Computer Architecture

Digital logic: Part 5

Digital Building Blocks

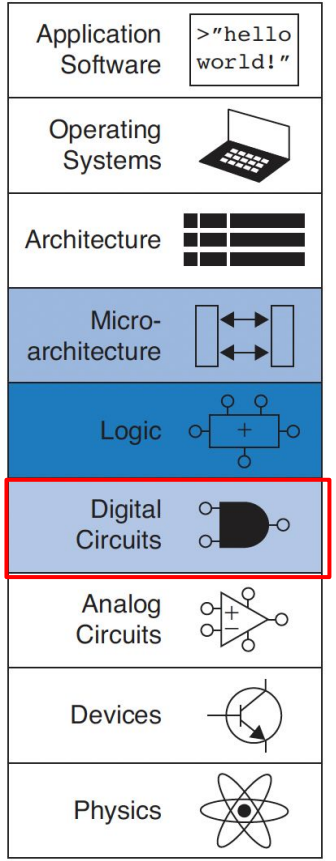
Agenda

- Carry propagate adder
  - Ripple-Carry adder
  - Carry lookahead adder
- ALU
- Shifters
- Multiplication
- Division
- Sequential Building Blocks
- Memory arrays
- Logic arrays

Digital blocks

We have examined the design of combinational and sequential circuits using Boolean equations and schematics.

We will look at some more elaborate circuit blocks.



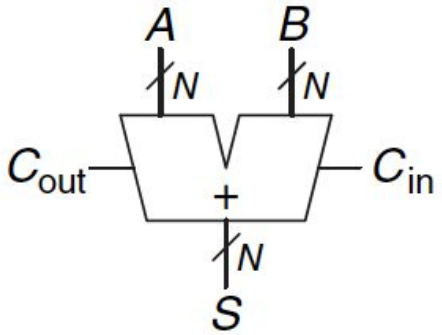
Carry propagate adder

Carry propagate adder (CPA): it is a fundamental type of adder in digital systems.

“Carry propagate” means when an *N*-bit adder sums two N-bit inputs, the carry bit is propagated from one bit position to the next, from the least significant bit to the most significant bit.

It has three common implementations:

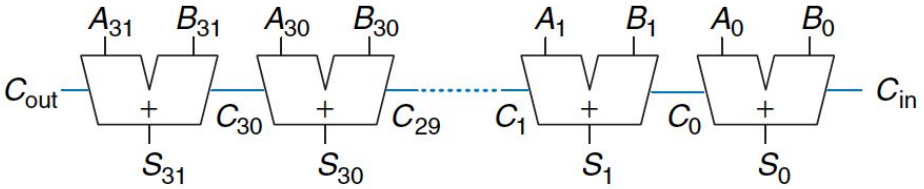
- ripple-carry adders,
- carry-lookahead adders, and
- prefix adders (please refer to H&H §5.2.1)



Ripple-Carry adder

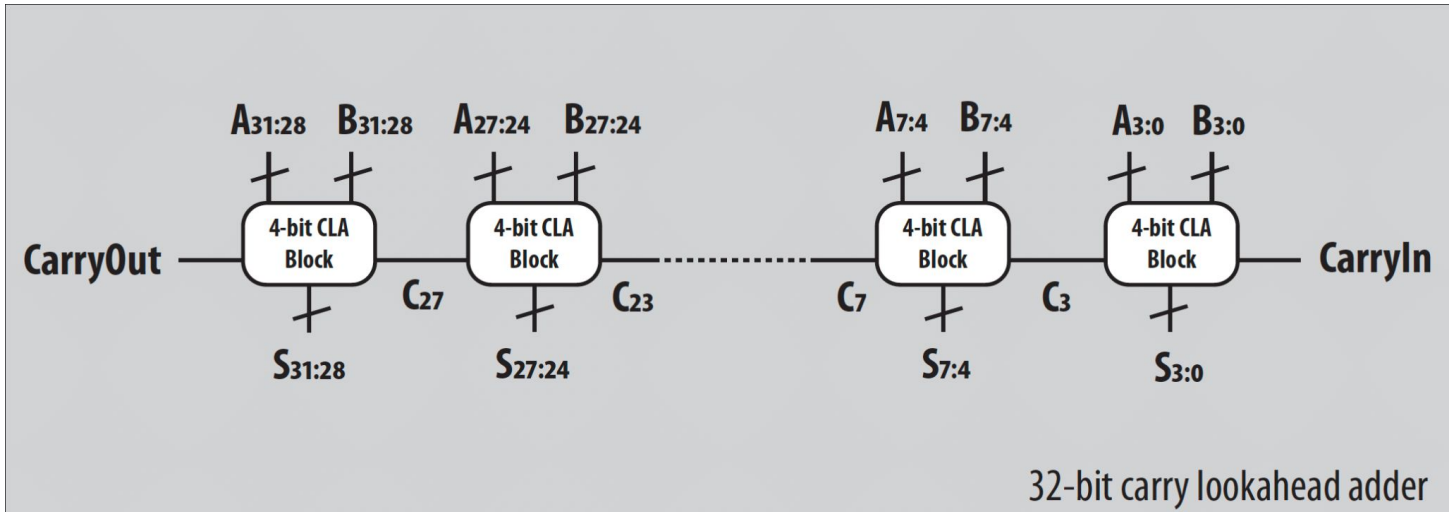
It is the simplest way to build a CPA.

- N full adders are chained;
- C_{out} of one stage is the C_{in} of the next stage.
- Advantage: reuse the full adder module, so simplify the design
- Disadvantage: being slow when N is large; Each S_i need to wait for the previous C_{i-1}.
- The delay $t_{ripple} = N*t_{FA}$, where t_{FA} is the delay of a full adder.

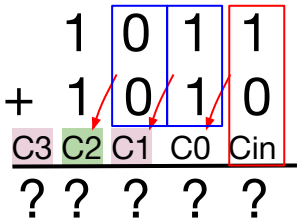


Carry lookahead adder

Carry lookahead adder (CLA): it divides the adder into blocks and provides circuitry to quickly determine the `carryOut` of a block as soon as the `carryIn` is known.



The logic for determining the carry out



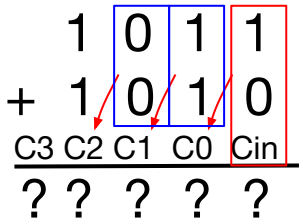
Three cases for a column to determine the carry out:

- Two bits are both 1 ⇒ CarryOut = 1
- Two bits are both 0 ⇒ CarryOut = 0
- One bit is 1, the other is 0 ⇒ CarryOut depends on the CarryIn.

In the left example,

- C0 is determined by Cin;
- C1 is always 1;
- C2 is always 0;
- C3 is always 1;

The logic for determining the carry out



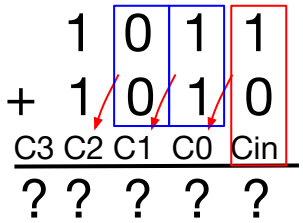
Generate signal (G) and propagate signal (P):

- Generate (G): it is 1 if a column can produce a carry out independent to its carry in.
- Propagate (P): it is 1 if a column can produce a carry out when carry in is 1.

