять (RAM, ОЗП): енергозалежна (**volatile)**
- Пам’ять тільки для читання (ROM, ПЗП): енергонезалежна (**nonvolatile)**

RAM, ОЗП: Оперативна пам’ять

- **Енергозалежна:** вміст пам’яті втрачається при відключенні електроживлення
- Швидке читання і записування
- Основна пам’ять в комп’ютері – RAM (DRAM)

Історично склалася назва «пам’ять з довільним доступом», так як в ній доступ до любого слова даних для читання або записування виконується завжди за один і той же час (на відміну від пам’яті з послідовним доступом, такої як магнітна стрічка)

ROM, ПЗП. Пам’ять тільки для читання

- **Енергонезалежна:** вміст пам’яті зберігається при відключенні електроживлення
- Читання швидке, але запис неможливий або повільний
- Флеш пам’ят у відеокамерах, флеш-нагромаджувачах – ROM

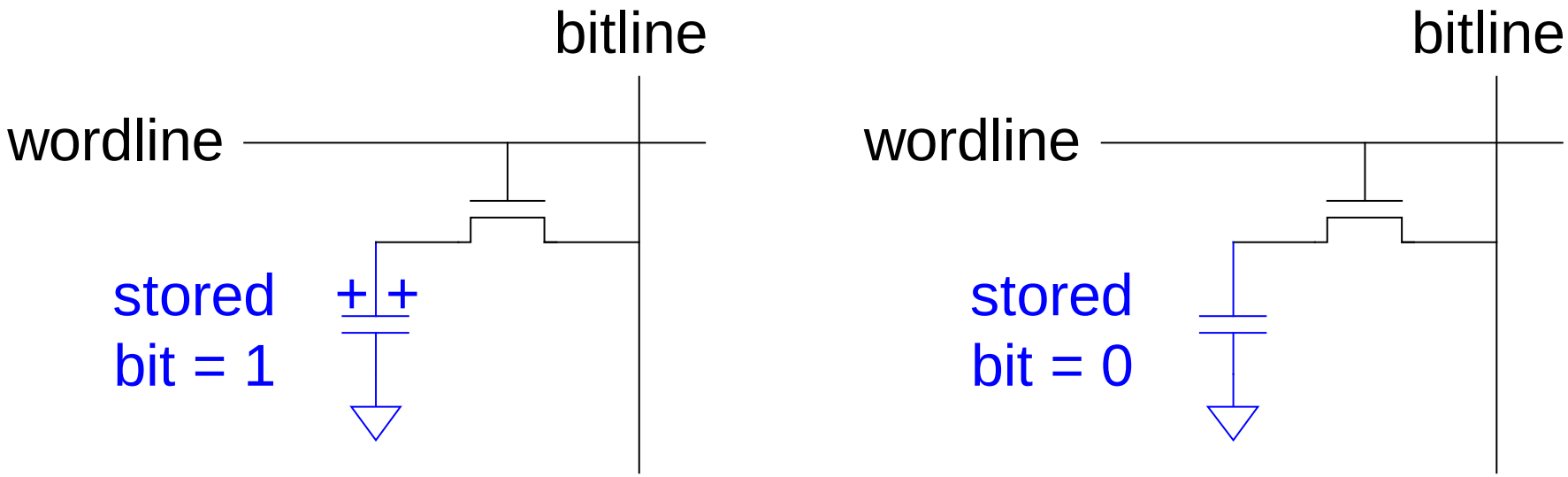
Історично склалася назва «пам’ять тільки для читання, ROM», оскільки інформація в неї могла бути записана тільки при її виготовленні або шляхом перепалу плавких перемичок. Після того, як пам’ять була сконфігурована, її не можна було записати знову. Тепер це не так.

Типи RAM

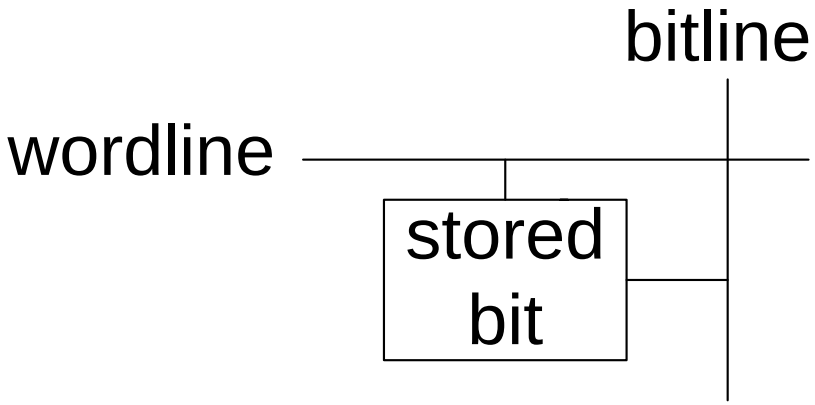
- **DRAM** (динамічний ОЗП - Dynamic random access memory)
- **SRAM** (статичний ОЗП - Static random access memory)
- Відрізняються способом зберігання даних:
  - DRAM використовує конденсатор
  - SRAM використовує інвертори з перехресними зворотніми зв’язками

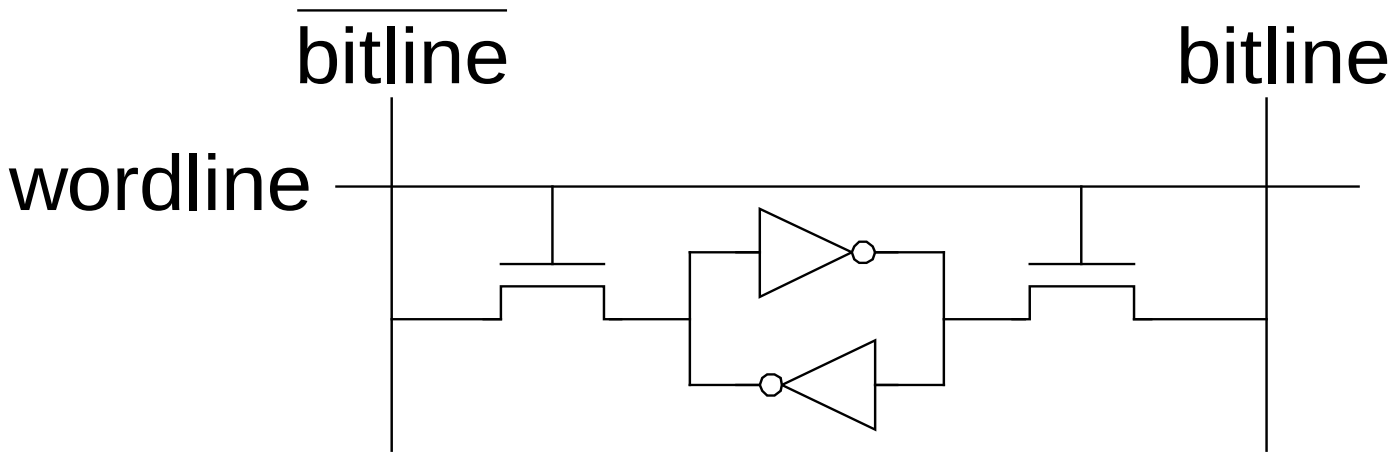
DRAM

- Біти даних зберігаються в конденсаторах
- *Динамічна*, тому що значення має бути оновлене (перезаписане) як періодично, так і після зчитування:
  - Витікання заряду конденсатора руйнує значення
  - Читання знищує збережене значення

