3. Combinational Circuits II

Aims

- to show how logic gates can be combined to produce useful circuits such as

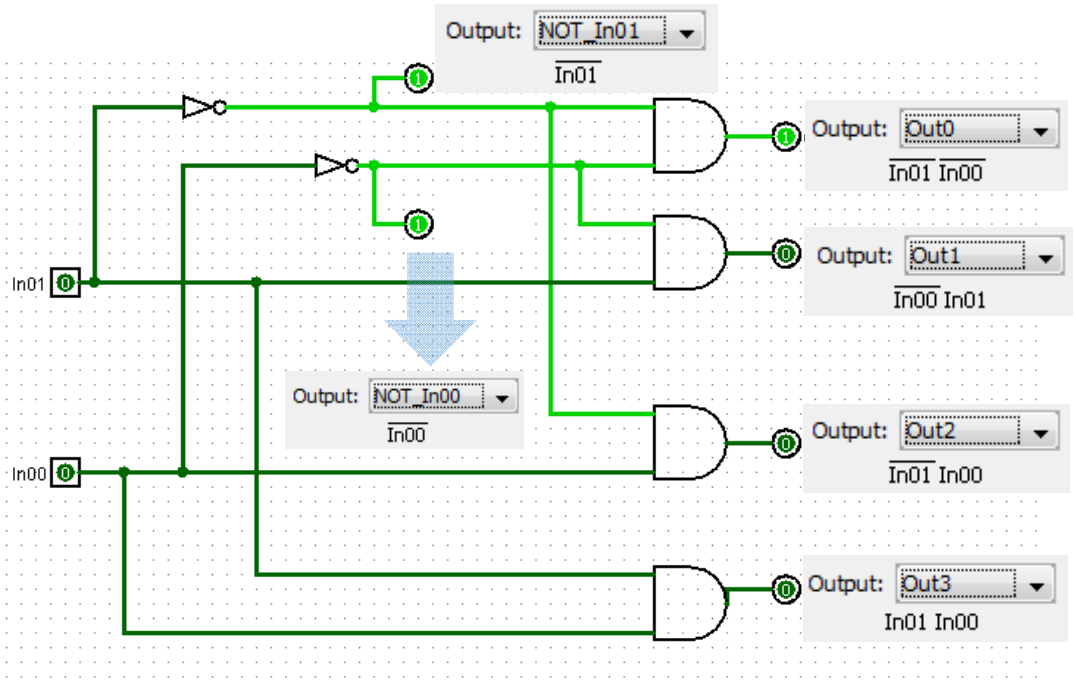
  decoders and **adders**

- to show how individual combinational circuits can be combined to produce a simple **ALU**[1]

- to introduce a simple ***memory storage*** device called a **register**

1. Arithmetic & Logic Unit

3.1 Decoders

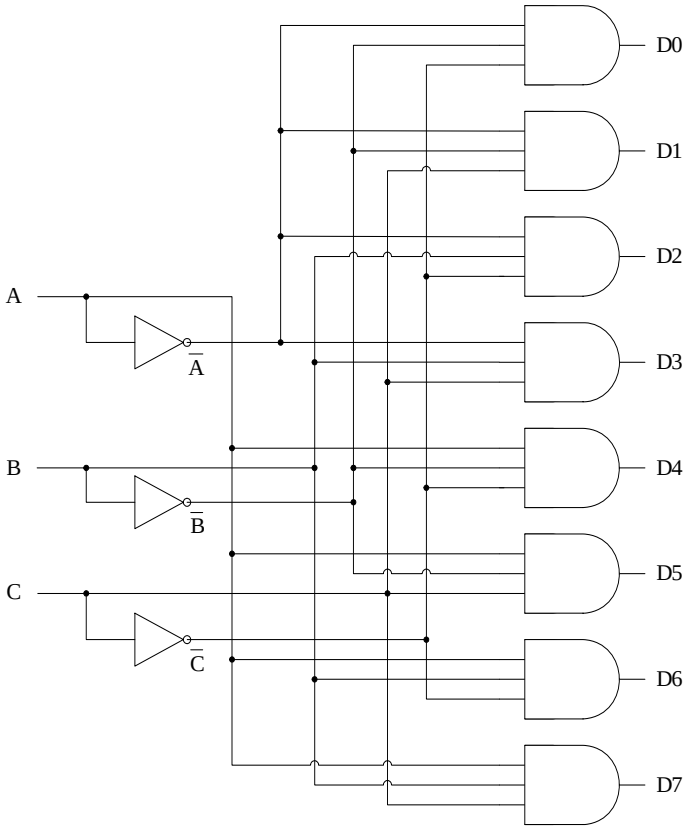


- A **decoder** is a combinational circuit that converts binary information from
  n inputs to a maximum of
  2^n unique outputs

- A decoder takes an
  ***n*-bit number** as input and uses it to select (set to 1) exactly
  one of the **2^n outputs**

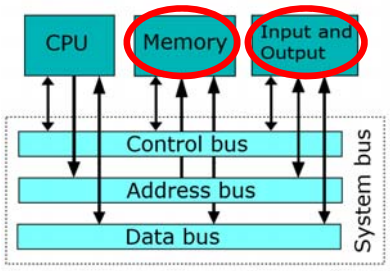
In00	In01	NOT_In00	NOT_In01	Out0	Out1	Out2	Out3
0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0
1	0	0	1	0	0	1	0
1	1	0	0	0	0	0	1

3.1.1 Line Decoders



- For each possible input combination there are **seven** outputs that are equal to **0** and only one that is equal to **1**.
- The output equal to **1** represents the equivalent of the binary number that is applied to the inputs.

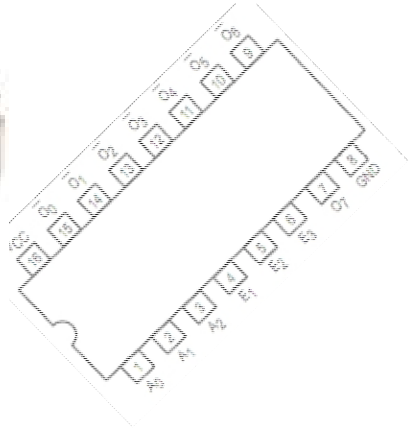
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	**1**	0	0	0	0	0	0	0
0	0	1	0	**1**	0	0	0	0	0	0
0	1	0	0	0	**1**	0	0	0	0	0
0	1	1	0	0	0	**1**	0	0	0	0
1	0	0	0	0	0	0	**1**	0	0	0
1	0	1	0	0	0	0	0	**1**	0	0
1	1	0	0	0	0	0	0	0	**1**	0
1	1	1	0	0	0	0	0	0	0	**1**



A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Application:

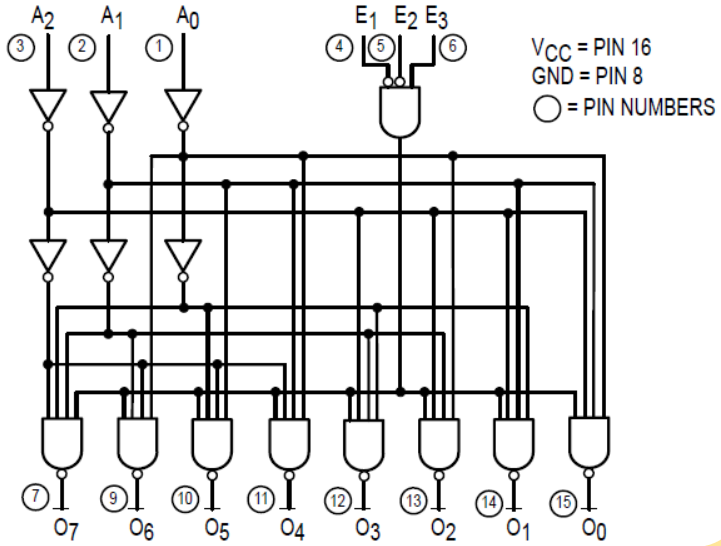
device selector



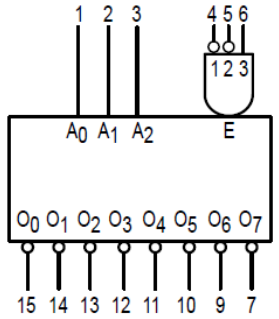
MOTOROLA **SN54/74LS138**

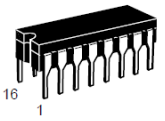
1-OF-8 DECODER/ DEMULTIPLEXER

LOGIC DIAGRAM

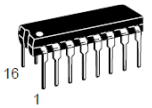


LOGIC SYMBOL





J SUFFIX
CERAMIC
CASE 620-09



N SUFFIX
PLASTIC
CASE 648-08

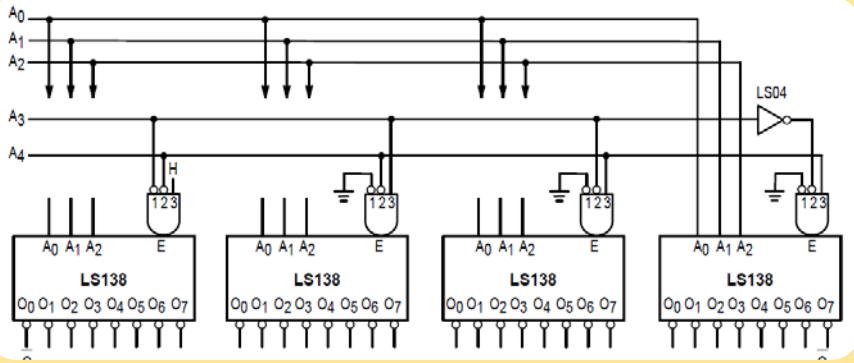
D SUFFIX
SOIC
CASE 751B-03

ORDERING INFORMATION

SN54LSXXXJ	Ceramic
SN74LSXXXN	Plastic
SN74LSXXXD	SOIC

INPUTS						OUTPUTS							
E_1	E_2	E_3	A_0	A_1	A_2	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care



3.2.1 Half Adder

- A **half adder** is an arithmetic circuit that performs the

  addition of *two* bits

- The circuit has two inputs (**A**, **B**) and two outputs (**Sum**, **Carry**)

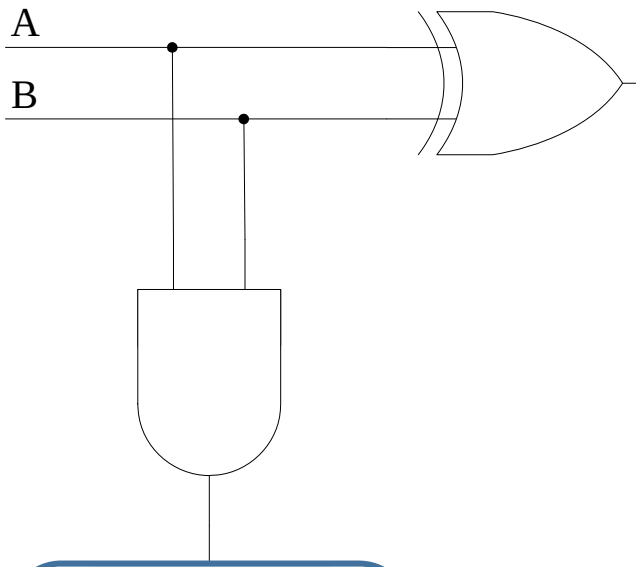
- Two outputs are necessary ....

  because the

  sum of two binary digits ranges from **0** to **2**

  and

  the binary equivalent of 2 requires **two** digits



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

The simple addition performed by a **half adder** can consist of *four* possible operations :

0+0	=	**0**
0+1	=	**1**
1+0	=	**1**
1+1	=	**10**

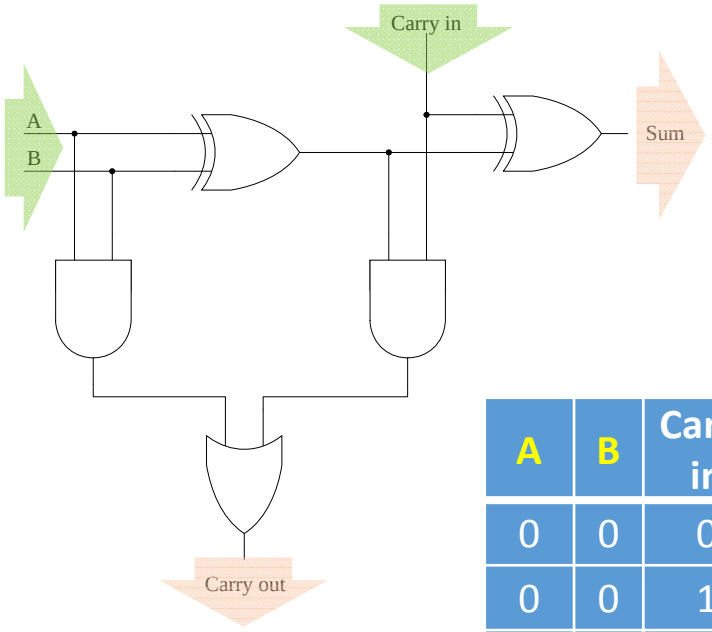
A	B	Carry	Sum
0	0	**0**	**0**
0	1	**0**	**1**
1	0	**0**	**1**
1	1	**1**	**0**

3.2.2 Full Adder

- A full adder is a circuit that performs the **addition of three bits**
- The name of the circuit stems from the fact that

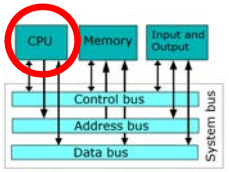
  two half adders are joined together to create a ***full adder***
- The circuit has **three inputs** (**A**, **B**, **Carry in**) and **two outputs** (**Sum**, **Carry out**)

A	B	CarryIn	CarryOut	Sum
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

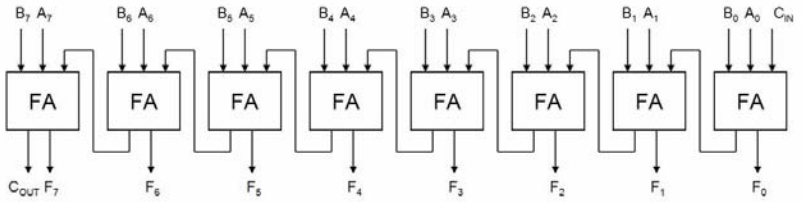


Two ***outputs*** are necessary because the sum of three binary digits ranges from 0 to 3 and the binary equivalent of 2 or 3 requires **two digits**

A	B	Carry in	Carry out	Sum
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**



- To build an adder for two **8-bit numbers** the full adder is replicated ***eight*** times



- The **carry out** of one full adder is used as the **carry in** into its left neighbour
  - The carry into the rightmost bit is set to 0.