Let A, B be two binary numbers in the addition; A_i, B_i are the bits of the $i\text{-th}$ column, and C_i is the $i\text{-th}$ carry out. We have,

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

Carry lookahead adder



Generate (G) and propagate (P) on multiple blocks:

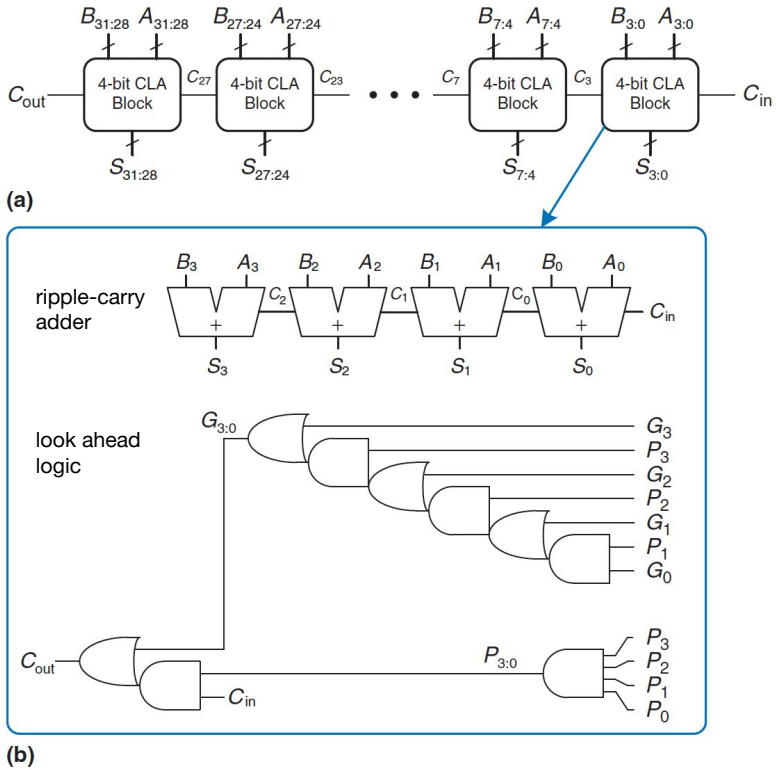
- $G_{i:j}$ and $P_{i:j}$ represent the signals for blocks spanning columns i through j.
- A block generates a carry if the most significant column generates a carry,
- A block propagates a carry if all columns in the block propagates the carry.

So, for a block spanning column 3 through 0, we have

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

For the i-th block, we have $C_i = G_{i:j} + P_{i:j} C_j$.

Example: a 32-bit Carry lookahead adder

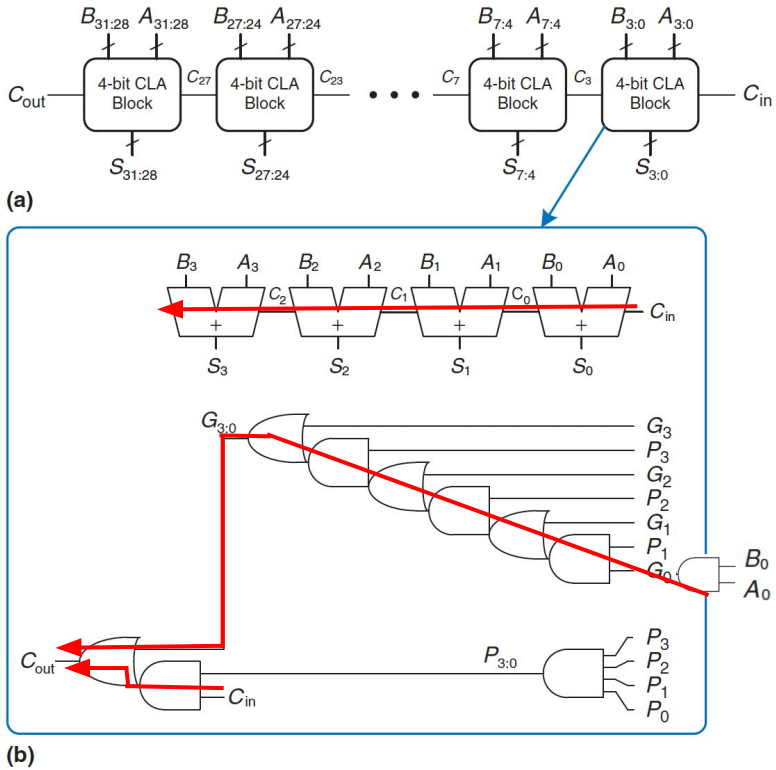


It has eight 4-bit blocks; each block contains a 4-bit ripple-carry adder and some look ahead logic.

The AND and OR gates are used to compute the C_{out} by

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

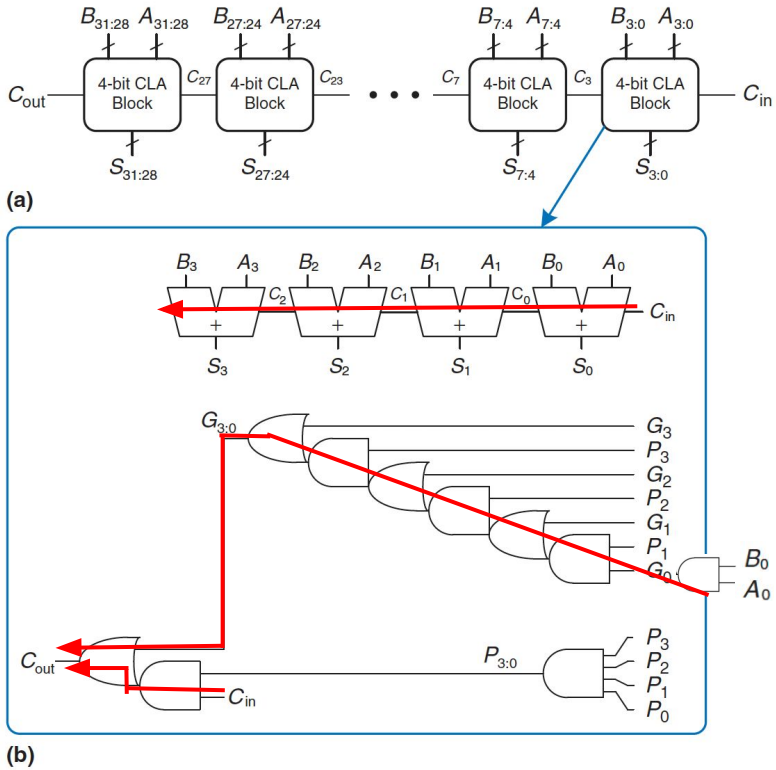
Paths in the Carry lookahead adder



The paths in the block:

- 4 adders for calculating S_i;
- 4 AND gates and 4 OR gates for calculating the $G_{3:0}$ and C_{out};
- The AND gate and OR gate to handle C_{in}

Propagation delay of the CLA



The propagation of the *N*-bit CLA with *k*-bit blocks:

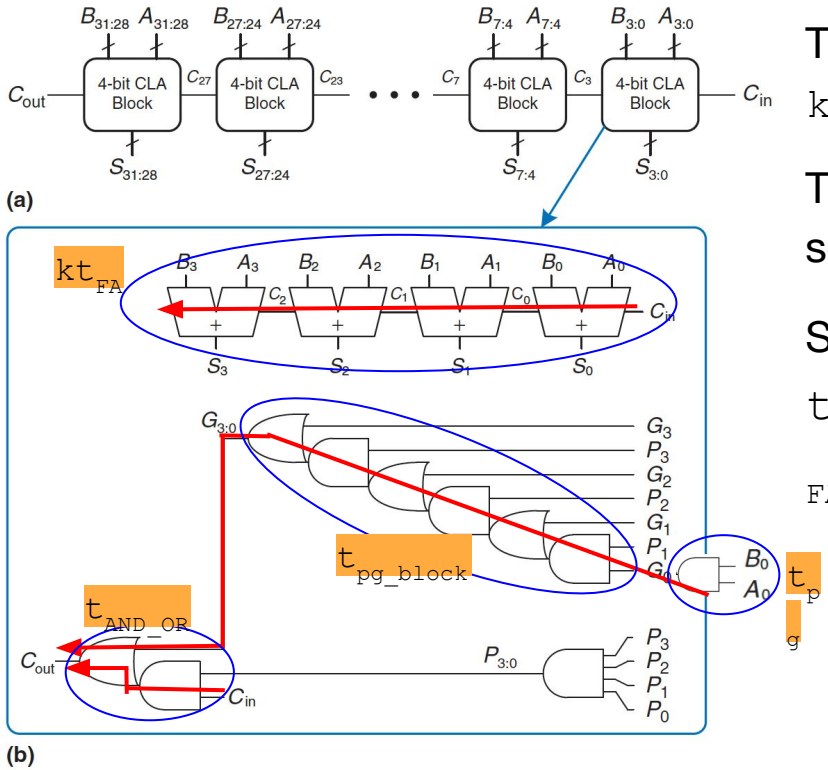
$$t_{CLA} = t_{pg} + t_{pg\_block} + ((N/k) - 1)\, t_{AND\_OR} + k t_{FA}$$

where

- kt_{FA}: k adders for calculating S_i; (i = 0, 1, …, k-1)
- t_{pg}: 1 AND (or OR) gate for calculating G_i (or P_i);
- t_{pg_block}: 3 AND gates and 3 OR gates for calculating the $G_{3:0}$ and C_{out};
- t_{AND_OR}: the AND gate and OR gate to handle C_{in} in each block.

(**Note**: the least significant block has no t_{AND_OR}.)

Propagation delay of the CLA



The CLA has (N/k) blocks, and they process the k bits simultaneously.

The C_{in} passes sequentially from left to right, so we need to multiply $(N/k)-1$ with t_{AND_OR}.

So, the delay of CLA is as the following,

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

Exercise 1: (from H&H Example 5.1)

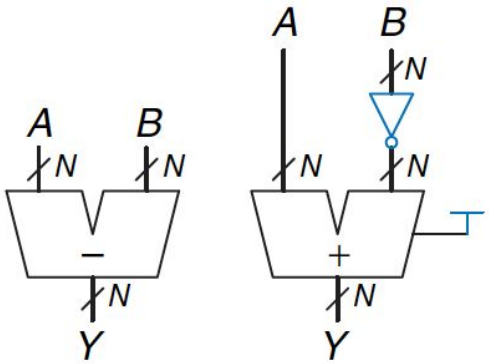
Compare the delays of a 32-bit ripple-carry adder and a 32-bit carry-lookahead adder with 4-bit blocks. Assume that each two inputs gate delay is 100 ps and that a full adder delay is 300 ps.

- Ripple-carry adder: `32*300ps = 9.6ns`
- CLA: t_{pg}=100ps, t_{pg_block}=6*100=600ps, and t_{AND_OR}=2*100 =200ps.

  So, the propagation delay is `100+600+(32/4-1)*200+4*300=3.3ns`, which is 3 times faster than ripple-carry adder.

Subtraction

- Subtraction can be implemented by adding positive and negative numbers using two's complement number representation.



Symbol of subtractor

$Y=A-B$, $\Rightarrow$ $Y=A+\overline{B}+1$.

So, given an adder, we can do subtraction by flipping the input B with a NOT gate, and set the C_{in} to be 1.

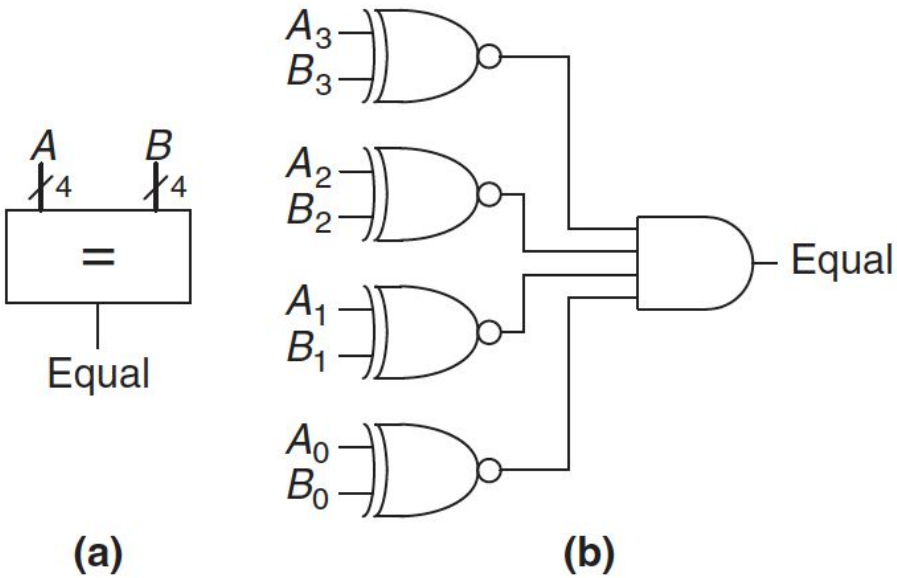
Comparators

It receives two *N*-bit binary numbers and determines if the two binary numbers are equal or if one is greater or less than the other.

Two types of comparators:

- Equality comparator: it produces a single output indicating if A==B.
- Magnitude comparator: it produces one or more outputs indicating the relative values of A and B.

Equality comparator

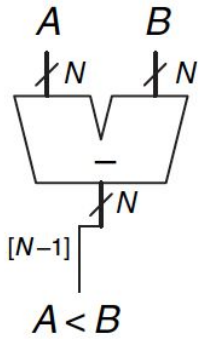


It checks if the corresponding bits in each column of A and B are equal using `NOT` `XOR` gates.