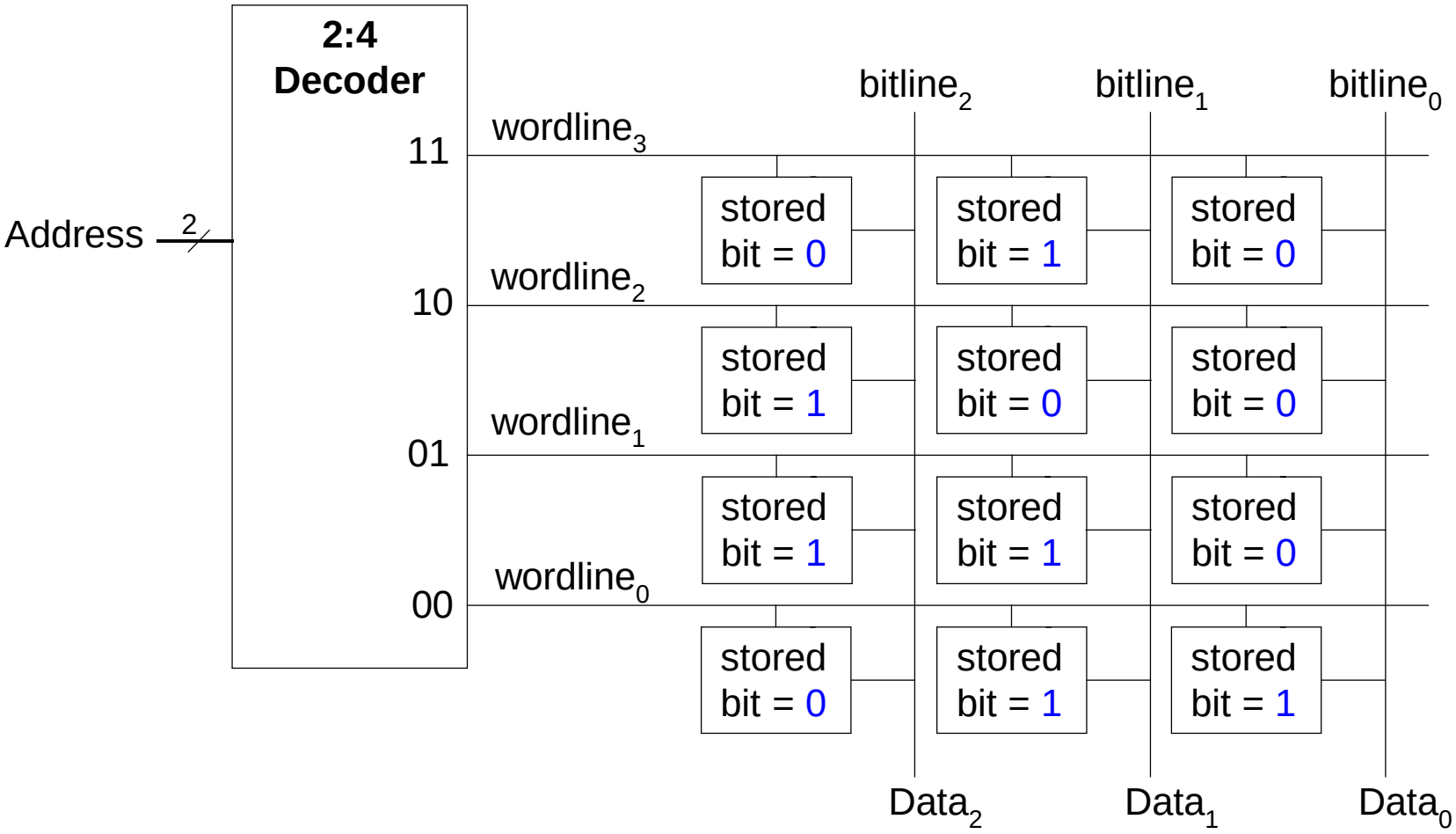


SRAM

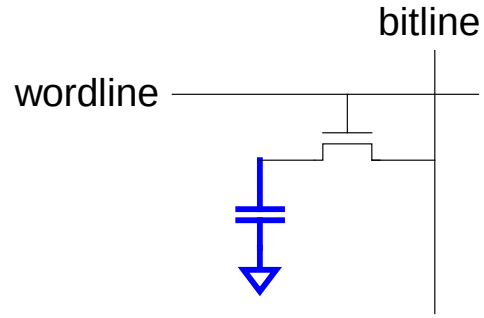




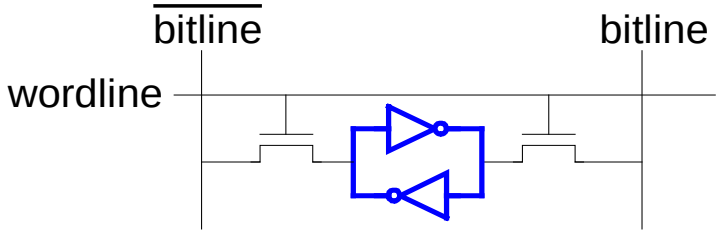
Матриця пам’яті. Огляд



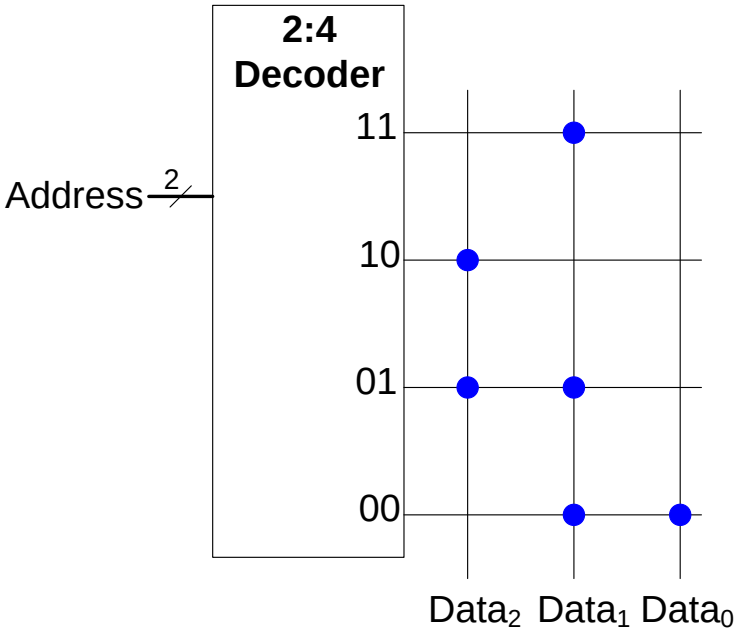