This type of adder is called a

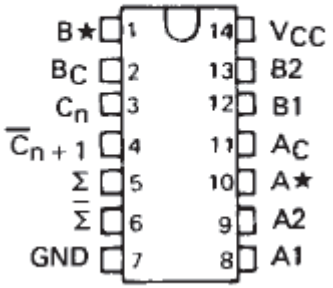
ripple carry adder

because the addition cannot complete until the carry has rippled all the way across the adders

74 80 Gated Full Adder

SN7480 . . . J OR N PACKAGE

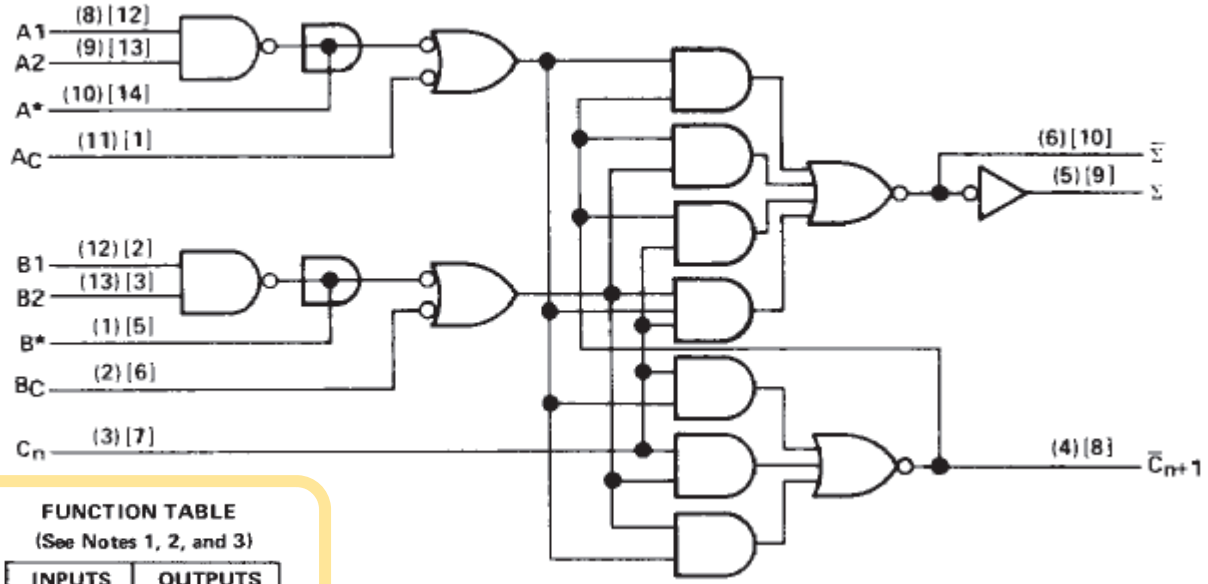
(TOP VIEW)



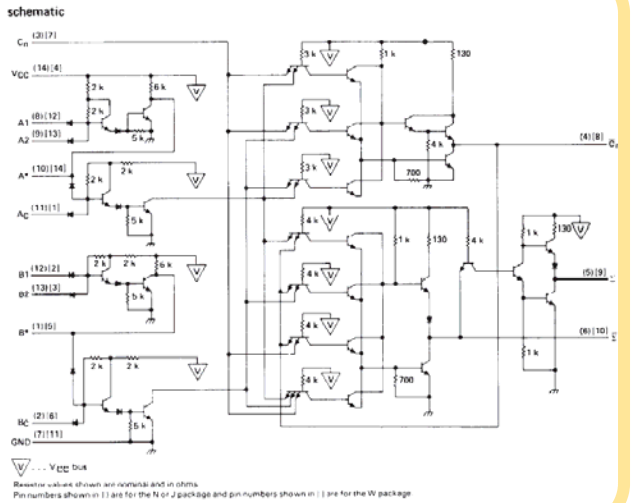
description

These single-bit, high-speed, binary full adders with gated complementary inputs, complementary sum (Σ and $\overline{\Sigma}$) outputs and inverted carry output are designed for medium-and high-speed, multiple-bit, parallel-add/serial-carry application. These circuits (see schematic) utilize diode-transistor logic (DTL) for the gated inputs, and high-speed, high-fan-out transistor-transistor logic (TTL) for the sum and carry outputs and are entirely compatible with the TTL logic families. The implementation of a single-inversion, high-speed, Darlington-connected serial-carry circuit minimizes the necessity for extensive "look-ahead" and carry-cascading circuits.

(DUAL-IN-LINE) [FLAT PACKAGE]



schematic



FUNCTION TABLE

(See Notes 1, 2, and 3)

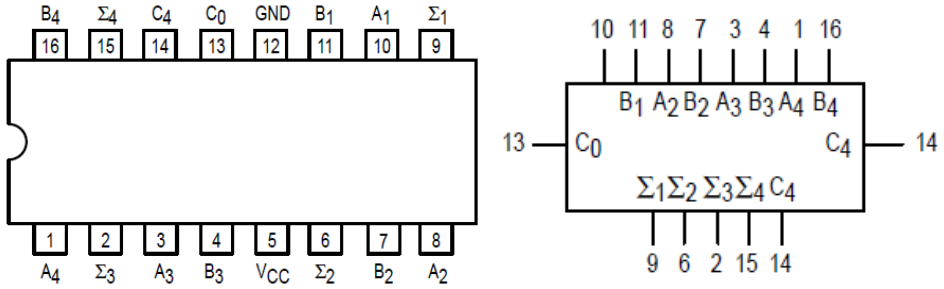
INPUTS			OUTPUTS		
C_n	B	A	$\overline{C}_{n+1}$	$\overline{\Sigma}$	Σ
L	L	L	H	H	L
L	L	H	H	L	H
L	H	L	H	L	H
L	H	H	L	H	L
H	L	L	H	L	H
H	L	H	L	H	L
H	H	L	L	H	L
H	H	H	L	L	H

H = high level, L = low level

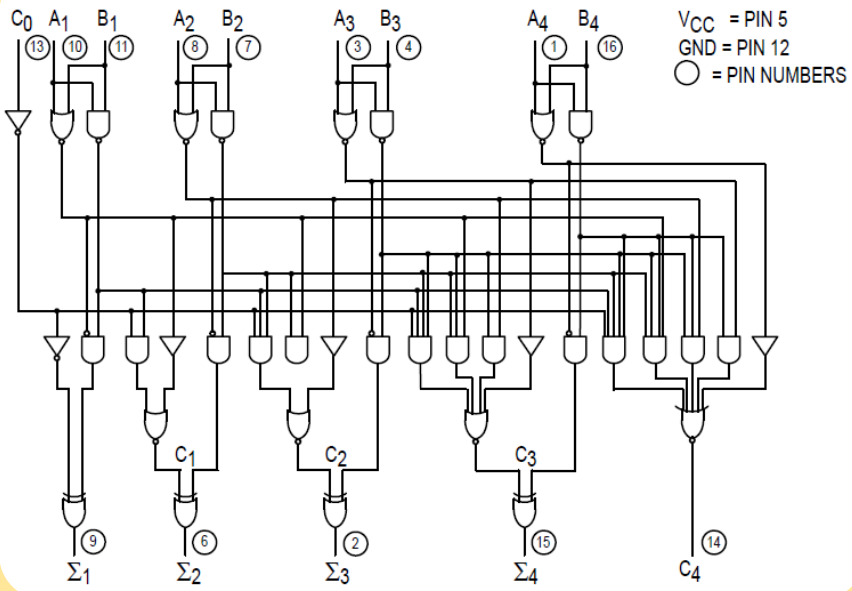
NOTES: 1. $A = \overline{A}_C + \overline{A}\star + A1 \cdot A2$, $B = \overline{B}_C + \overline{B}\star + B1 \cdot B2$.
2. When A★ is used as an input, A1 or A2 must be low. When B★ is used as an input, B1 or B2 must be low.
3. When A1 and A2 or B1 and B2 are used as inputs, A★ or B★, respectively, must be open or used to perform dot-AND logic.

74 83A Full Adder

LOGIC SYMBOL



LOGIC DIAGRAM



FUNCTIONAL TRUTH TABLE

C (n–1)	A_n	B_n	Σ_n	C_n
L	L	L	L	L
L	L	H	H	L
L	H	L	H	L
L	H	H	L	H
H	L	L	H	L
H	L	H	L	H
H	H	L	L	H
H	H	H	H	H

C_1 — C_3 are generated internally
C_0 — is an external input
C_4 — is an output generated internally

FUNCTIONAL DESCRIPTION

The LS83A adds two 4-bit binary words (A plus B) plus the incoming carry. The binary sum appears on the sum outputs ($\Sigma_1 - \Sigma_4$) and outgoing carry (C_4) outputs.

$C_0 + (A_1+B_1)+2(A_2+B_2)+4(A_3+B_3)+8(A_4+B_4) = \Sigma_1+2\Sigma_2+4\Sigma_3+8\Sigma_4+16C_4$

Where: (+) = plus

Due to the symmetry of the binary add function the LS83A can be used with either all inputs and outputs active HIGH (positive logic) or with all inputs and outputs active LOW (negative logic). Note that with active HIGH Inputs, Carry Input can not be left open, but must be held LOW when no carry in is intended.

Example

	C_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4	Σ_1	Σ_2	Σ_3	Σ_4	C_4	
Logic Levels	L	L	H	L	H	H	L	L	H	H	H	L	L	H	
Active HIGH	0	0	1	0	1	1	0	0	1	1	1	0	0	1	(10+9 = 19)
Active LOW	1	1	0	1	0	0	1	1	0	0	0	1	1	0	(carry+5+6 = 12)