Magnitude comparator

Magnitude comparison is done by computing `A-B` and looking at the sign of the result.

- If the sign bit (i.e., the most significant bit) is 1, then `A<B`. Otherwise, `A >= B`.
- `[N-1]` represents the most significant bit.

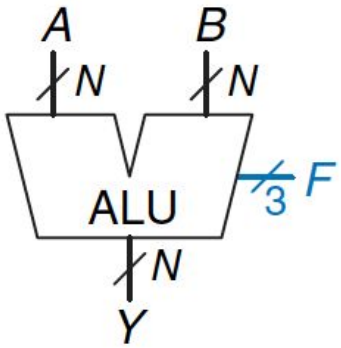


Arithmetic Logical Unit

ALU combines a variety of mathematical and logical operations into a single unit. It is the heart of computer systems.

Table 5.1 ALU operations

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

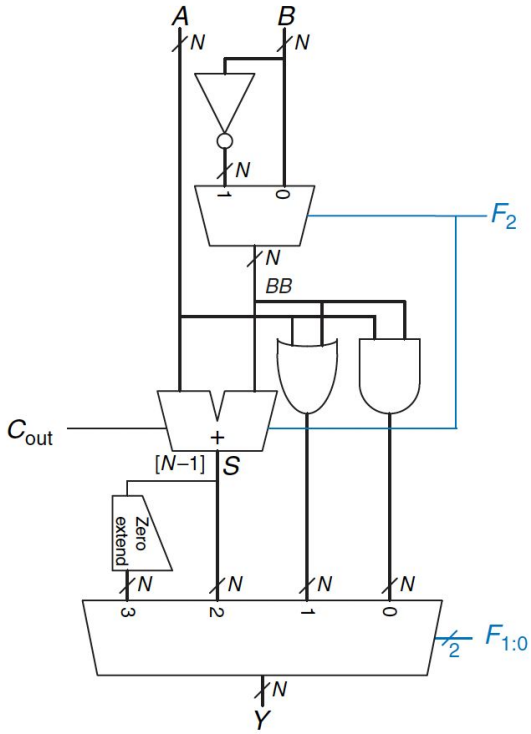


- The ALU receives a control signal F that specifies which function to perform.
- In circuit diagrams, control signals are shown in blue to distinguish them from the data.

ALU implementation

Table 5.1 ALU operations

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



The ALU has an N-bit adder and N two-input `AND` and `OR` gates.

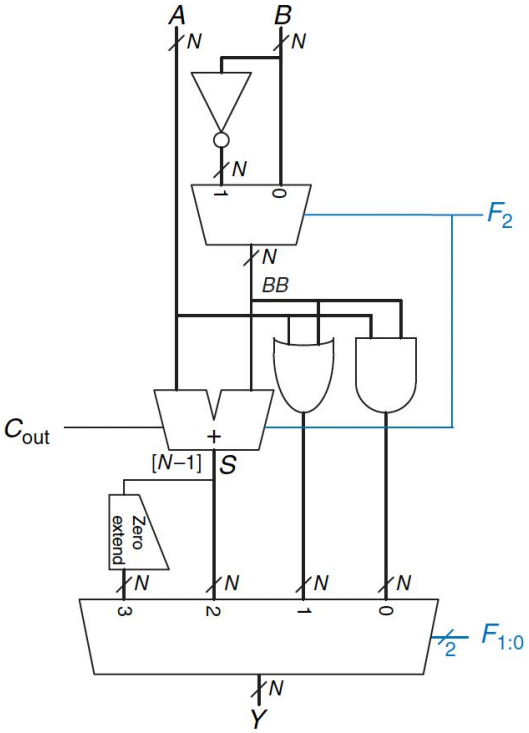
It also has inverters and a multiplexer to invert `B` when F_2 is asserted.

A **4:1** mux chooses the desired function based on the $F_{1:0}$ control signals.

ALU implementation

Table 5.1 ALU operations

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



Suppose `F=110` ⇒ $F_{1:0}$=10, F_2=1;

F_2=1 ⇒ `B` is inverted; the carry in of the adder is 1;

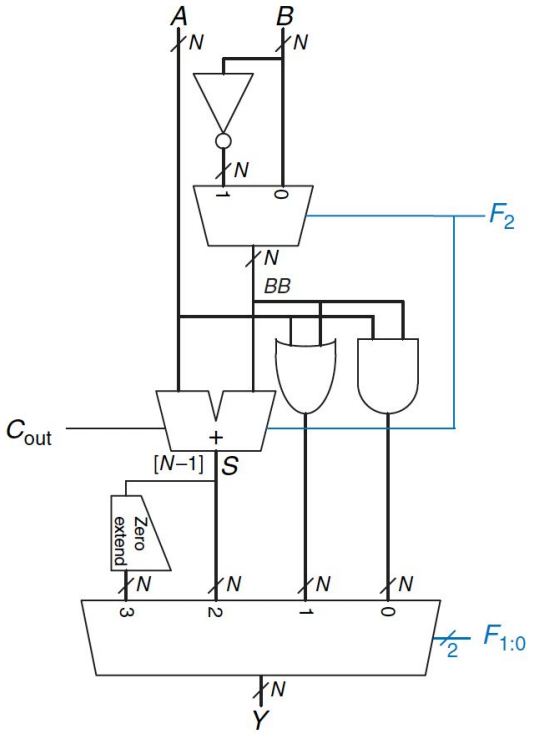
$F_{1:0}$=10 ⇒ ALU performs arithmetic operations; One input is `A`, the other one is $\overline{B}$, and the `carryIn = 1`.

So, the operation is `A+`$\overline{B}$`+1`, which is equivalent to `A-B`.

Exercise 2: (H&H Example 5.3)

Table 5.1 ALU operations

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT



Show the execution of the `SLT` operation. Suppose $A=25_{10}$ and $B=32_{10}$. Show the control signals and output `Y`.

For `SLT`, $F_{2:0}=111$. With $F_2=1$, the adder performs the subtraction. Since $25-32<0$, the output of the adder `S` is $111\ldots1001_2$. So, the most significant bit `[n-1]` is `1`. It goes into the zero extend unit, and the output is the fourth input (S_{31}) of the mux.

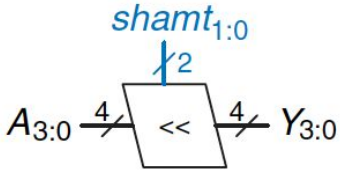
With $F_{1:0}=11$, the mux sets $Y=S_{31}=000\ldots01_2$.

