Computer Design Basics

Datapath and Arithmetic/Logic Unit (ALU)

Gloria Beraldo (gloria.beraldo@unipd.it)
Department of Information Engineering, University of Padova

Topics:

- Hardware/Software programming
- A simple Computer Architecture
- The Arithmetic/Logic Unit (ALU)

Book Reference:

- Chapter 8

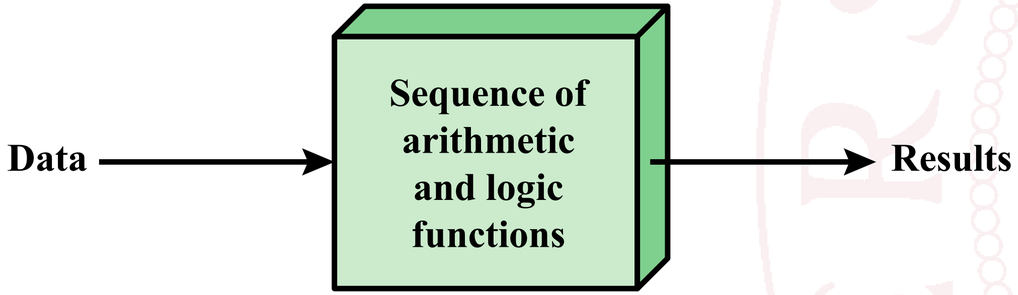
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Hardware and software programming

How to implement a program?

Hardware programming (hardwired programming)

- There is a set of **basic logic components** (and, or, multiplexer, ROM,…)
- A **program is a circuit** composed of these elements
- Input data are processed by the circuit to **provide the results in output**