4.1 ALU – Arithmetic Logic Unit

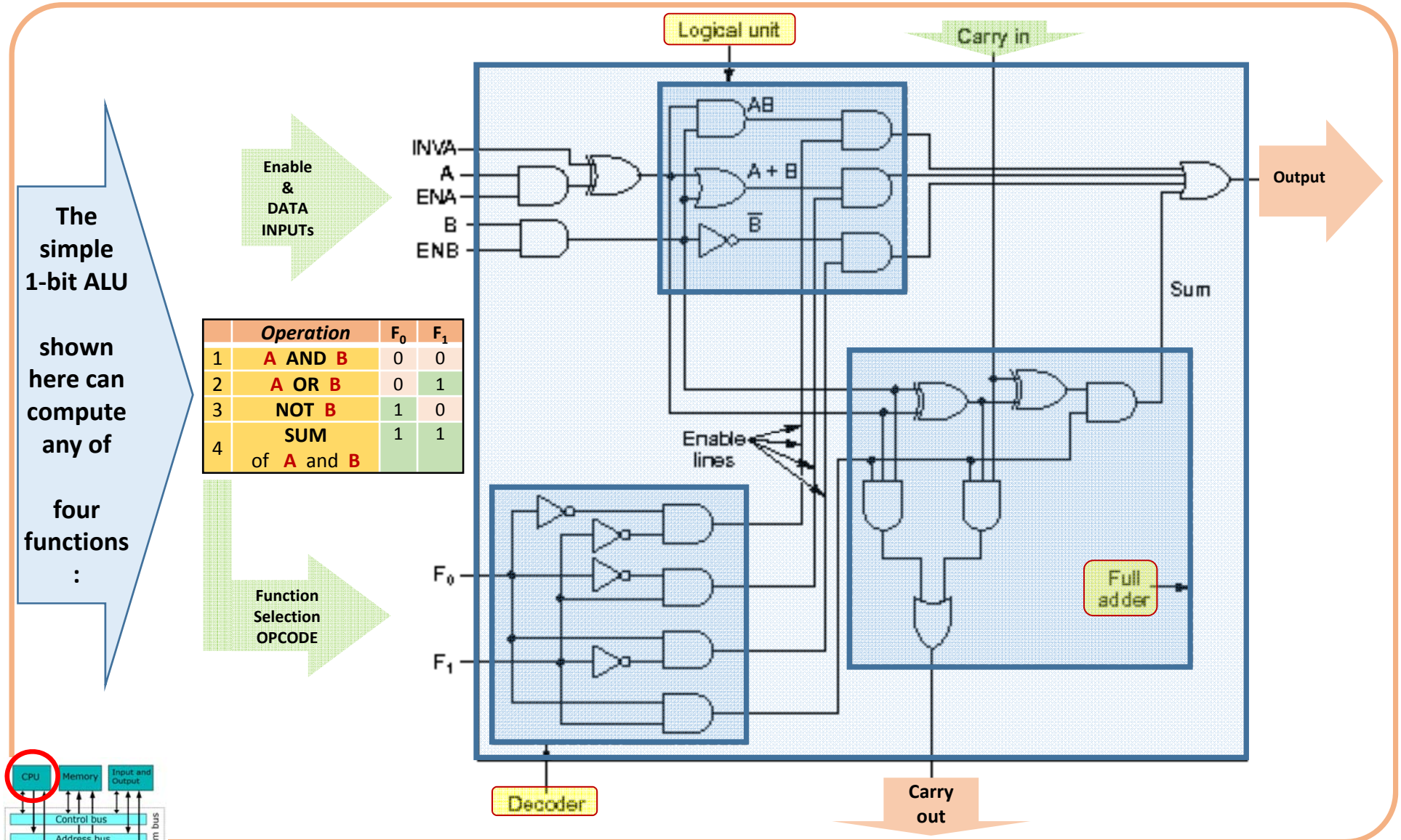
The heart of every computer is a processor, or ***CPU*** (Central Processing Unit)

CPUs consist of an ALU and a Control Unit

The ***ALU*** is a circuit, used in the **CPU** for performing **A**rithmetic and (Boolean) **L**ogical operations

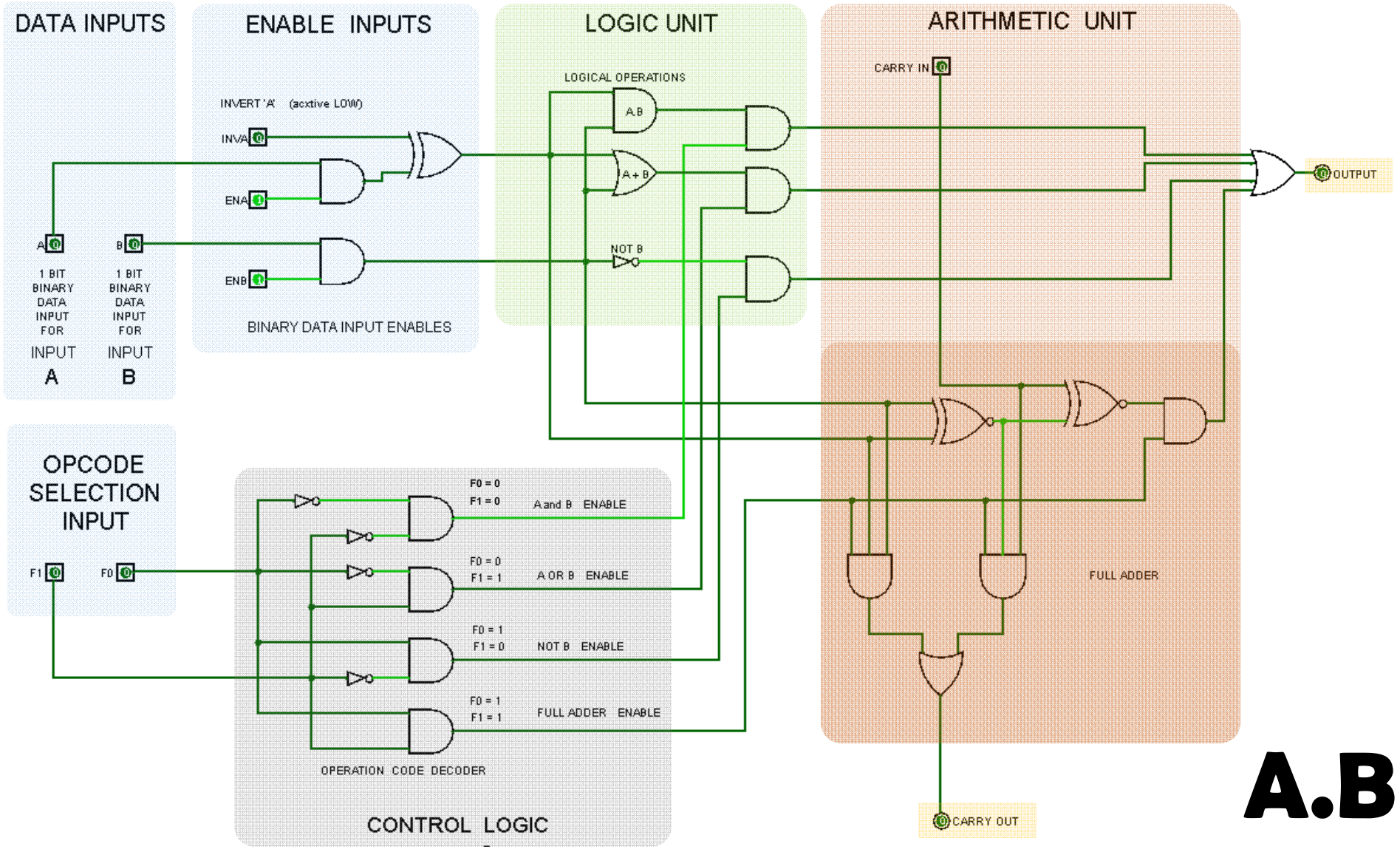
The simple 1-bit ALU shown here can compute any of four functions :

	Operation	F_0	F_1
1	A AND B	0	0
2	A OR B	0	1
3	NOT B	1	0
4	SUM of A and B	1	1

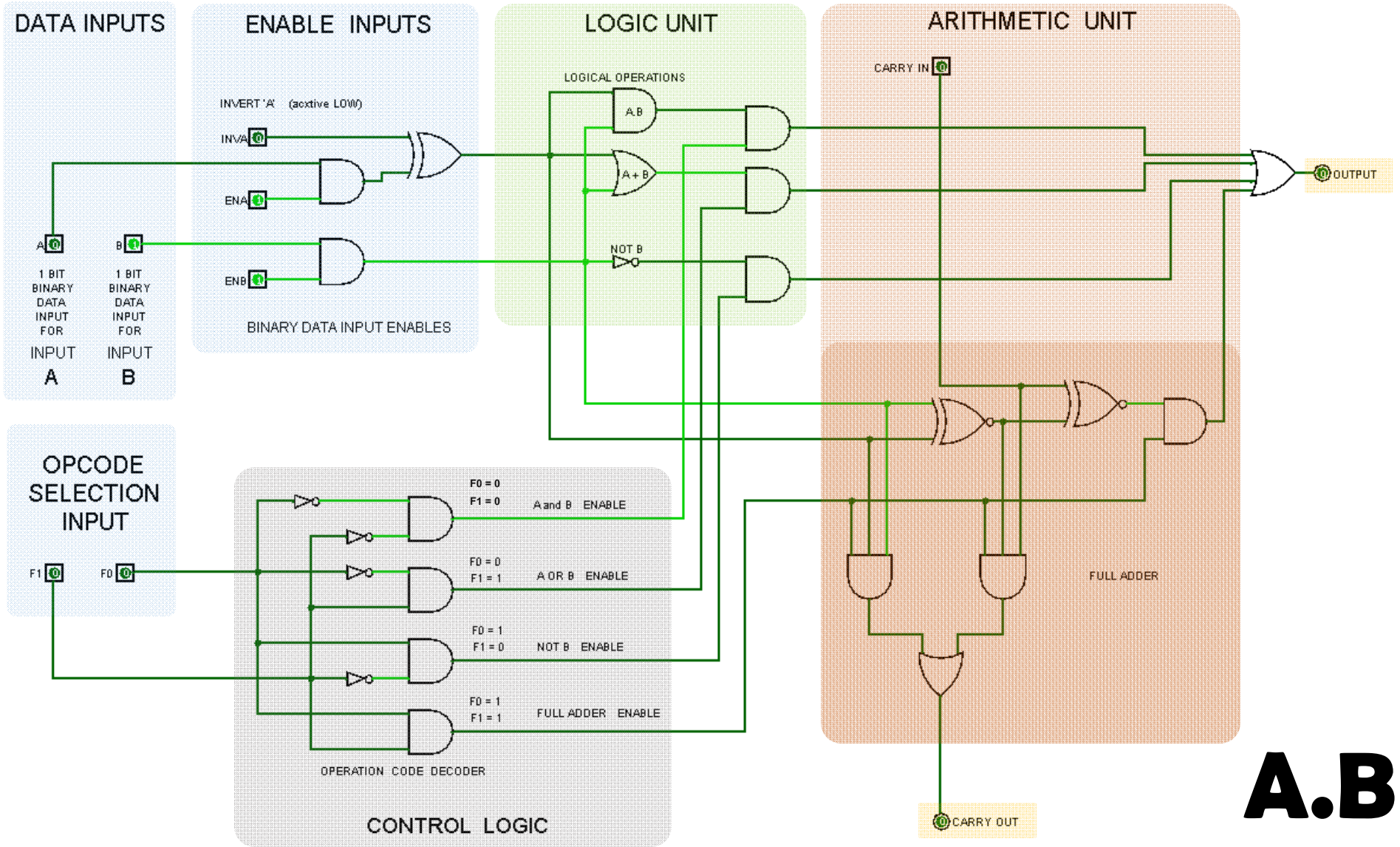


Now let's see how each of the **function codes** affect the operation of the **one-bit ALU** for all binary input values

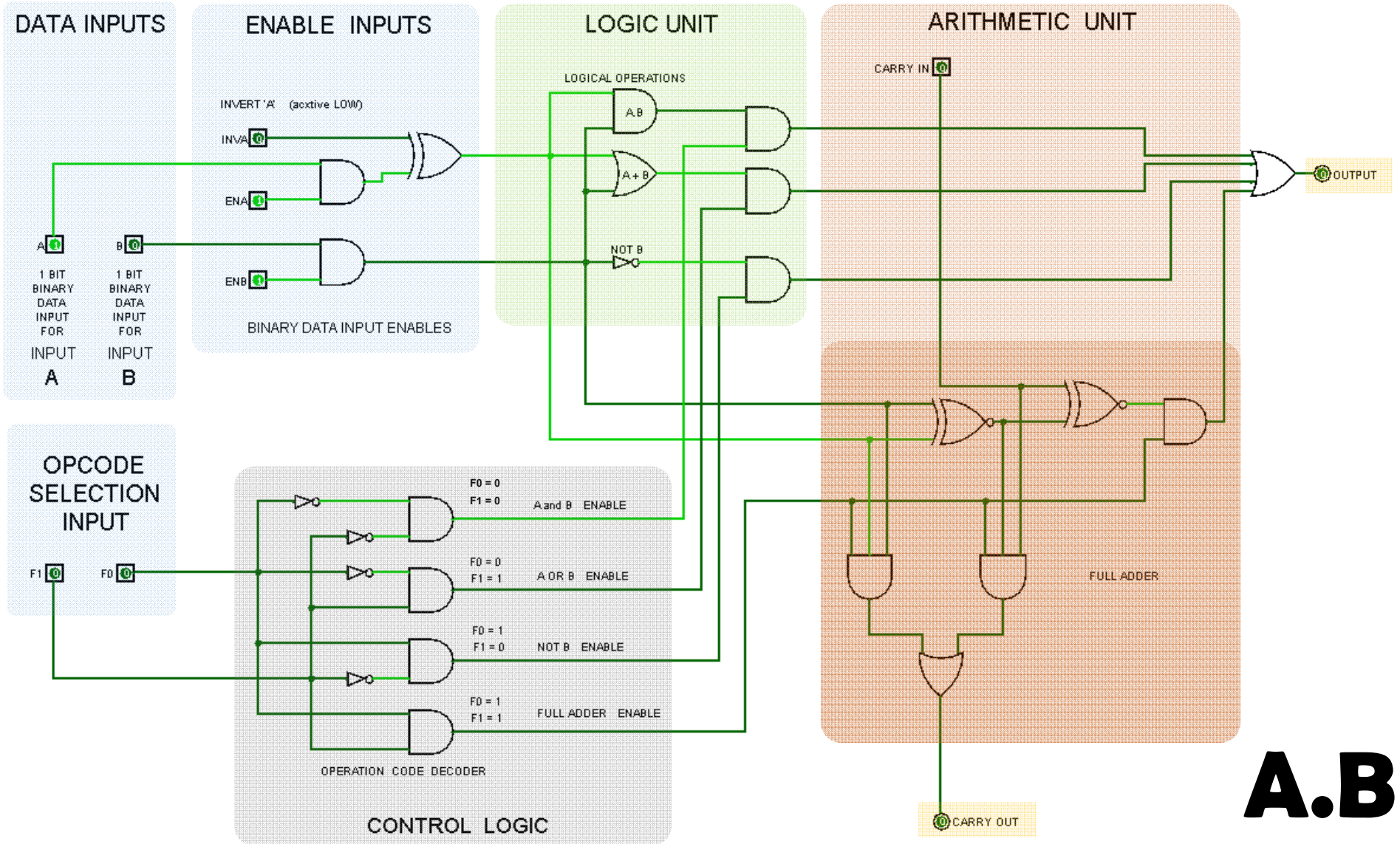
1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	0	0	0	1	1	0	0	0	0