DRAM bit cell:

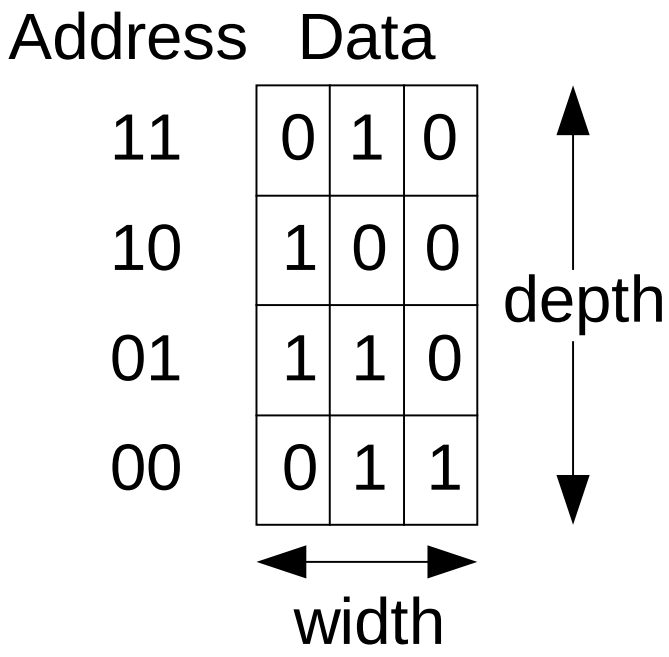


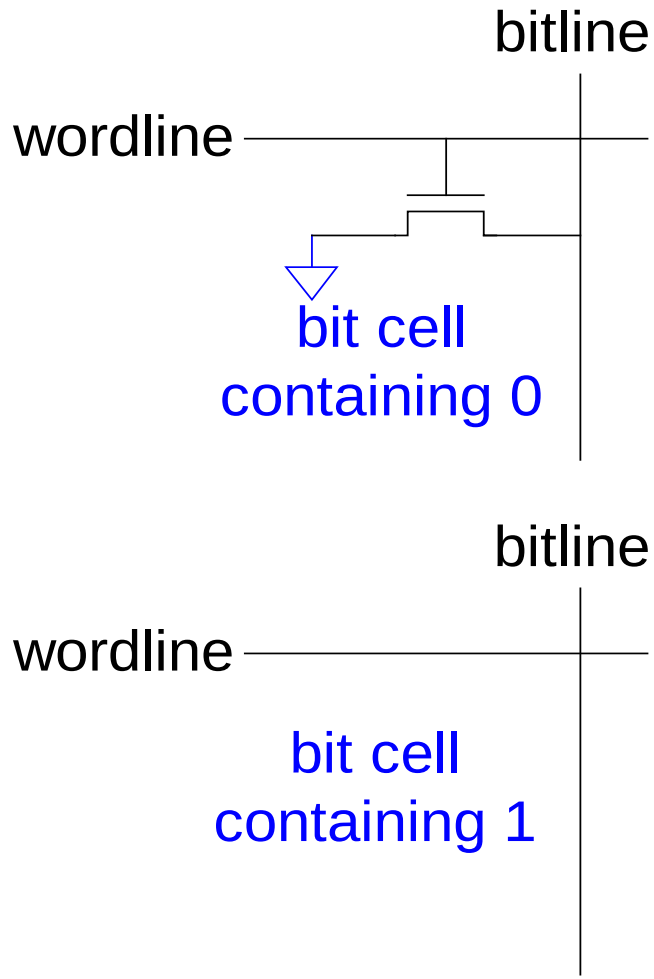
SRAM bit cell:



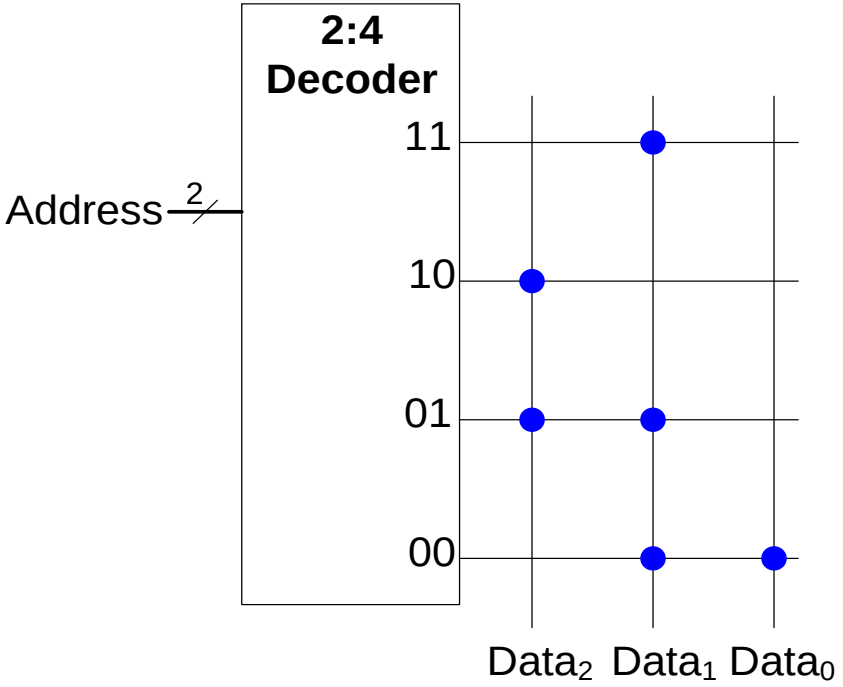
ПЗП (ROM): Крапкова нотація. Зберігання даних