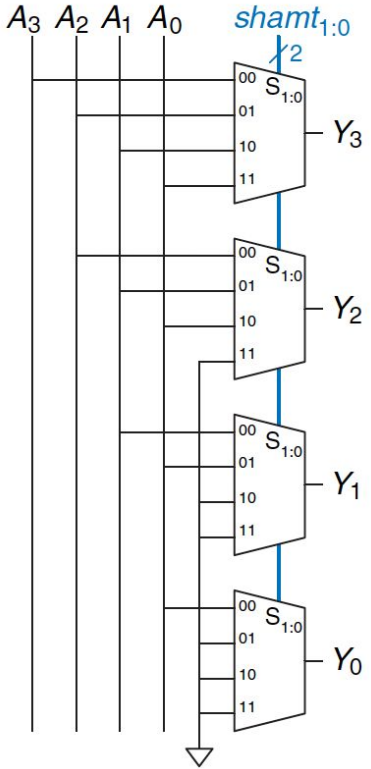
Shifters

Shifter moves bits and multiply or divide by powers of 2. There are several commonly used shifters:

- **Logical shifter**: it shifts the number to the left (LSL) or right (LSR) and fills empty spots with 0's.
  - e.g., `11001 LSR 2=00110; 11001 LSL 2=00100`
- **Arithmetic shifter**: it is the same as a logical shifter, but on right shifts, it fills the most significant bits with a copy of the old most significant.
  - e.g., `11001 ASR 2=11110; 11001 ASL 2=00100`
- **Rotator**: it rotates number in circle such that empty spots are filled with bits shifted off the other end.
  - e.g., `11001 ROR 2=01110; 11001 ROL 2=00111`

Shift Left



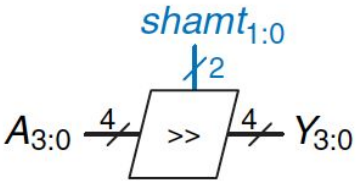


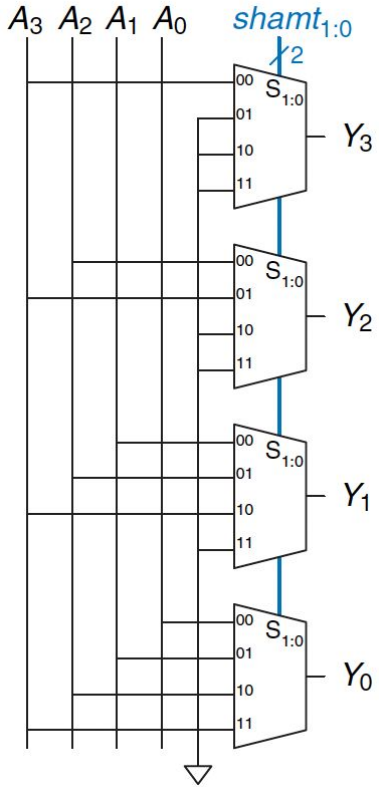
It is a 4-bit shifter.

shamt controls the multiplexes so to shift the value of Y_i from A_i.

- shamt=00 ⇒ Y = A
- shamt=01 ⇒ $Y_3Y_2Y_1Y_1=A_2A_1A_0$0

Shift Right



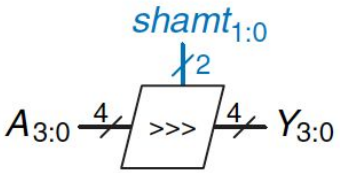


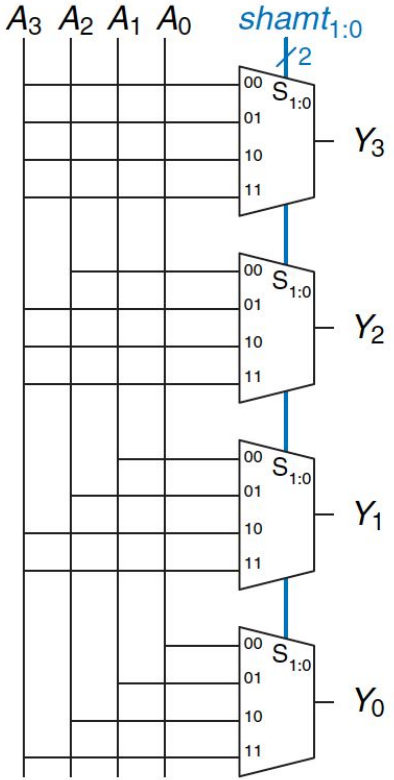
It is a 4-bit shifter.

shamt controls the multiplexes so to shift the value of Y_i from A_i.

- shamt=00 ⇒ Y = A
- shamt=01 ⇒ $Y_3Y_2Y_1Y_1=0A_3A_2A_1$

Arithmetic Shift Right





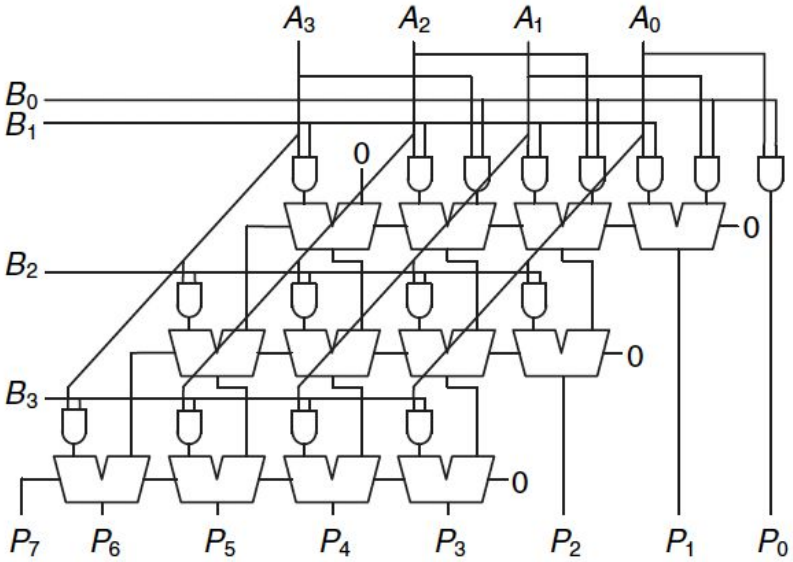
It is a 4-bit shifter.

shamt controls the multiplexes so to shift the value of Y_i from A_i.