1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	1	0	0	1	1	0	0	0	0

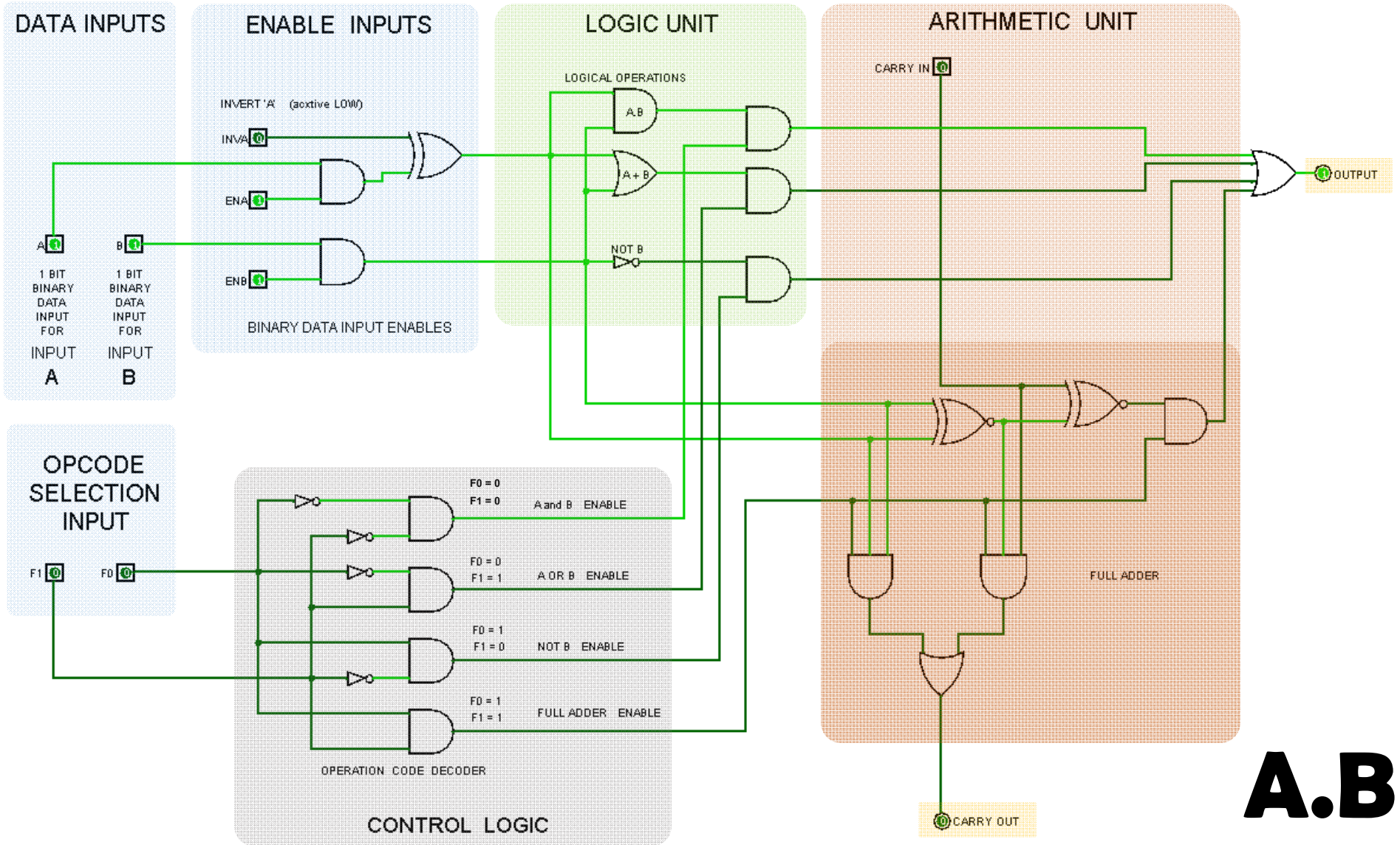


1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	0	0	0	1	1	0	0	0	0

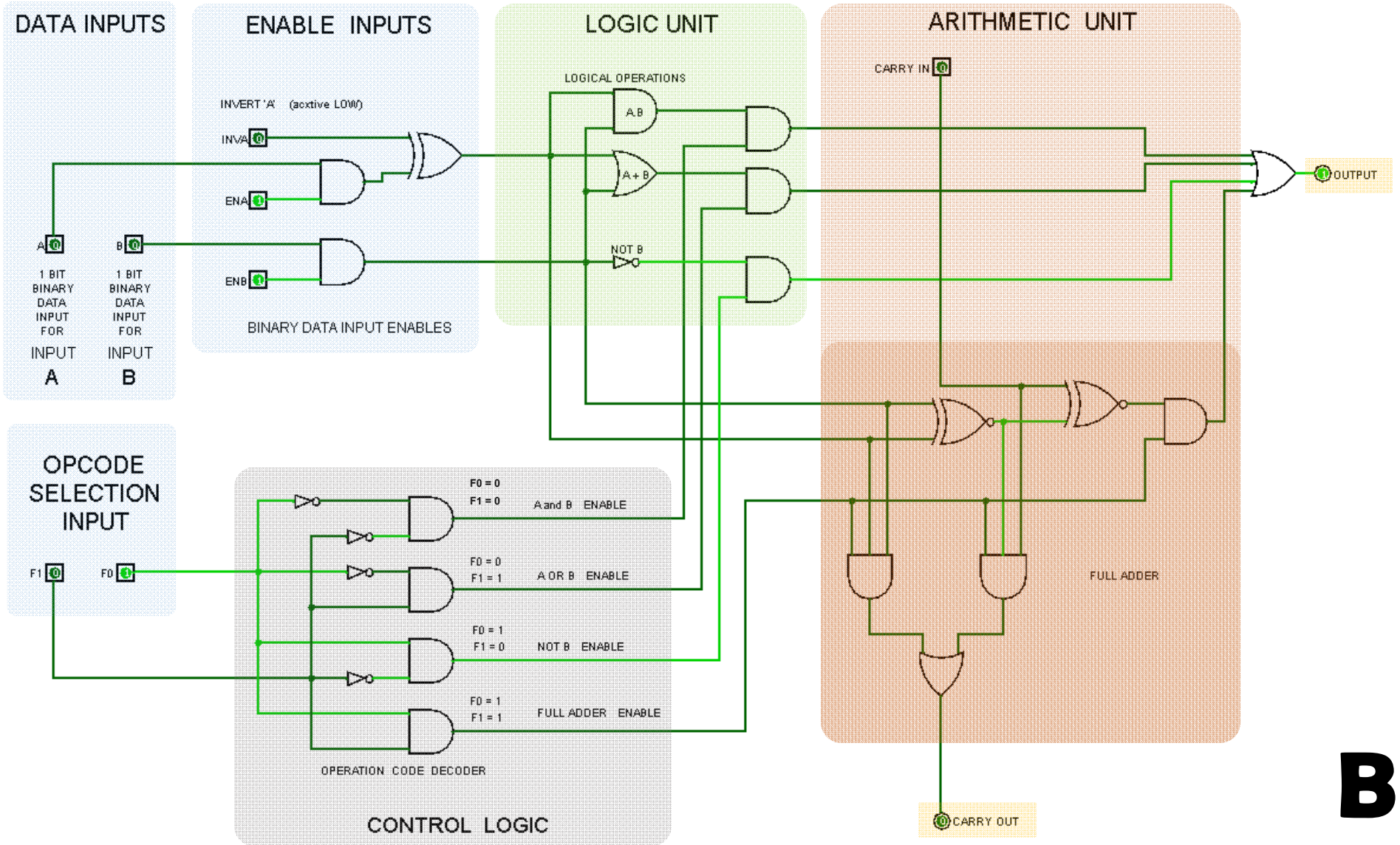


A.B

1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	1	0	0	1	1	0	0	0	1

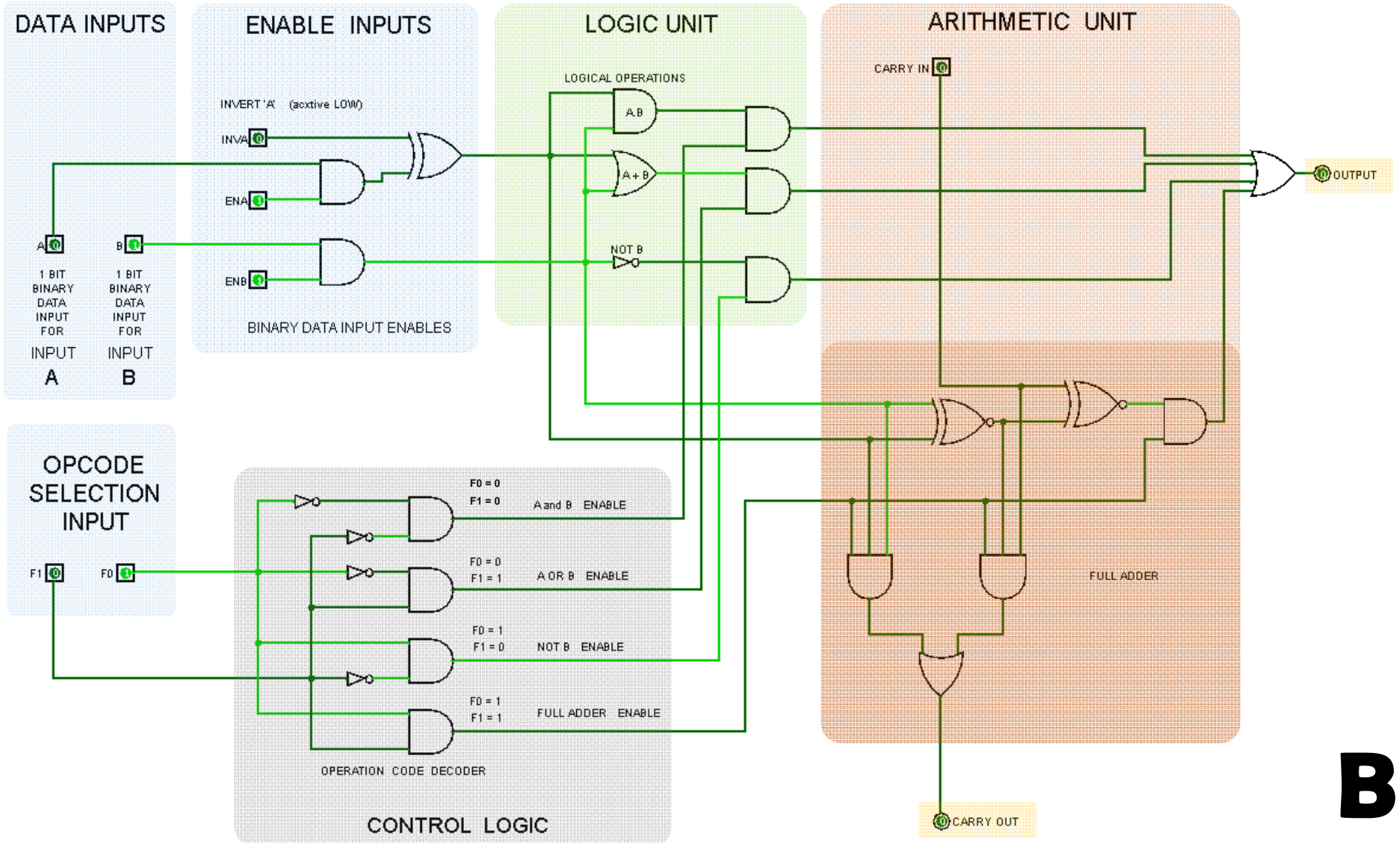


1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	0	0	1	1	1	0	0	0	1



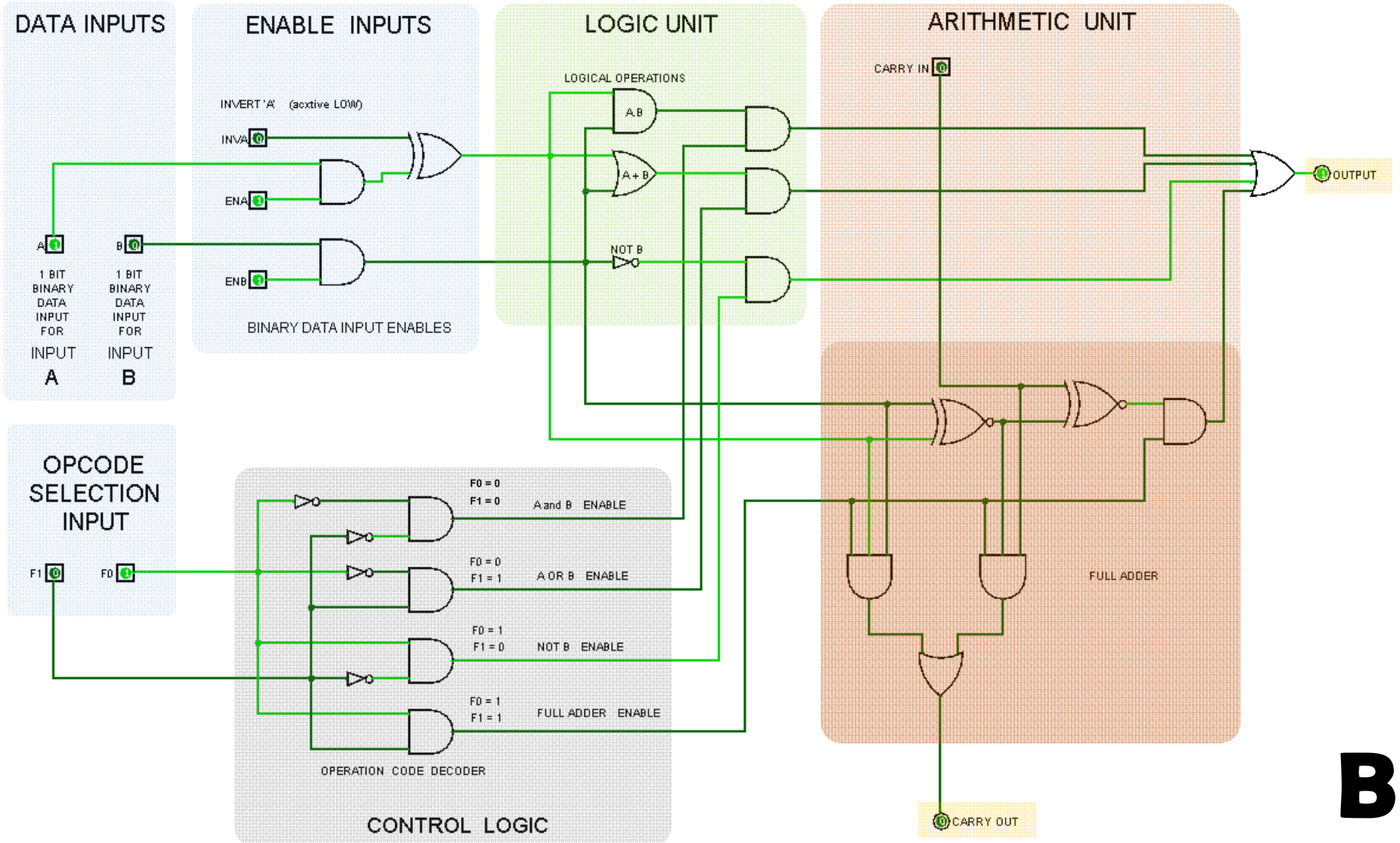
B