Логічні функції і ПЗП



$$Data_2 = A_1 \oplus A_0$$

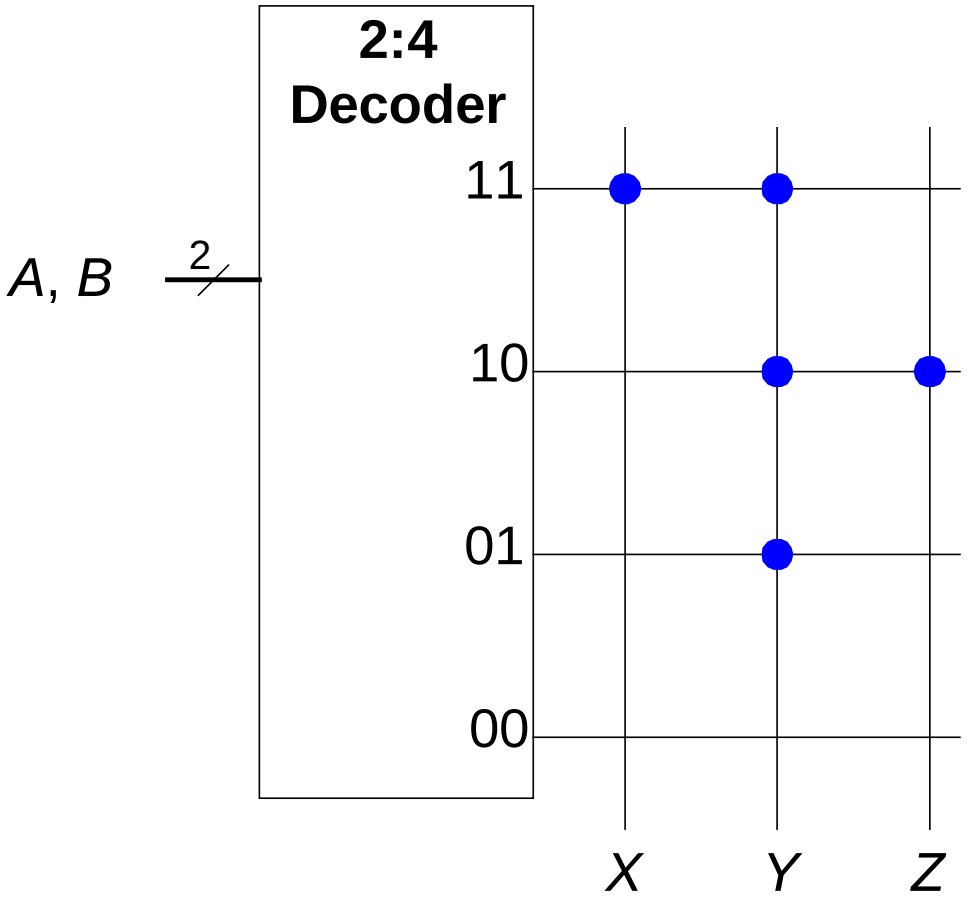
$$Data_1 = \overline{A_1} + A_0$$

$$Data_0 = \overline{A_1 A_0}$$

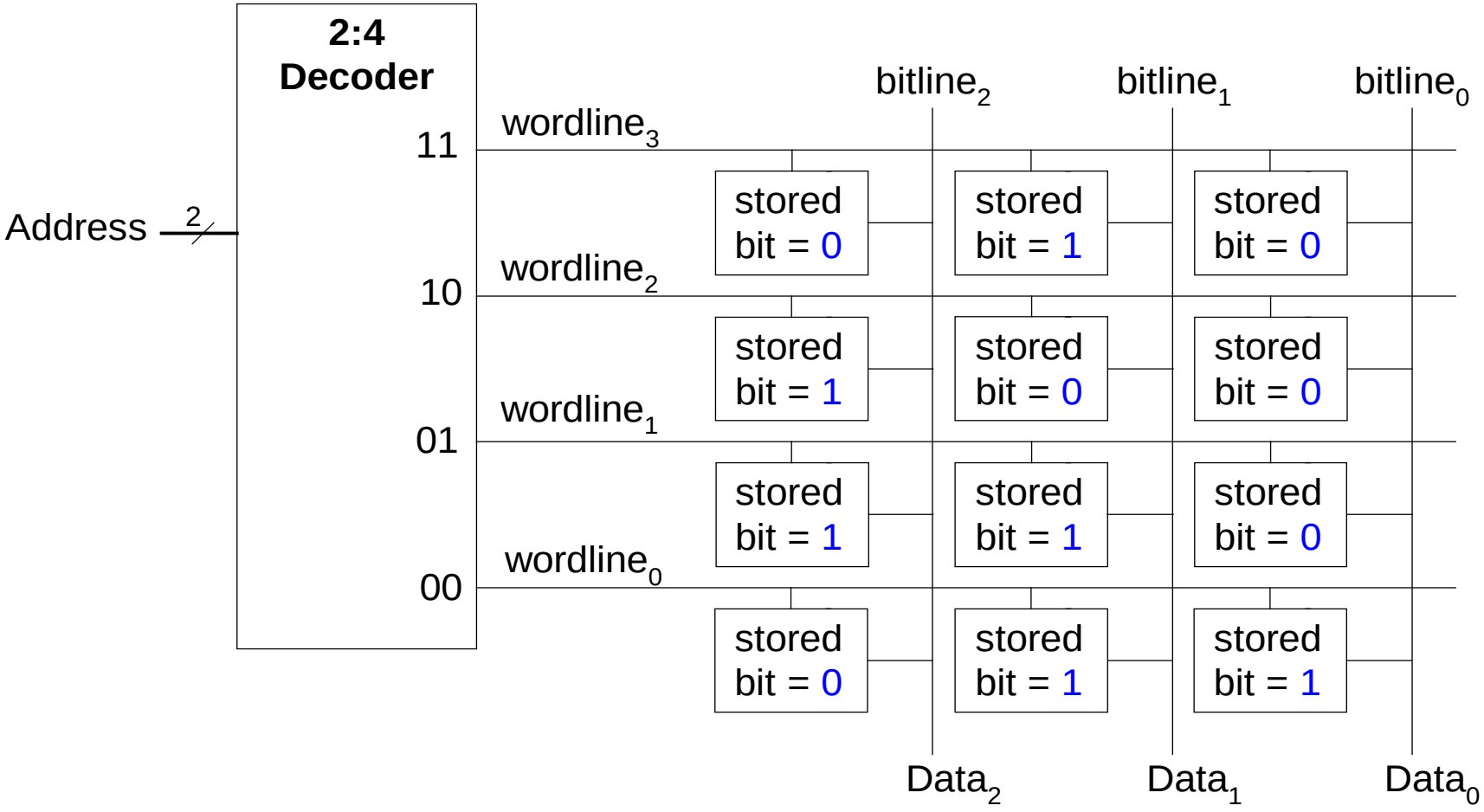
Логіка на основі ПЗП. Приклад

Реалізувати наступні логічні функції, використовуючи ПЗП $2^2 \times 3$-біти:

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



Логіка на основі любої матриці пам’яті



$$Data_2 = A_1 \oplus A_0$$

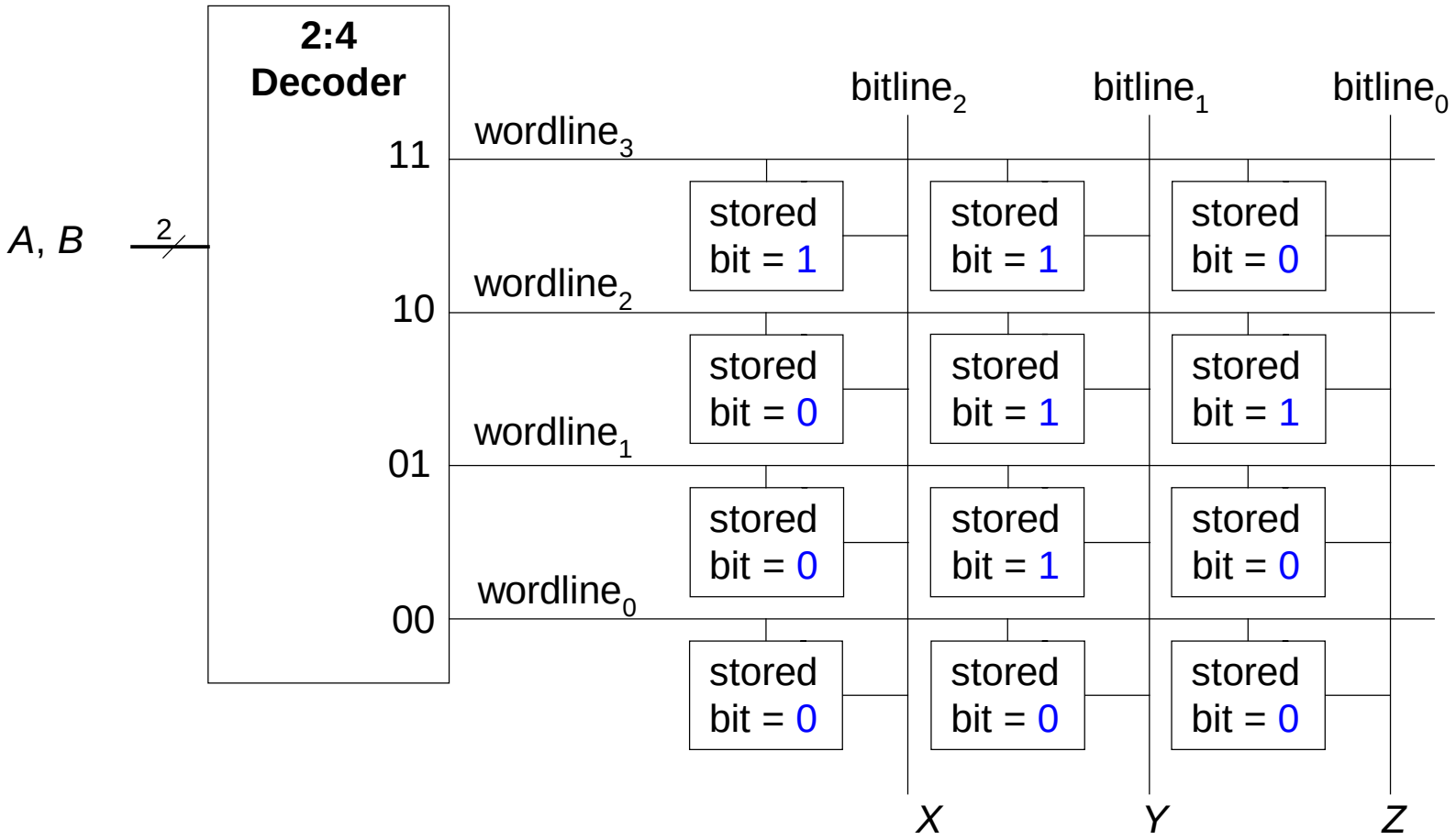
$$Data_1 = \overline{A_1} + A_0$$

$$Data_0 = \overline{A_1 A_0}$$

Логіка на основі матриці пам’яті

Реалізувати наступні логічні функції, використовуючи $2^2 \times 3$-бітову матрицю пам’яті:

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

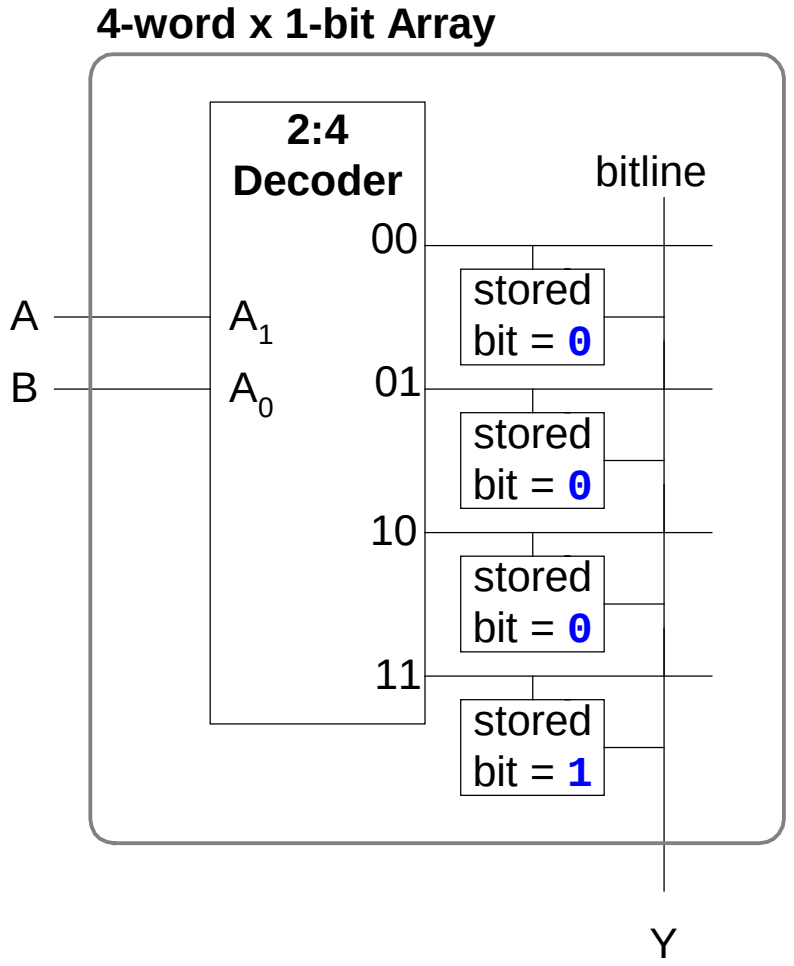


Логіка на основі матриці пам’яті. Таблиці перетворень

Таблиці перетворень (*lookup tables*, LUTs) - кожній вхідній комбінації відповідає деякий стан виходу

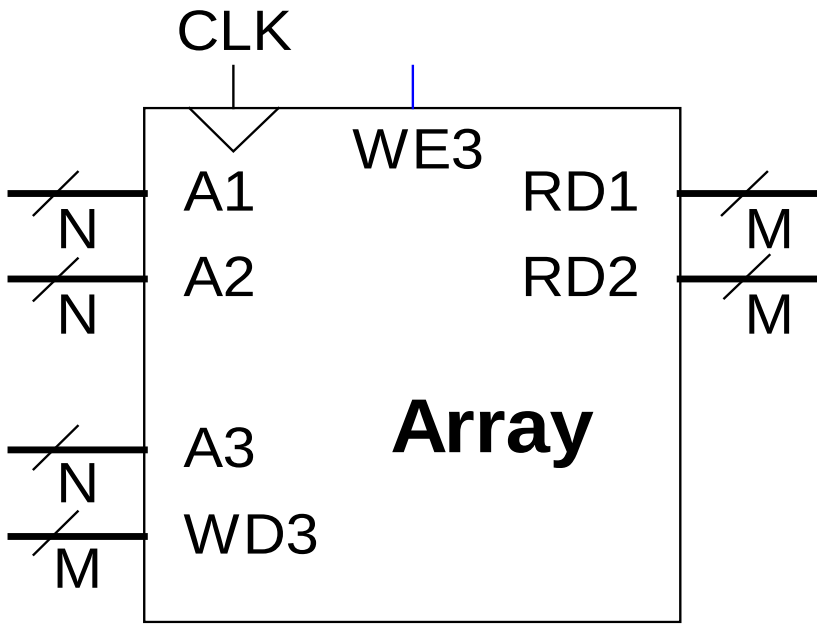
Truth Table

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



Багатопортова пам’ять

- **Порт:** пара адреса/дані
- 3-портова пам’ять
  - 2 порти читання (A1/RD1, A2/RD2)
  - 1 порт записування (A3/WD3, WE3 дозвіл записування)
- **Регістровий файл:** мала багатопортова пам’ять