- shamt=00 ⇒ Y = A
- shamt=01 ⇒ $Y_3Y_2Y_1Y_1=A_3A_3A_2A_1$

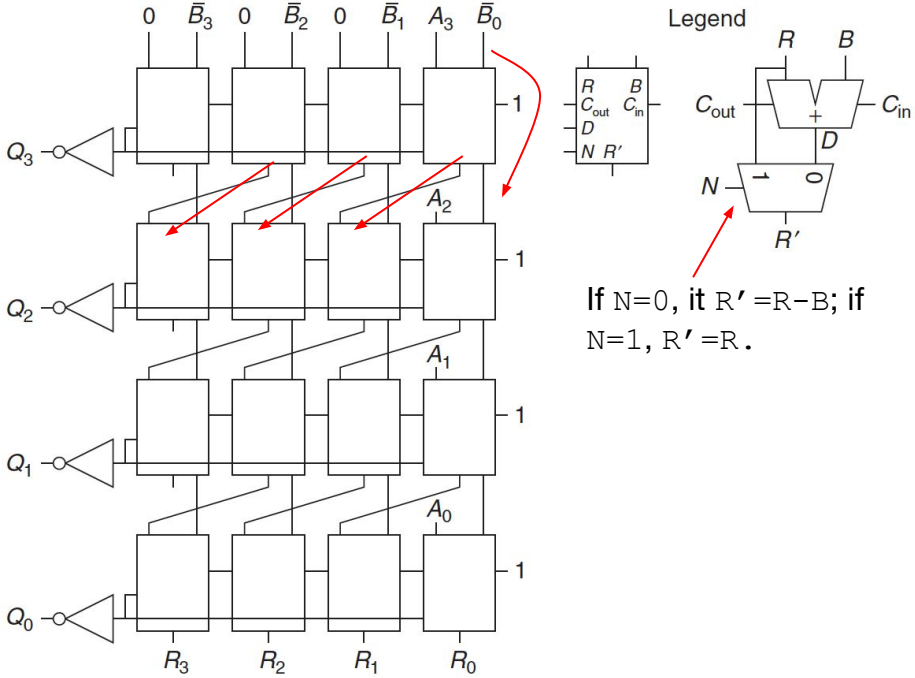
Multiplication

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



Another implementation of multiplication: using 1-bit adders and `AND` gates.

Division



- 1st row: $000A_3$ + (-B)
  - If the sign bit is 1 (i.e., negative), Q_3=0; otherwise, 1.
- 2nd row: [Reminder of 1st row: A_2] + (-B)
- …

$$B_3B_2B_1B_0 \overline{)\ A_3A_2A_1A_0}$$

1. $0\ 0\ 0\ A_3\ A_2\ A_1\ A_0$
   $-\ B_3\ B_2\ B_1\ B_0$
2. $0\ 0\ A_3\ A_2\ A_1\ A_0$
   $-\ B_3\ B_2\ B_1\ B_0$

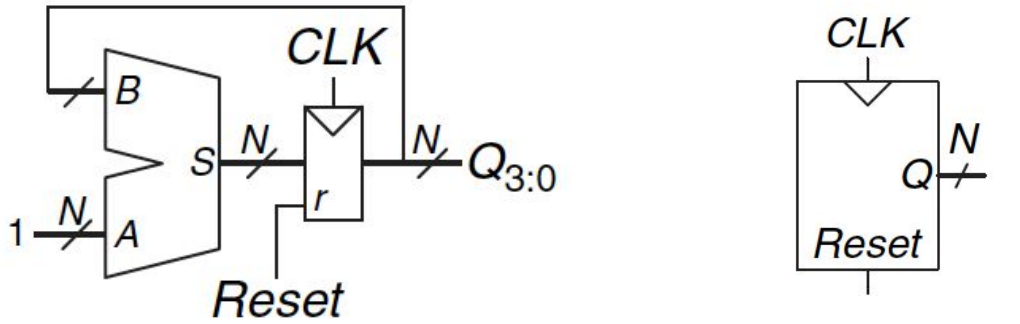
……

Sequential Building Blocks

- Counters
- Shift registers

Counters

An N-bit binary counter is a sequential arithmetic circuit with clock and reset inputs and an N-bit output Q.

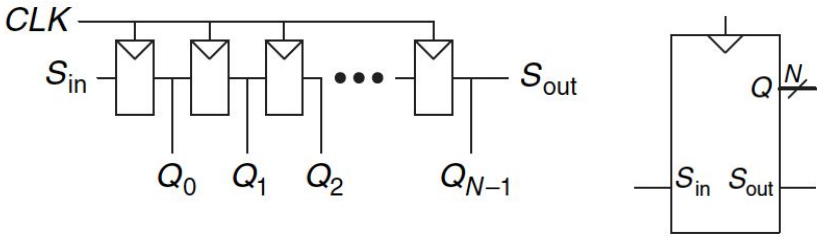


The schematic and symbol of N-bit counter. Reset initializes the output to 0. The counter increment by 1 on the rising edge of the clock. It can be built by an adder and a resettable register.

Shift Register

A shift register has a clock, a serial input S_{in}, a serial output S_{out}, and N parallel outputs $Q_{N-1:0}$. On each rising edge of the clock,

- a new bit is shifted in from S_{in} and all the subsequent contents are shifted forward;
- the last bit in the shift register is available at S_{out}.

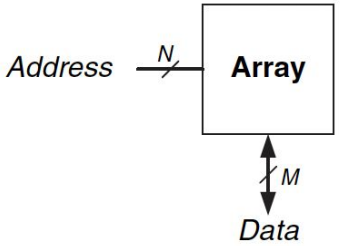


The schematic and symbol of a shift register. It is constructed from *N* flip-flops connected in series. The input is provided one bit at a time at S_{in}; after *N* cycles, the past *N* inputs are available in parallel at $Q_0Q_1\ldots Q_{N-1}$.

Memory arrays

Memory is organized as a two-dimensional array of memory cells.

- It reads or writes the contents of one of the rows of the array.
- Each row is specified an address
  - the value read or written is called data.
  - each row of data is called a word.
- An array with N-bit address and M-bit data has 2^N rows and M columns.

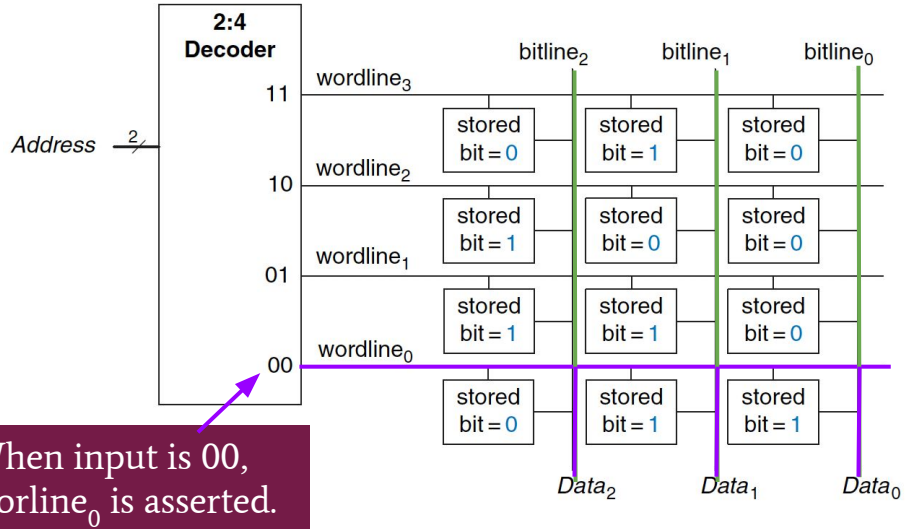


The symbol of a memory array of 2^N rows and M columns, i.e., it contains 2^N M-bit words.

Memory arrays

Memory arrays are built as an array of bit cells.