1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	1	0	1	1	1	0	0	0	0



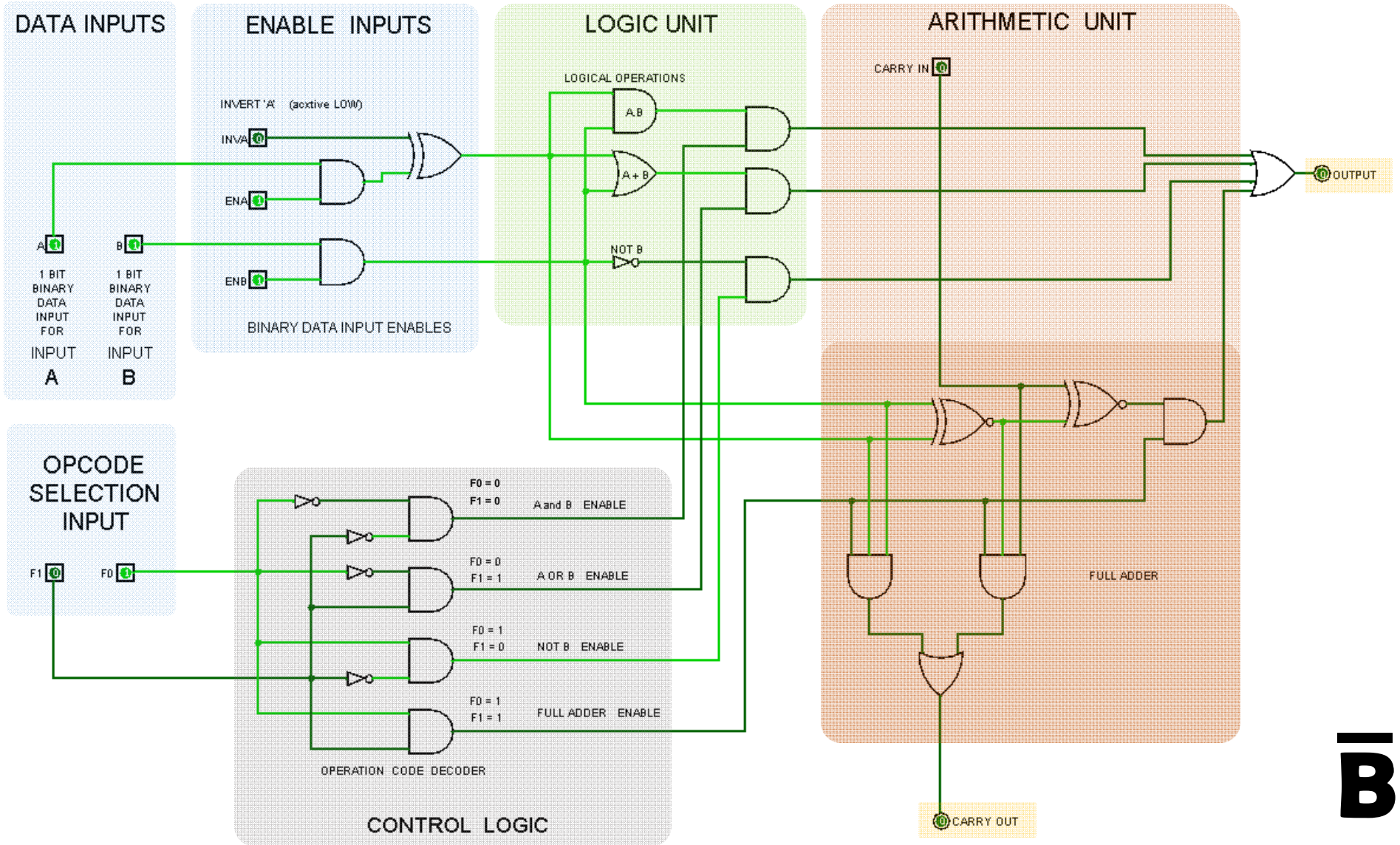
B

1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	0	0	1	1	1	0	0	0	1



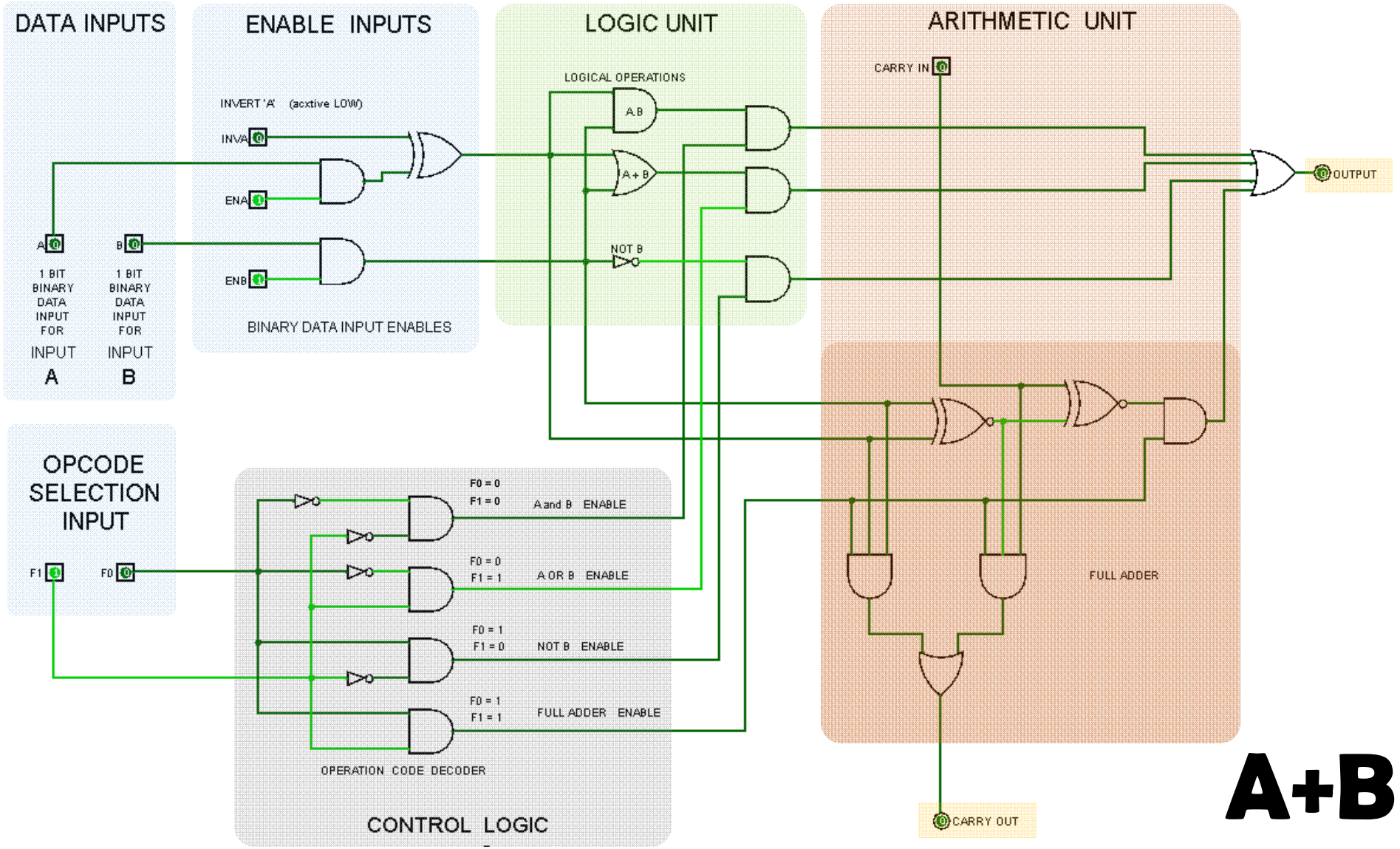
B

1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	1	0	1	1	1	0	0	0	0

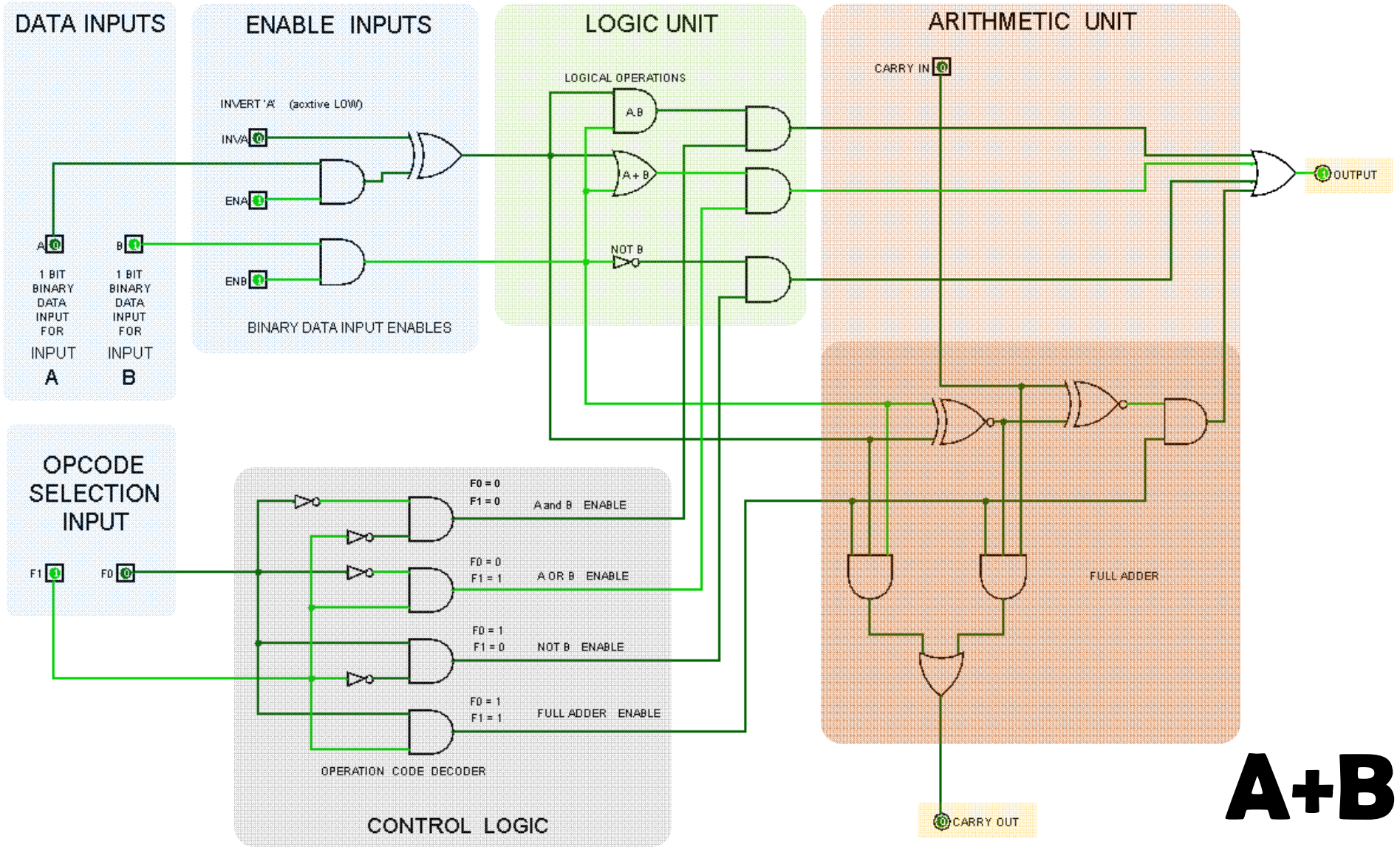


$\overline{B}$

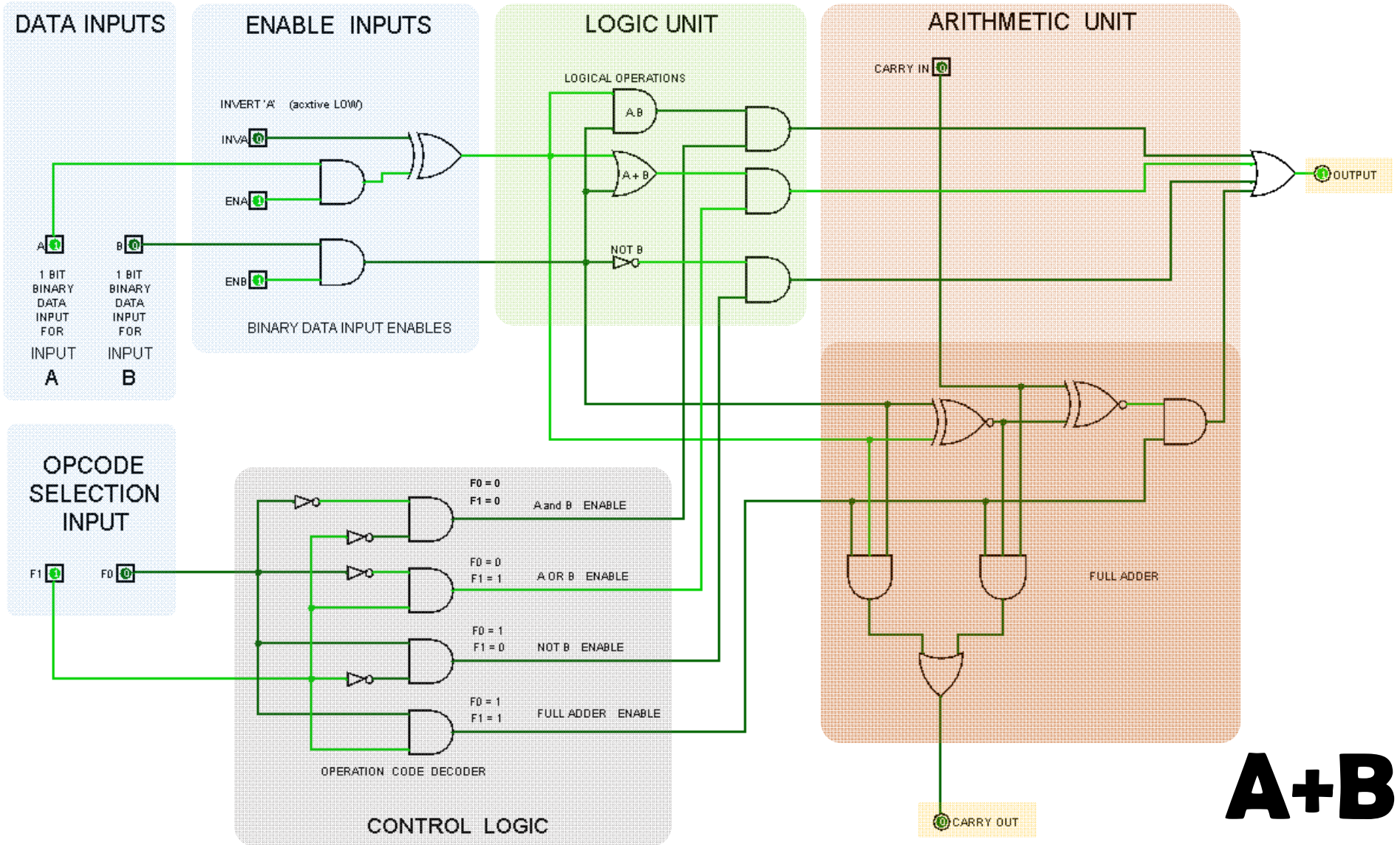
1 bit ALU	INPUTS						OUTPUTS		
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	0	1	0	1	1	0	0	0	0