SystemVerilog

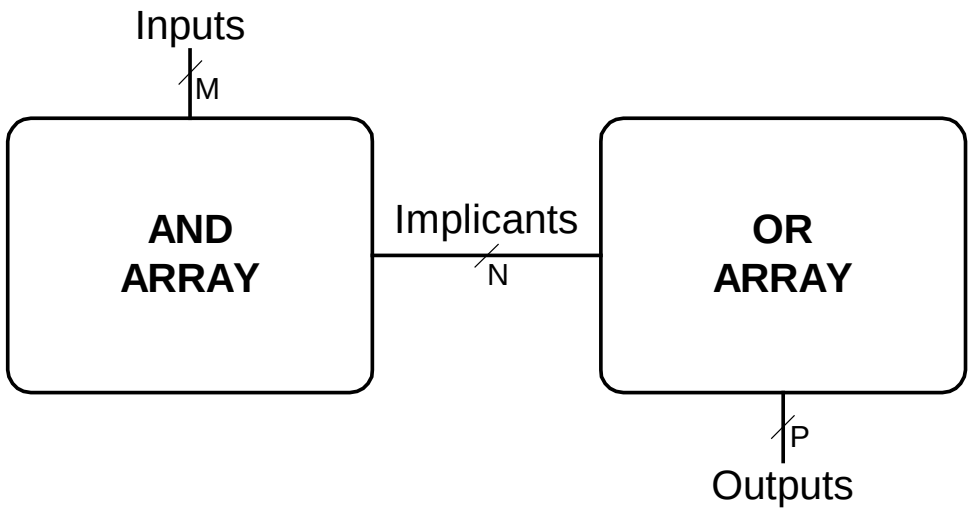
```
// 256 x 3 модуль пам'ят з одним портом читання/записування
module dmem(   input  logic       clk, we,
               input  logic[7:0]  a
               input  logic [2:0] wd,
               output logic [2:0] rd);

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

  assign rd = RAM[a];

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

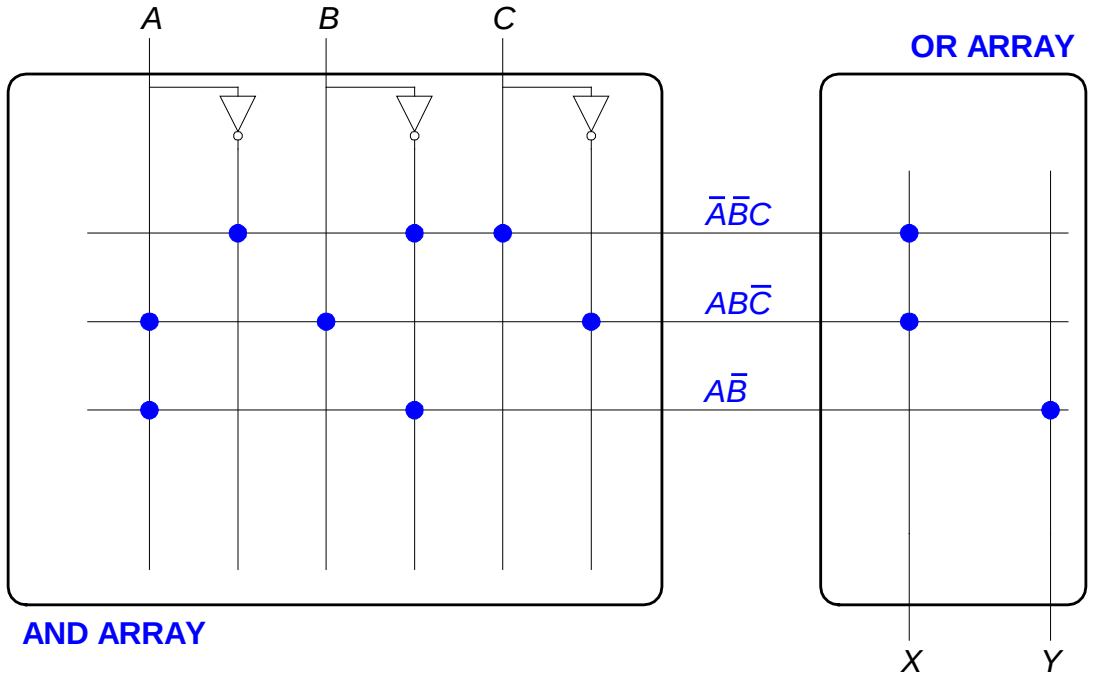
Матриці логічних елементів

- **PLAs** (ПЛМ, Programmable logic arrays) – програмовна логічна матриця
  - AND матриця, потім OR матриця
  - Тільки комбінаційна логіка
  - Фіксовані внутрішні з’єднання
- **FPGAs** (Field programmable gate arrays) – програмовна користувачем матриця логічних елементов
  - Масив конфігуровних логічних блоків CLB
  - Комбінаційна і послідовнісна логіка
  - Програмовні внутрішні з’єднання

Програмовні логічні матриці. Крапкова нотація

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

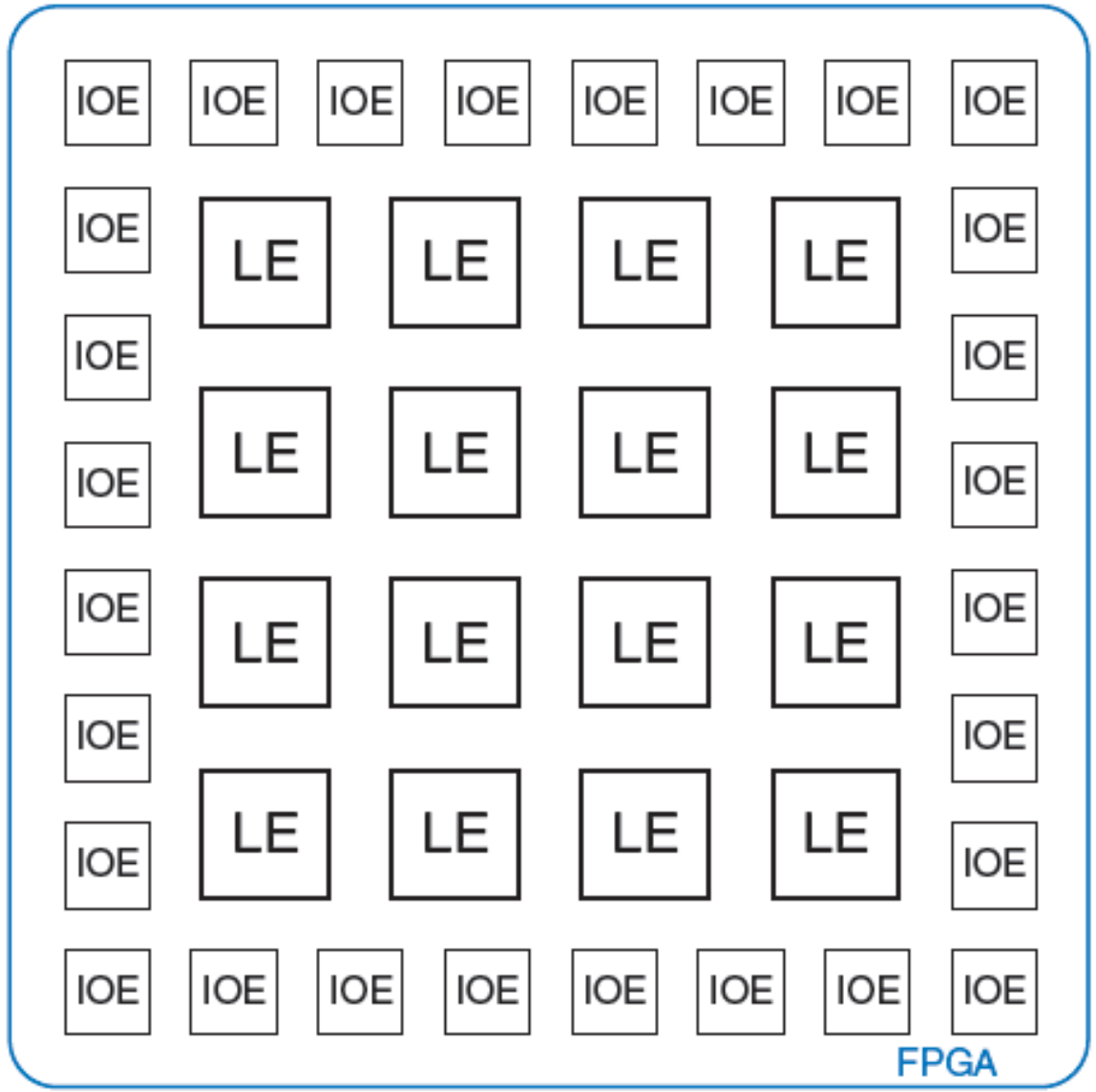




FPGA. Програмовна користувачем матриця логічних елементов

- Складається з:
  - **LE** (логічних елементів): реалізує логіку
  - **IOE** (Елементів введення/виведення): інтерфейс із зовнішнім світом