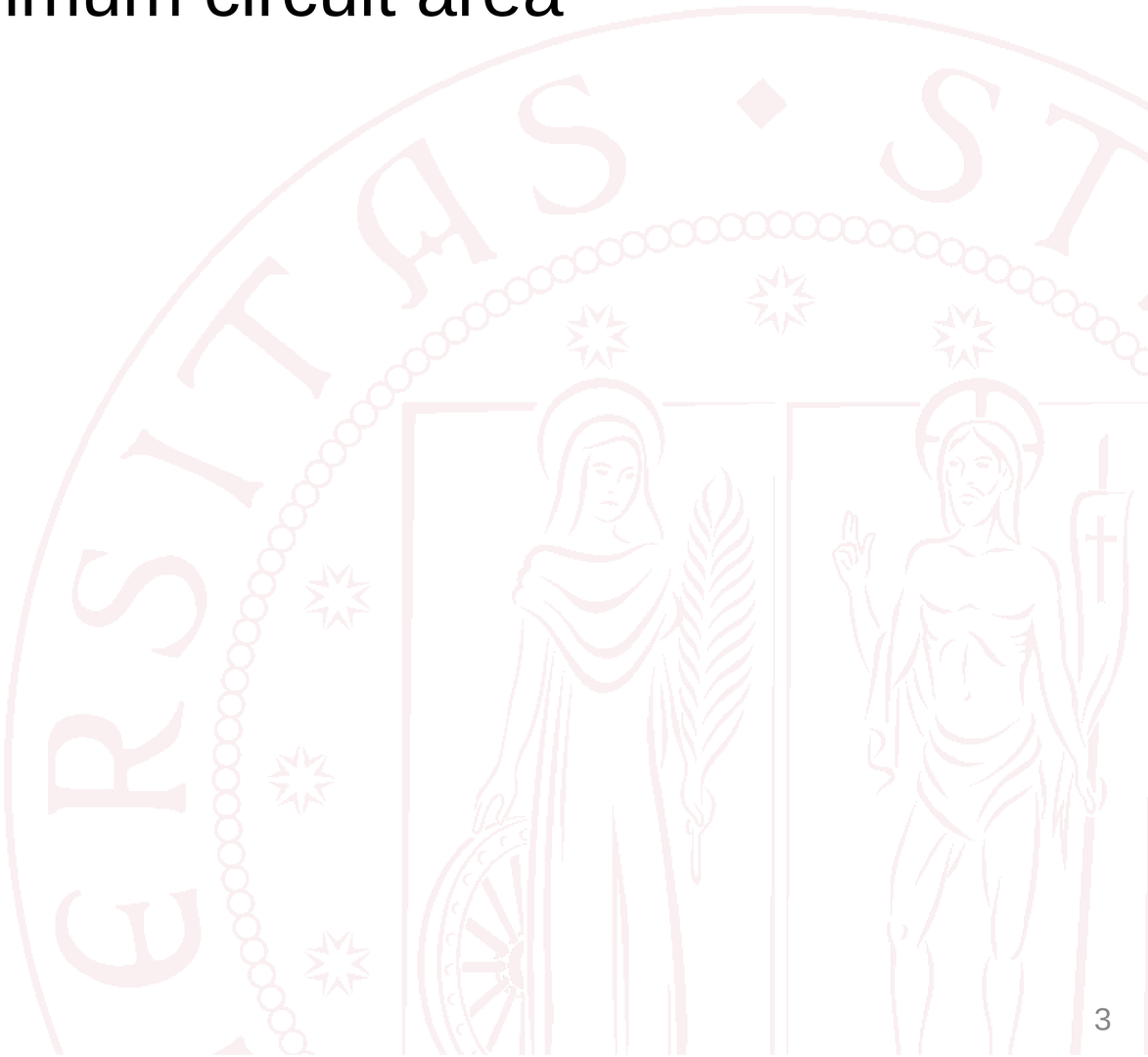
Hardware programming

PROS

- Allow to implement a program with minimum circuit area
- Fast execution
- Energy saving

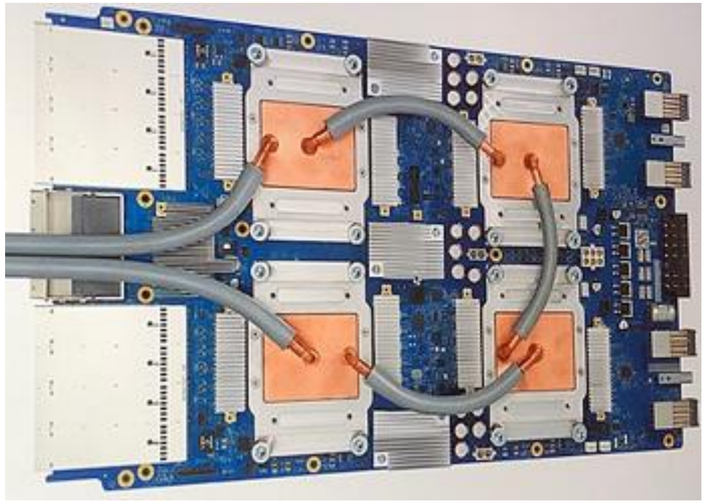
CONS

- The program **cannot** be modified
- High cost implementation
- Specific approach and not general



Hardware programming: Examples

- This approach is commonly used to accelerate the execution of machine learning algorithms