1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	1	1	0	1	1	0	0	0	1

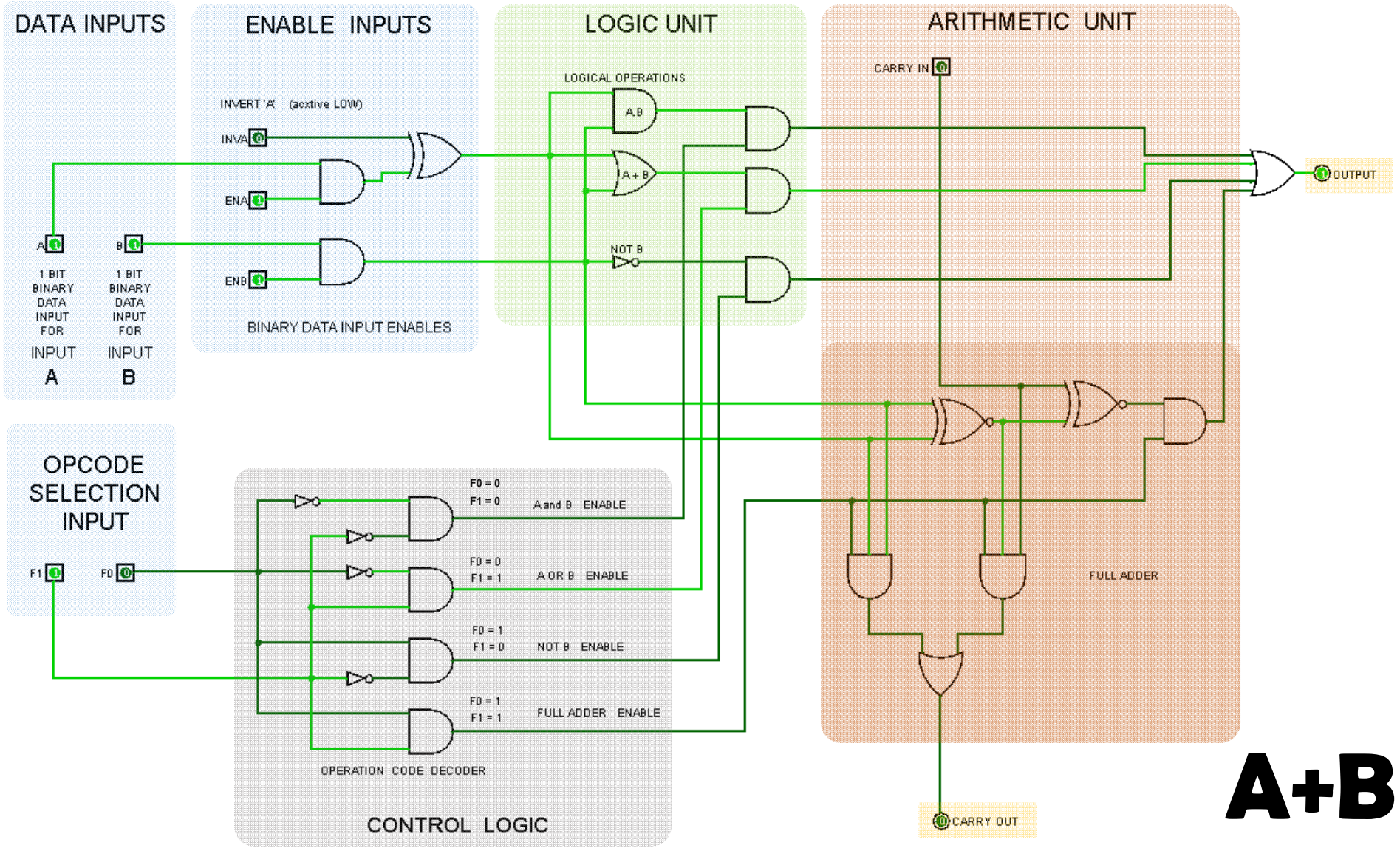


1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	0	1	0	1	1	0	0	0	1

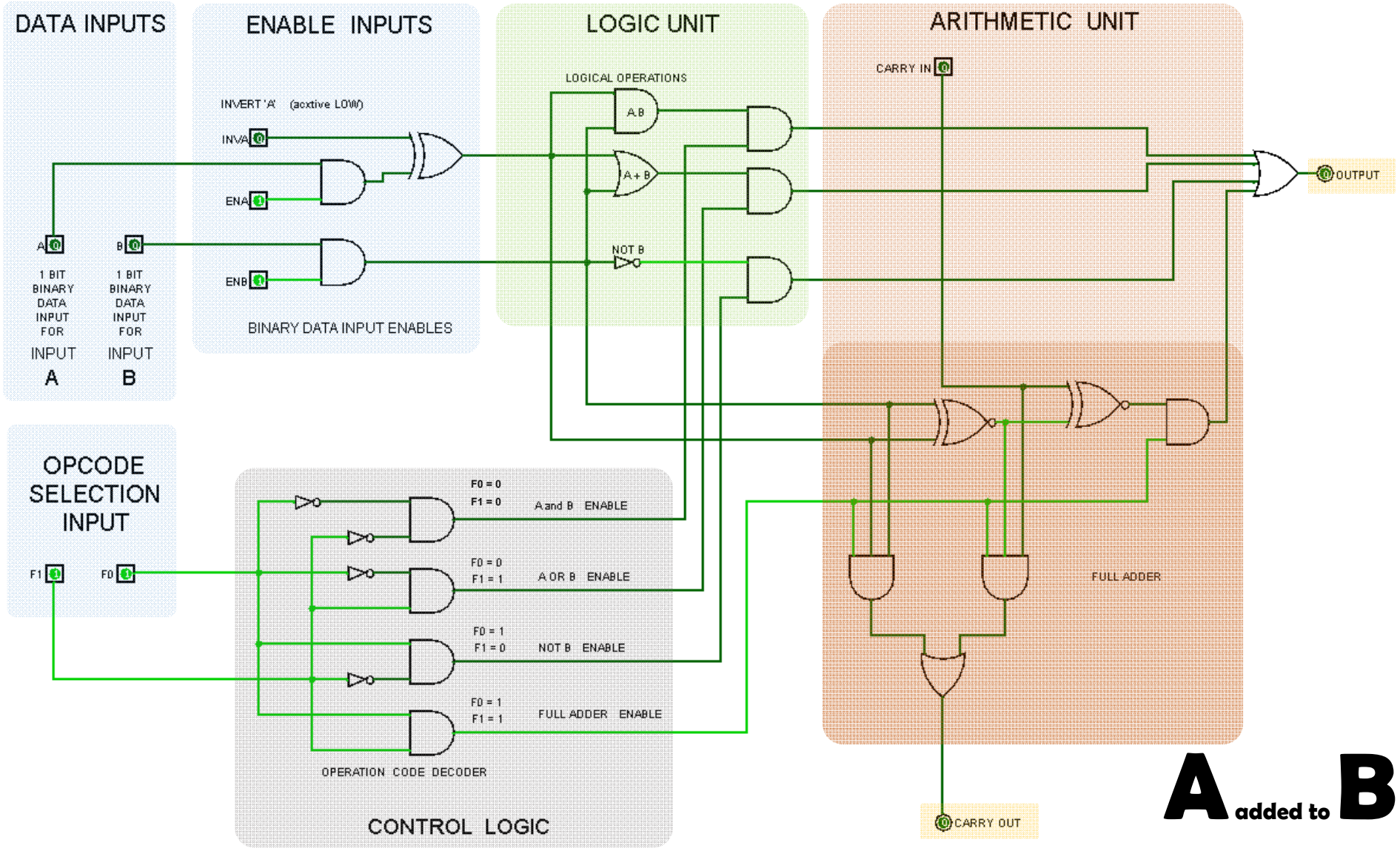


A+B

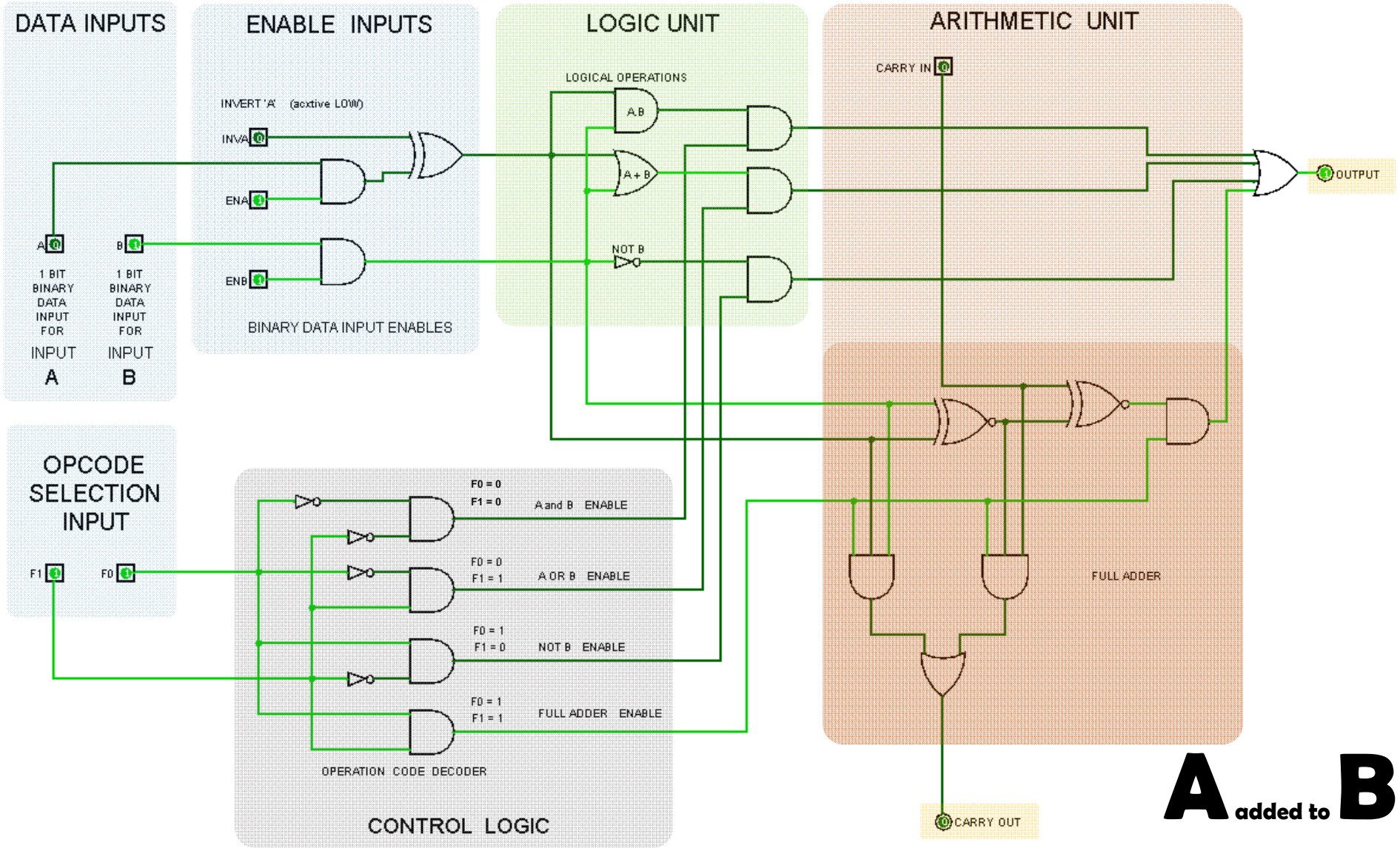
1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	1	1	0	1	1	0	0	0	1



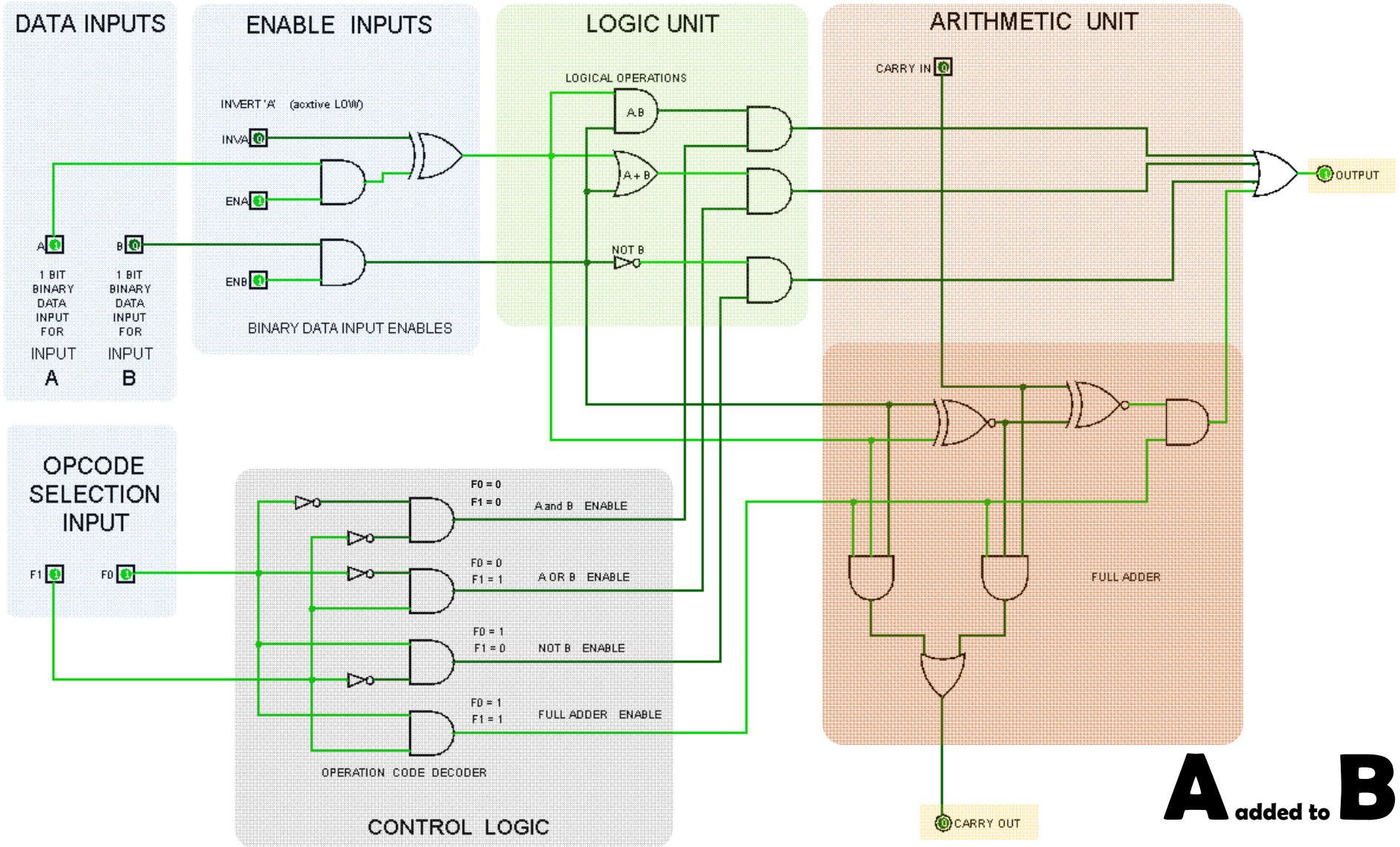
1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	0	1	1	1	1	0	0	0	0



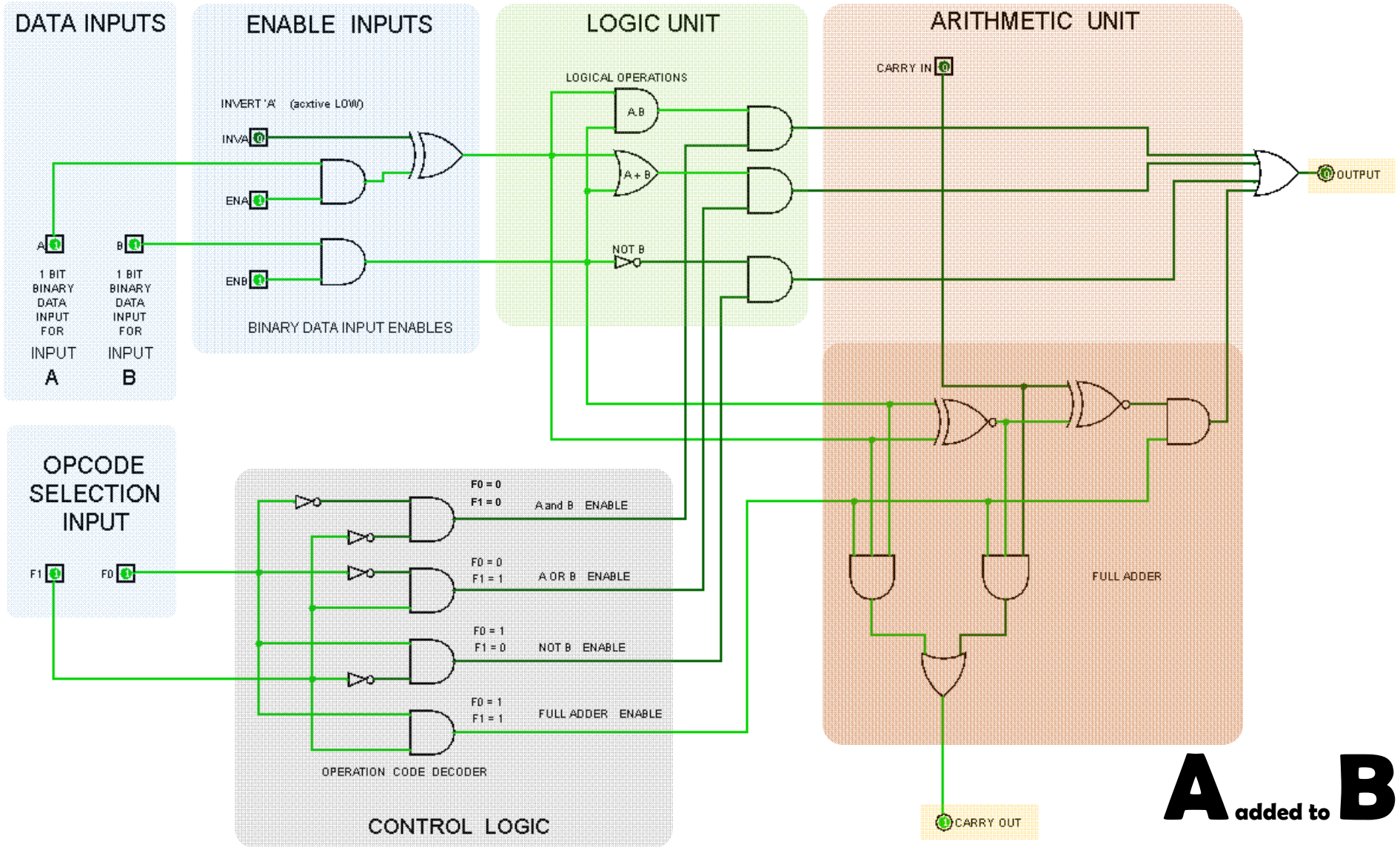