  - **Programmable interconnection:** програмовані з'єднання зв'язують логічні елементи з елементами введення/виведення
  - Деякі FPGAs вмістять і інші блоки, такі як перемножувачі і пам'ять RAMs

Узагальнена структура ПЛІС

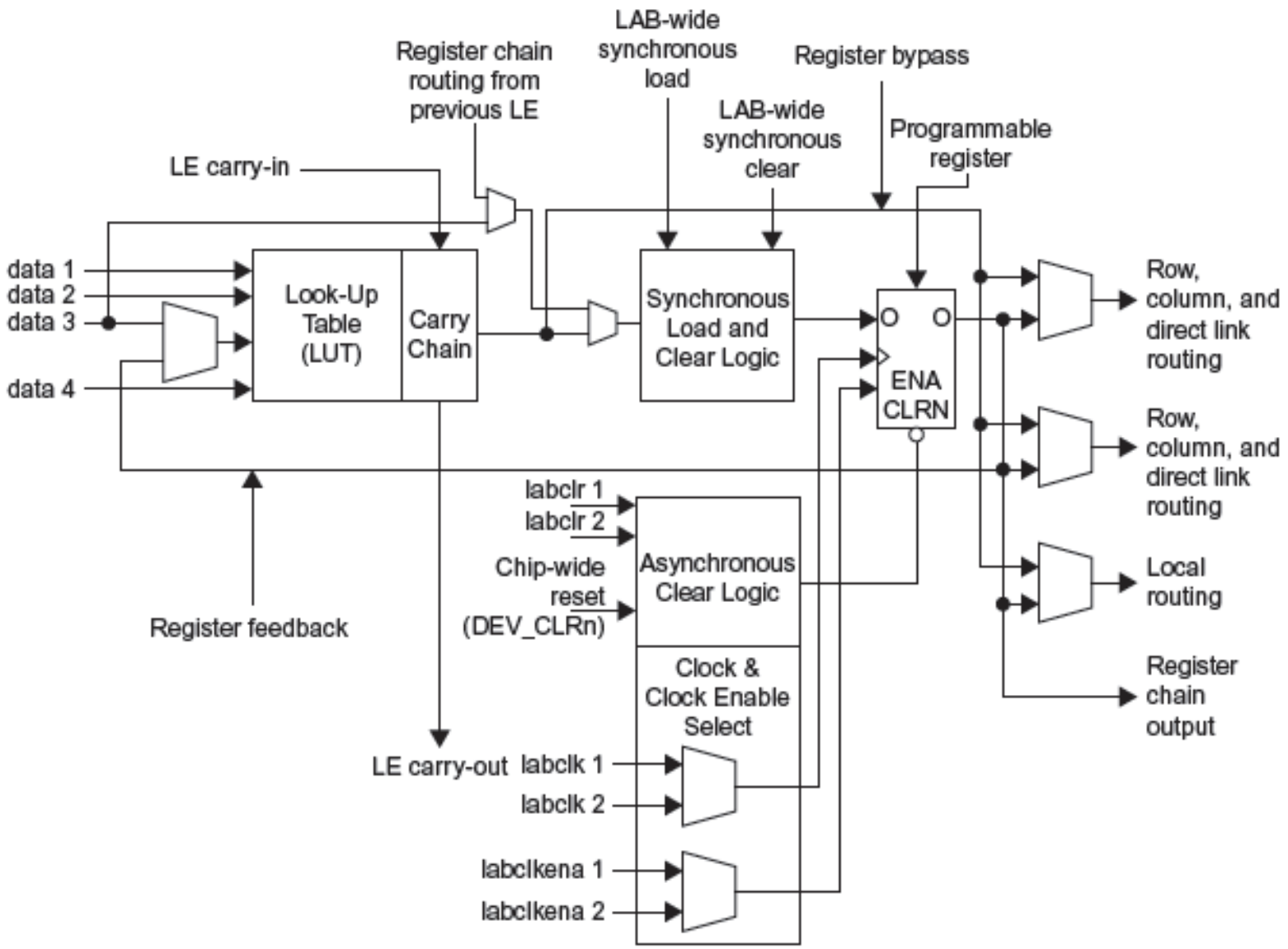


LE – логічні елементи
IOE – елементи введення/виведення

LE. Логічний елемент

- Складається з:
  - **Таблиць перетворення (LUT)**: реалізують комбінаційну логіку
  - **Тригерів:** реалізують послідовнісну логіку
  - **Мультиплексорів:** з’єднують таблиці перетворень (LUT) і тригери

ALTERA CYCLON IV LE

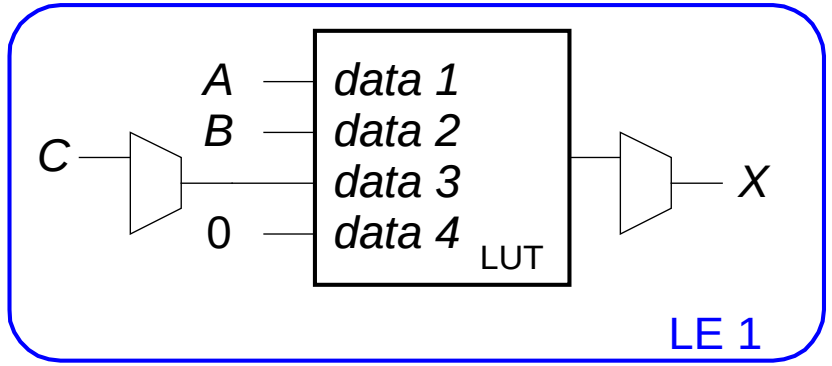


- Конфігуровні логічні блоки (CLB) Spartan мають:
  - 1 4-входову LUT
  - 1 регістровий вихід
  - 1 комбінаційний вихід

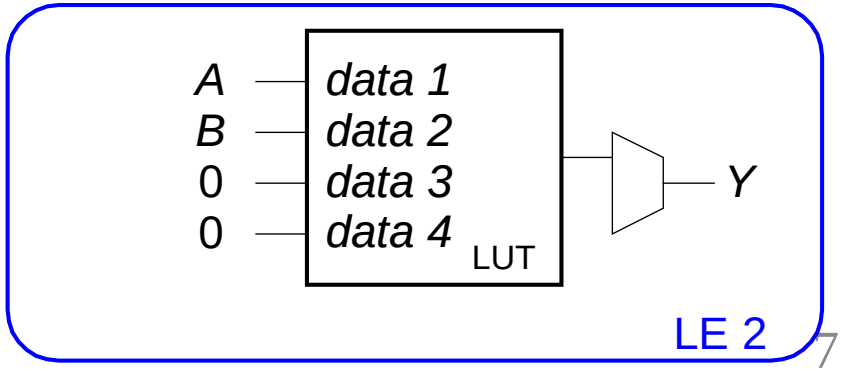
Конфігурація логічних блоків

- Зконфігуровані логічні блоки Cyclone IV для виконання наступних функцій:
  - X = $\bar{A}\bar{B}C + AB\bar{C}$
  - Y = $A\bar{B}$

(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. Послідовність проектування

Використовуючи засоби САПР (Altera’s Quartus або Xilinx Vivado):

- **Описати проект з** використанням редактора принципових електричних схем або HDL
- **Промоделювати** проект
- **Синтезувати** проект і виконати його реалізацію у FPGA
- **Завантажити конфігурацію** в FPGA
- **Протестувати** проект