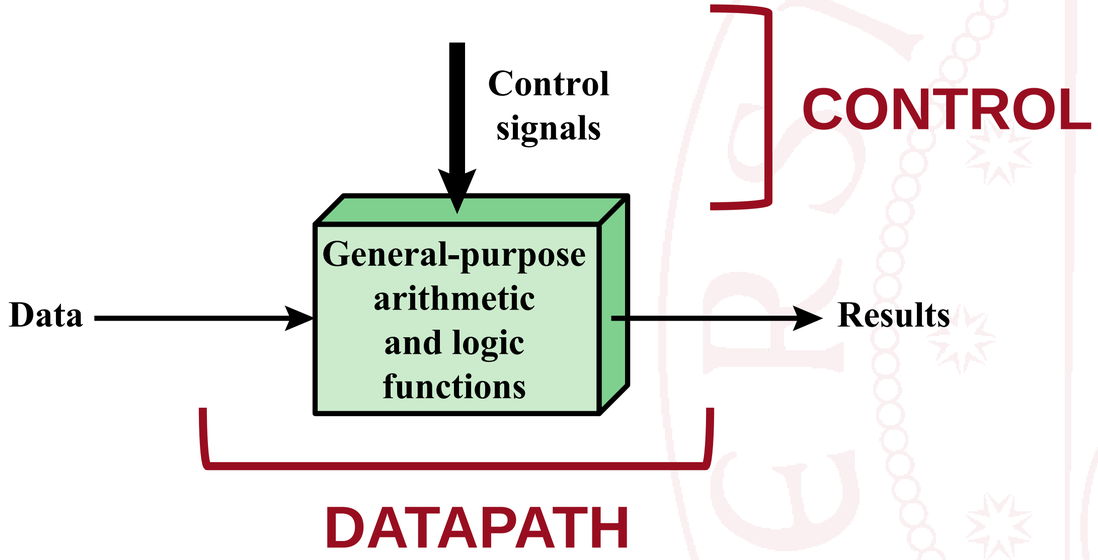
Google Tensor Processing Unit

Nvidia Quadro Volta

Software programming

Software programming:

- A set of logic and arithmetic functions is implemented in hardware
- **Control signals** specify which functions to apply on input data and in which order
- It is possible to implement new functions by changing the control signals



Software programming

PRONS

- The program **can** be changed
- Reduced implementation cost
- Generability

CONS

- Less computing efficiency
- More extensive and more expensive circuits
- Higher energy consumption