1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
0	1	1	1	1	1	0	0	0	1



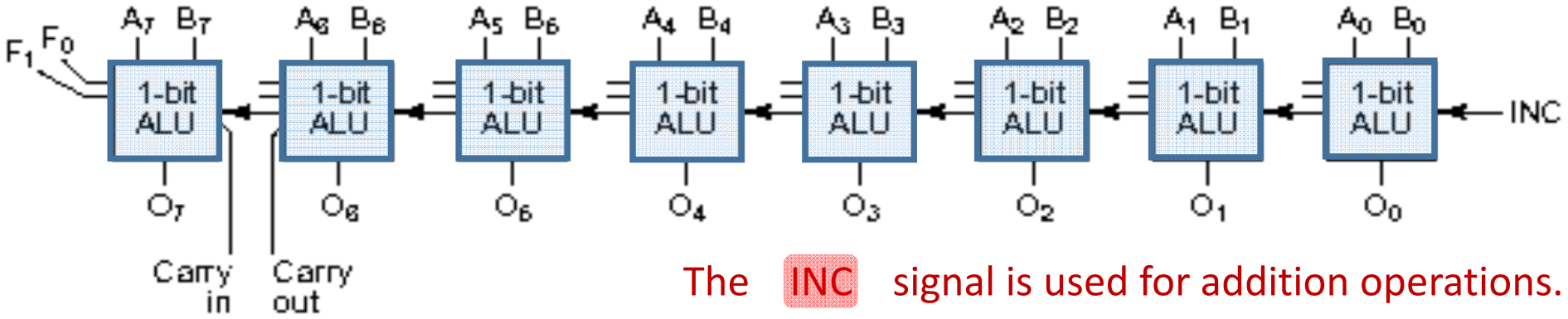
1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	0	1	1	1	1	0	0	0	1



1 bit ALU	INPUTS						OUTPUTS	
A	B	F_1	F_0	ENA	ENB	INVA	CARRY IN	CARRY OUTPUT	OUTPUT
1	1	1	1	1	1	0	0	1	0



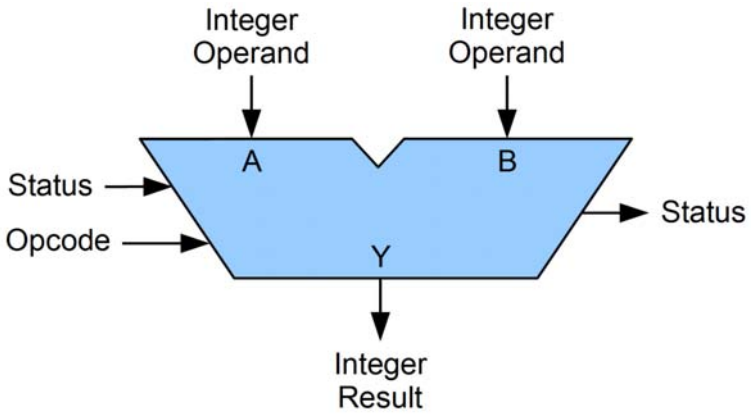
1-bit ALUs allow computer designers to build an ALU of any desired size

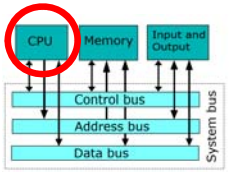


The INC signal is used for addition operations.