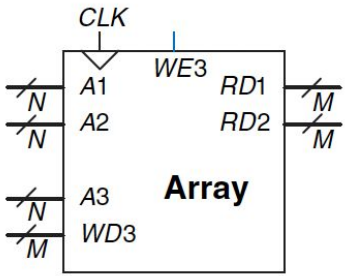
- Each bit cell stores 1 bit of data and is connected to a worldline and a bitline.
- For each address, the memory asserts a single worldline that activates the bit cells in that row.
- Types of memory
  - RAM
  - ROM



Memory ports

Memories can have one or more ports.

- Each port gives read and/or write access to one memory address.
- Multi-ported memories can access several addresses **simultaneously**.

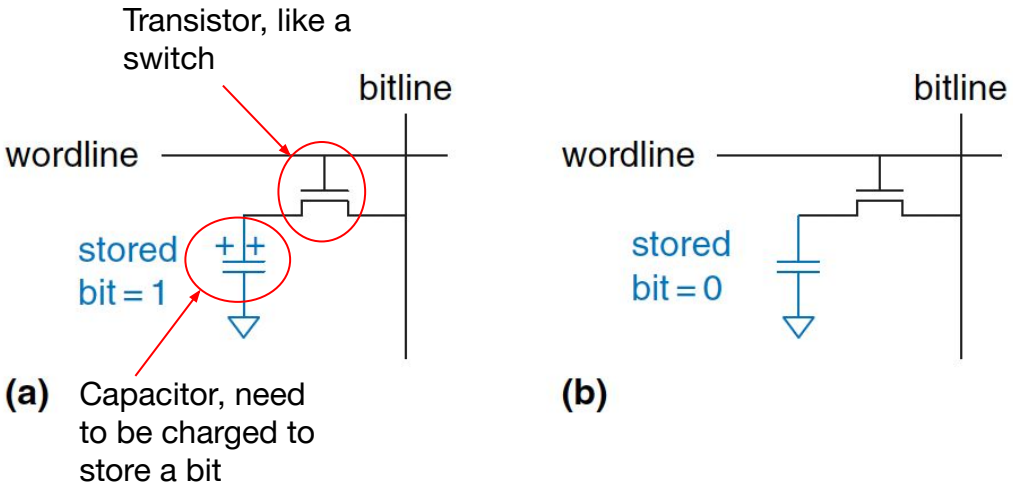


The left figure is a three-ported memory. It has two read ports (A1 and A2) and one write port (A3).

- Port 1 reads the data from address A1 onto the read data output RD1.
- Port 2 reads the data from address A1 onto the read data output RD2.
- Port 3 writes the data from the write data input WD3 into address A3 on the rising edge of the clock if the write enable WE3 is asserted.

Dynamic Random Access Memory (DRAM)

DRAM stores a bit using a capacitor.



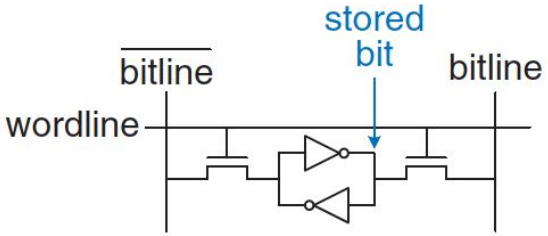
On a read, data values are transferred from the capacitor to the bitline. This destroys the bit values stored on the capacitors, so the data word must be restored (rewritten) after each read.

On a write, data values are transferred from the bitline to the capacitor.

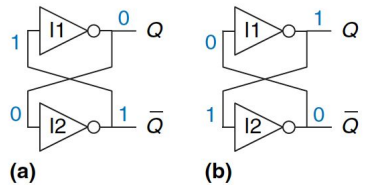
Actually, DRAM must be refreshed (read and written) every few milliseconds because the charge on the capacitor gradually leaks away. (This is why it is called dynamic RAM.)

Static Random Memory (SRAM)

SRAM stores a bit on cross-coupled inverters.



Cross-coupled inverters: it has two stable states, Q=0 and Q=1.



Bitline and the Bitline-bar (complementary bitline) are connected to the sense amplifier, which reads the stored bit by comparing the voltage difference between them.

When the wordline is asserted, Q and Q-bar are connected to the bitline and bitline-bar, respectively. Then, the sense amplifier detects the small voltage difference between BL and BL-bar and amplifier it quickly, which is called differential sensing.

The differential sensing allows the amplifier to achieve a fast and accurate read, with better noise rejection than single-ended sensing would provide.

Memory latency

Table 5.4 Memory comparison

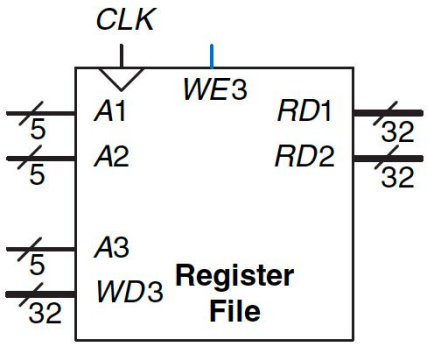
Memory Type	Transistors per Bit Cell	Latency
flip-flop	~20	fast
SRAM	6	medium
DRAM	1	slow

In general, the more transistores a device has, the more area, power, and cost it requires.

- Flip-Flop: data on flip-flop is available immediately.
- DRAM: must wait for charge to move slowly from the capacitor to the bitline. Aslo, it must refresh data periodically.
- Memory latency also depends on memory size: Larger memories tend to be slower than smaller ones if all else is the same.
- The design of memory is a tradeoff among the speed, cost and power constraints.

Register files

- Register file is a number of registers to store temporary variables, which can be built as a small, multi-ported SRAM array.
  - SRAM array is more compact than an array of flip-flops.

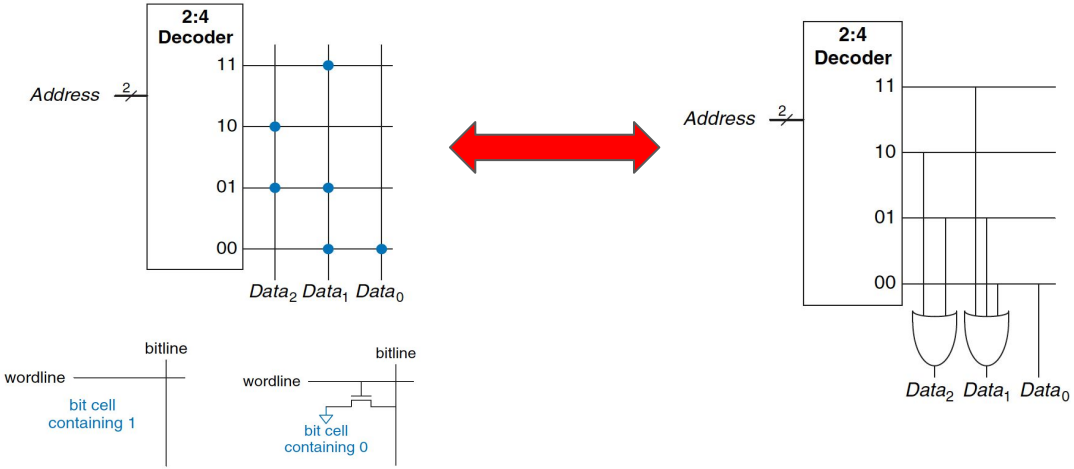


The left figure shows a 32-register x 32-bit three-ported register file built from a three-ported memory.