Comparison

Hardware programming

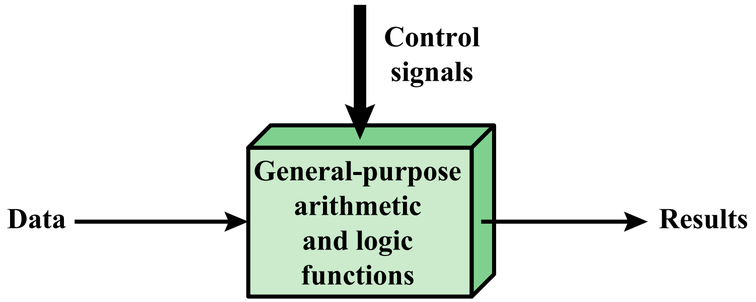
Software programming

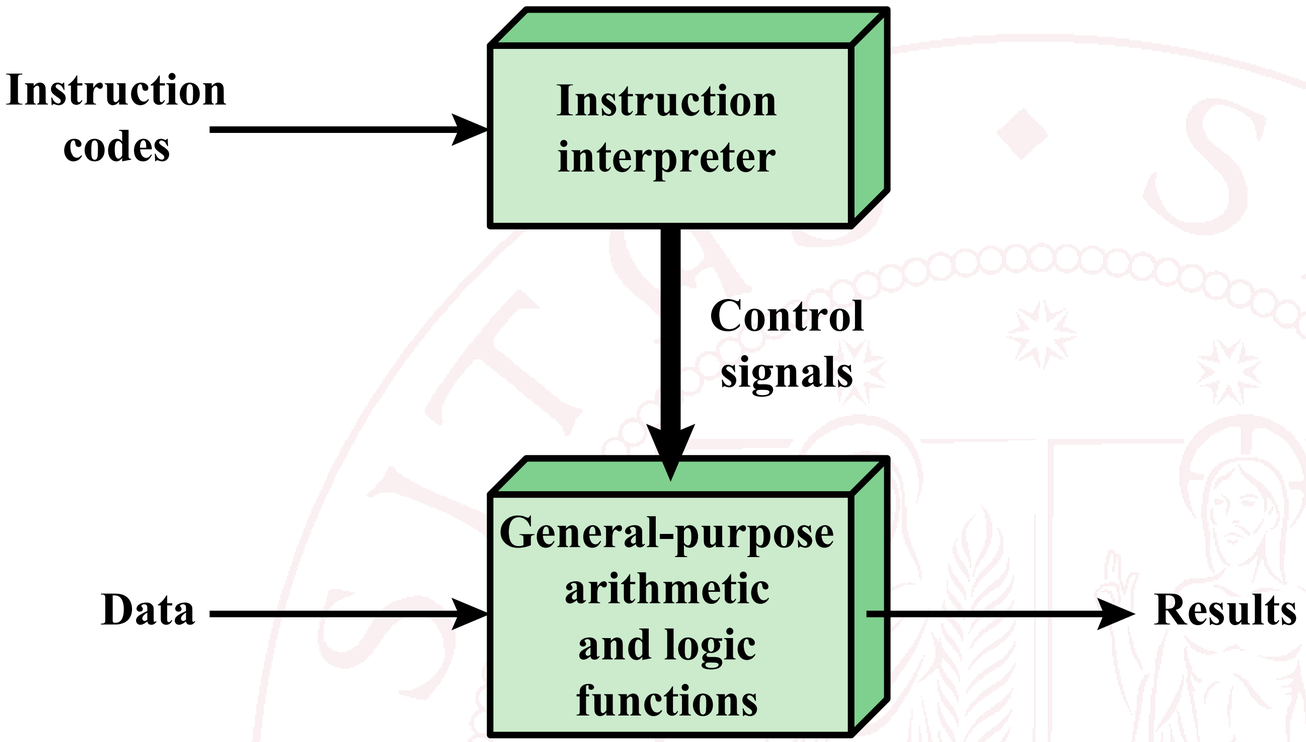
Computer Architecture

How to define the control signals?

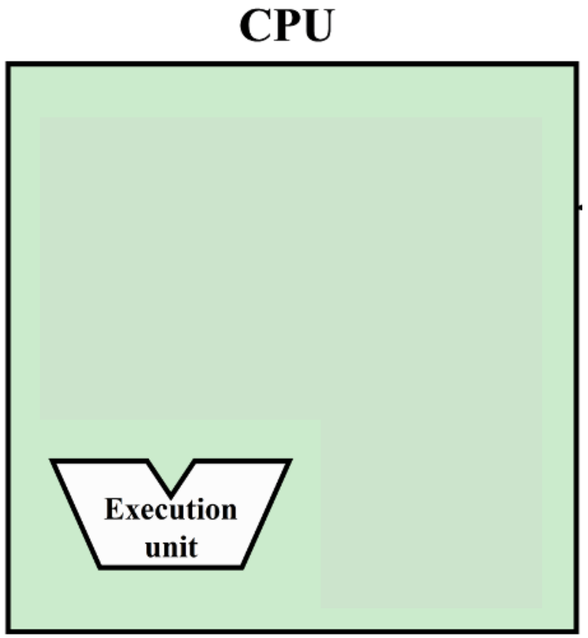
- A unique **code** is assigned to any sequence of control signals
- An hardware component is included to translate a specific code to control signals
- Each code refers to an **instruction** that uniquely determines the control signals to activate
- A program is a sequence of instructions/codes (**software**)

A system for software programming





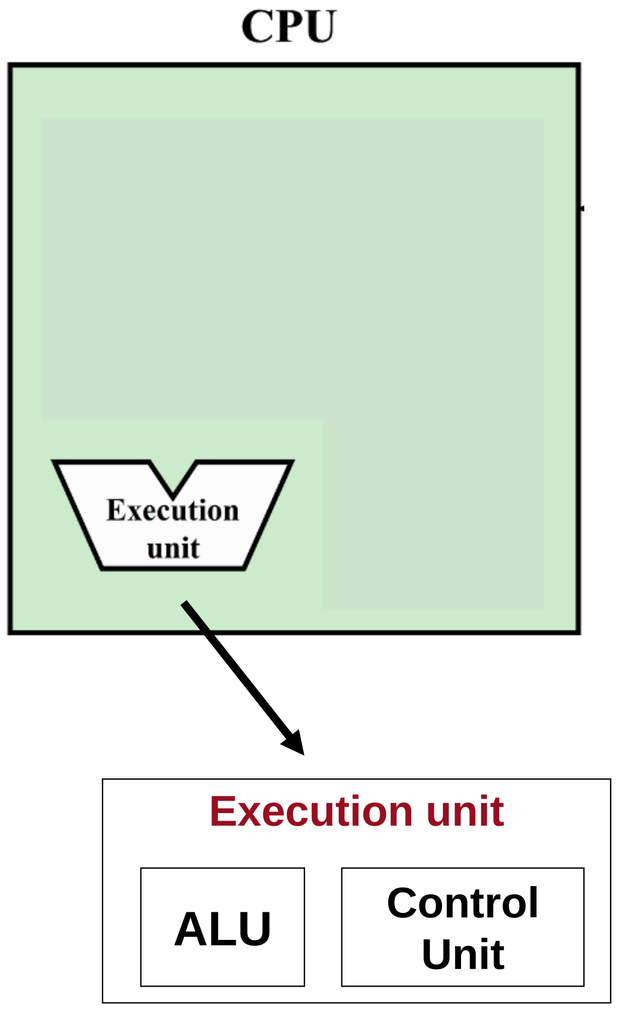
A system for software programming: CPU



A module is needed to interpret and execute the instructions.