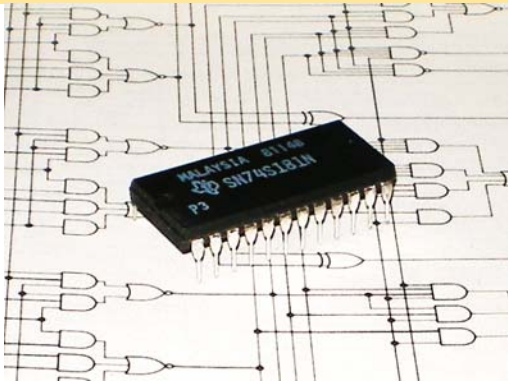
When present it increments (adds 1) to the result,
allowing the computation of sums such as ***A+1***

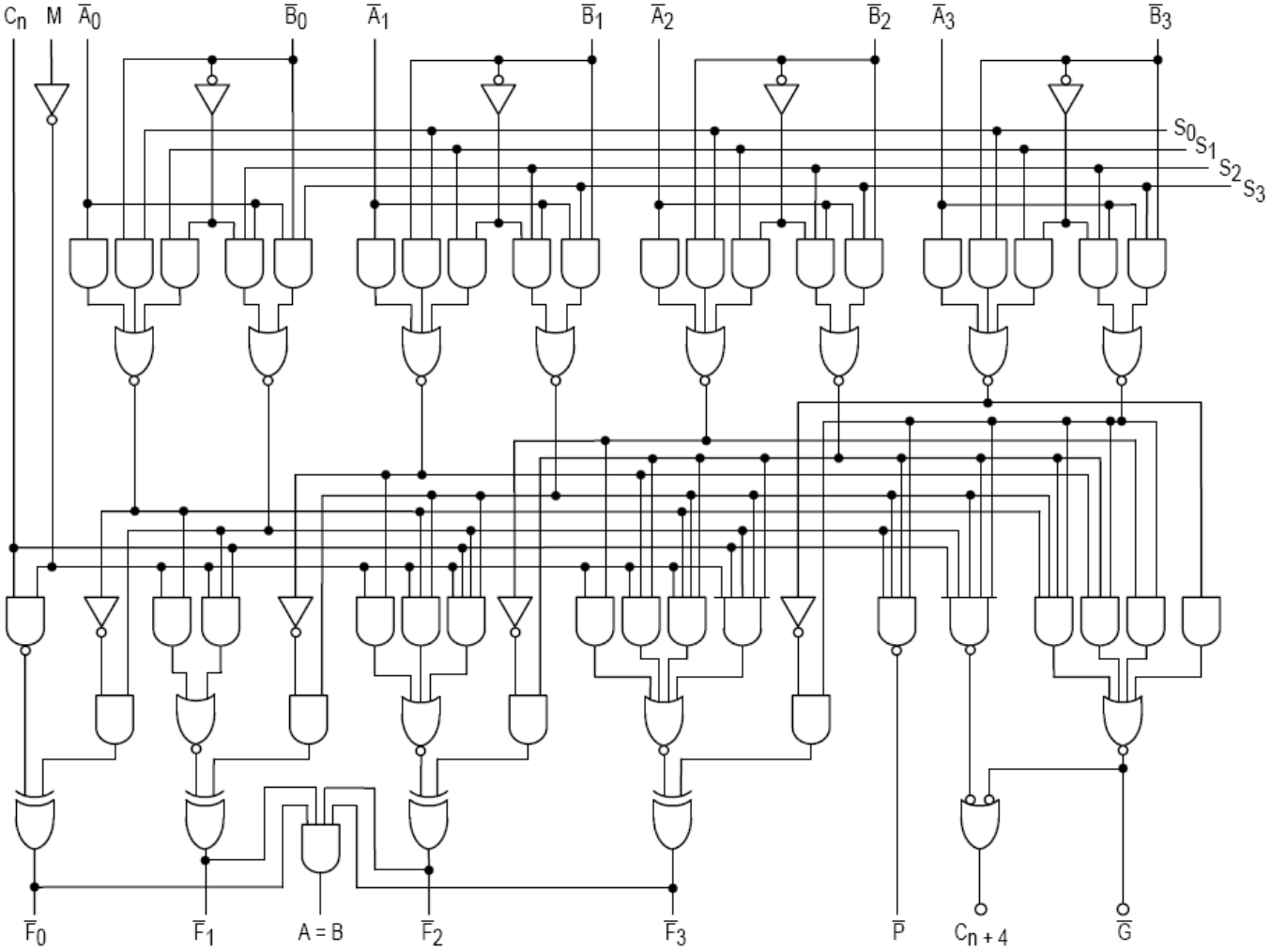
Common symbol used for an ALU





74181 bit slice Arithmetic & Logic Unit





4.2 Memory

Combinational circuits

- *outputs dependent on* ***present input values***

Sequential circuits

- *outputs depend on* ***present input values***, and on ***past input values***

4.2.1 Flip-Flops

- Sequential circuit that forms the basic building block of memory
  - stores a single bit

Flip : output changes from

Flop : output changes from

4.2.2 D-type Flip-Flops

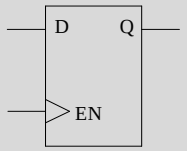
Two inputs:

D is the ***data*** input value (the data bit that the flip-flop is to store)

EN is the ***enable*** control input.

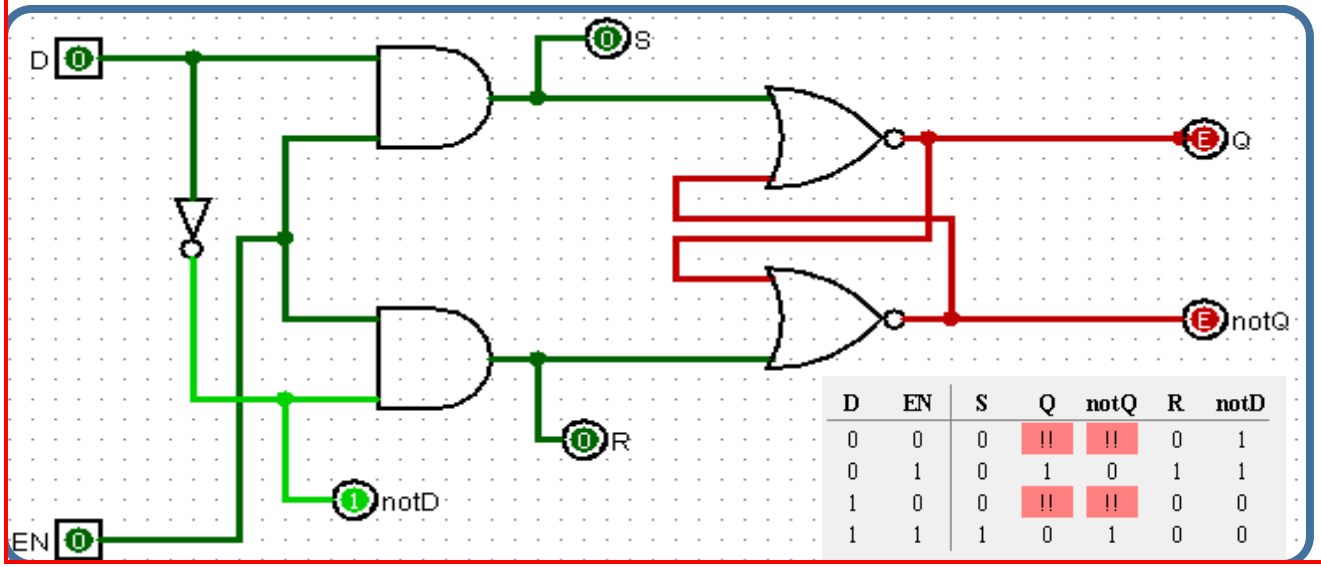
Standard symbol

74 79



- Output ***Q*** only changes on the transition of ***EN*** signal from one specified logic level to the other logic level.

- without a transition the output will not change even if the input ***D*** does.
- This is known as "*edge triggering*"
  - ***positive*** (0 to 1) vs ***negative*** (1 to 0) edge triggering.



D	EN	S	Q	notQ	R	notD
0	0	0	!!	!!	0	1
0	1	0	1	0	1	1
1	0	0	!!	!!	0	0
1	1	1	0	1	0	0

Operation of the flip-flop can be clarified by a simplified truth table.

EN	D	Q
0	0	Q
0	1	Q
1	0	0
1	1	1

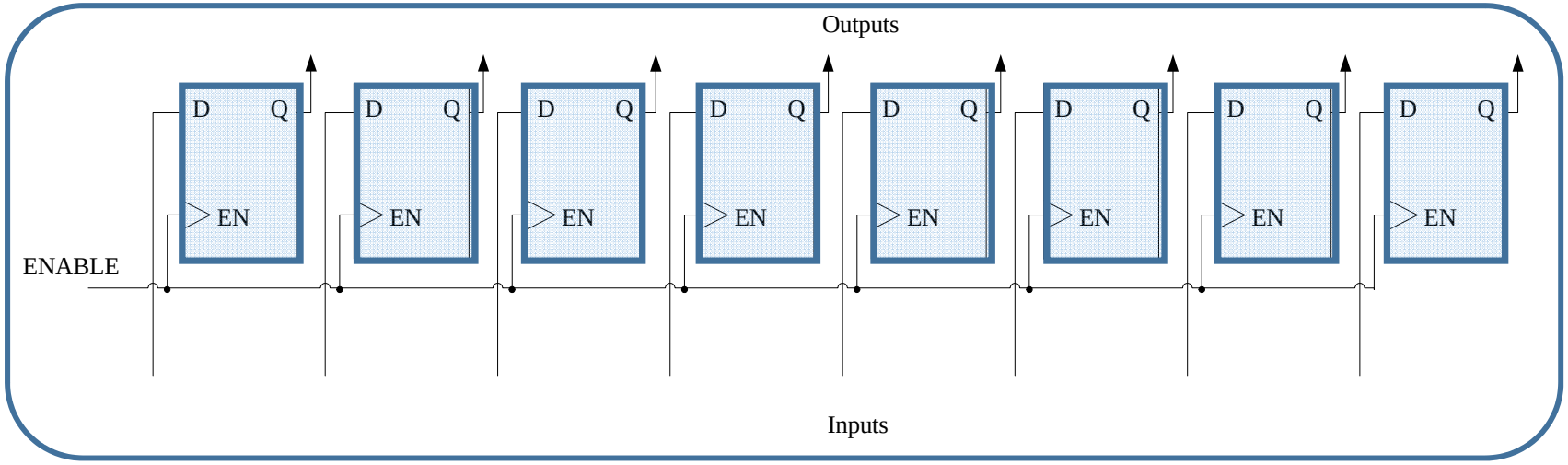
Q output becomes the value of ***D*** after the activating transition of the ***EN*** signal.

4.2.3 Registers

CPU also contains ***registers***

- small, high-speed memory used to store temporary results and control information
- each register holds one value

- To store eight bits, eight flip-flops must be combined



EN = 0

- flip-flops do not record the data being input to them, ***Q*** outputs remain unchanged

EN = 1

- flip-flops record the data on their ***D*** inputs and output it to their ***Q*** outputs

EN returns to 0

- ***Q*** outputs do not change and the data remains frozen in the flip-flops

References

- http://ece-research.unm.edu/pollard/classes/338/lademo/LookAheadDemo.htm