- It has two read ports (A1/RD1, and A2/RD2) and one write port (A3/WD3).
- A1, A2, and A3 have 5-bit long input signals, respectively, so each of them can access $2^5 = 32$ registers.
- This register file can read two registers and write one register simultaneously.

Read Only Memory (ROM)

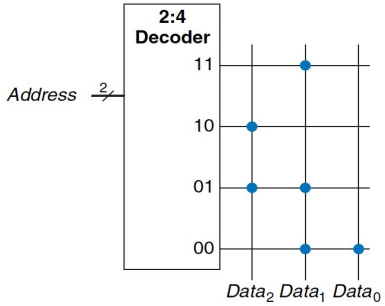
ROM: it stores a bit as the presence or absence of a transistor, and it is non-volatile.

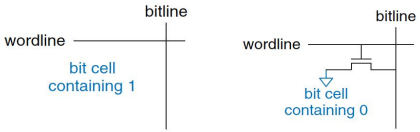


- Conceptually, a ROM can be built using two-level logic with a group of AND gates followed by a group of OR gates.
- The AND gates produce all possible minterms and hence form a decoder.

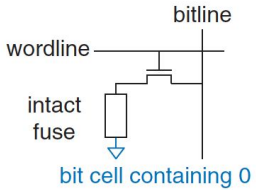
Read Only Memory (ROM)

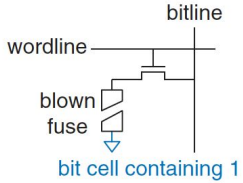
ROM: it stores a bit as the presence or absence of a transistor, and it is non-volatile.





- The contents of the ROM bit are specified during manufacturing by the presence or absence of a transistor in each bit cell.



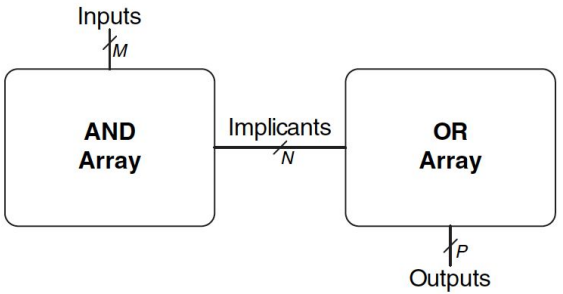


Fuse-programmable ROM: If the fuse is present, the transistor is connected GND and the cell holds a 0. If the fuse is destroyed, the transistor is disconnected from ground and the cell holds a 1. (It is also called a one-time programmable ROM.)

- Modern ROMs are not really read only; they can be written (programmed) as well. The difference between RAM and ROM is that ROMs take a longer time to write but are nonvolatile.

Logic Arrays

Logic gates can be organized into regular arrays. If the connections of gates are made programmable, the logic arrays can be configured to perform any function.



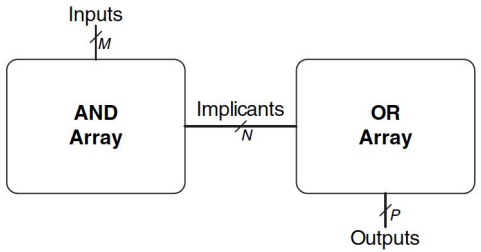
<u>Two types of logic arrays:</u>

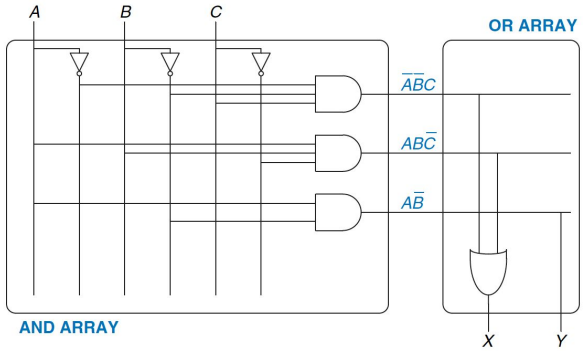
- PLAs: programmable logic arrays
- FPGAs: field programmable gate arrays

Programmable Logic Arrays

PLA: it implements two-level combinational logic in sum-of-products.

- It is built from an AND array followed by an OR array.
- The inputs (in true and complementary form) drive an AND array, and are ORed together in the OR array to form the outputs.





Field Programmable Gate Array (FPGA)

- FPGA is an array of reconfigurable logic elements (LEs).

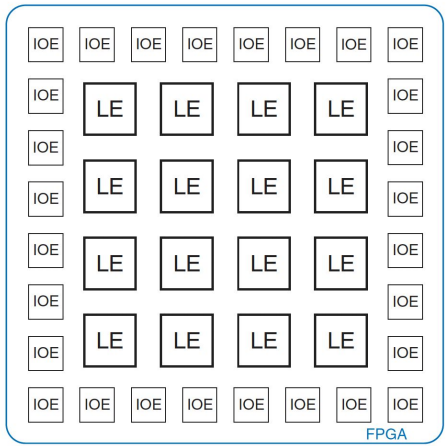
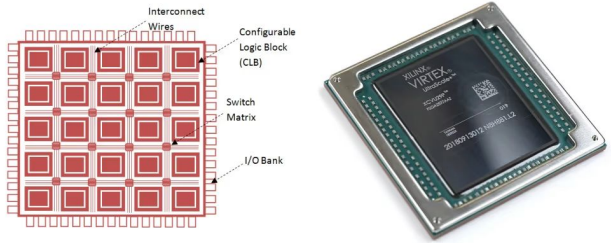


Figure 5.57 General FPGA layout

- An LE can be configured to perform combinational or sequential functions.
- The LEs are surrounded by input/output elements (IOEs) for interfacing with the outside world.
- The IOEs connect LE inputs and outputs to pins on the chip package.

Field Programmable Gate Array (FPGA)

- A user can implement designs on the FPGA using specific software programming tools.
- FPGAs are more powerful and flexible than PLAs.
  - They can implement both combinational and sequential logic.
  - They can also implement multi-level logic functions, while PLAs can only implement two-level logic.



Assignment 6: Q3

A	B	C	D	Y
0	0	0	0	X
0	0	0	1	X
0	0	1	0	X
0	0	1	1	0
0	1	0	0	0
0	1	0	1	X
0	1	1	0	0
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	X
1	1	1	1	1

Find a minimal Boolean equation for the function in the following truth table. Remember to take advantage of the don't care entries, and sketch a circuit for the function. Does your circuit have any potential glitches when one of the inputs changes? If not, explain why not. If so, show how to modify the circuit to eliminate the glitches.

CD\AB	00	01	11	10
00	x	0	1	1
01	x	x	1	0
11	0	x	1	1
10	x	0	x	x