Such a module is called
Central Processing Unit (CPU)

A system for software programming: CPU

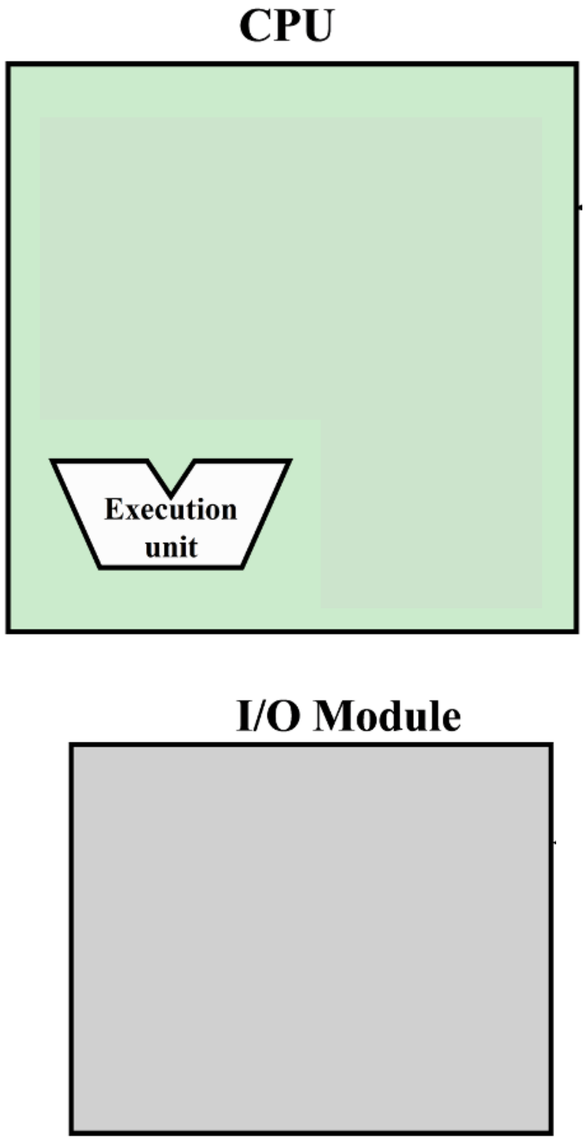


Execution unit is formed by:

Arithmetic and Logic Unit (ALU)

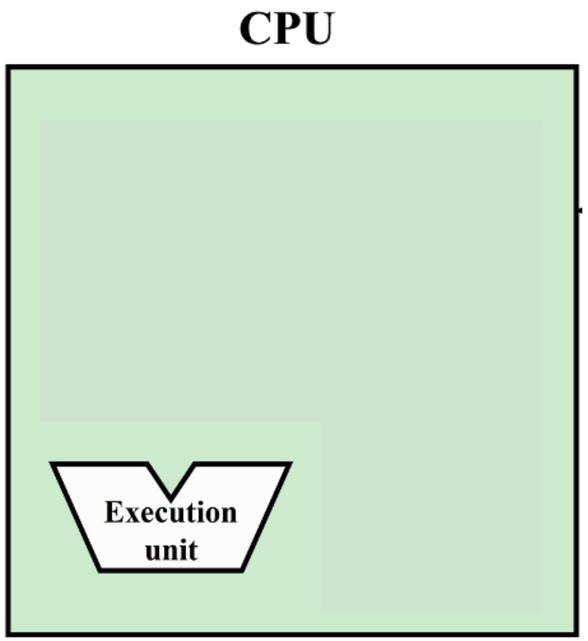
- Control Unit

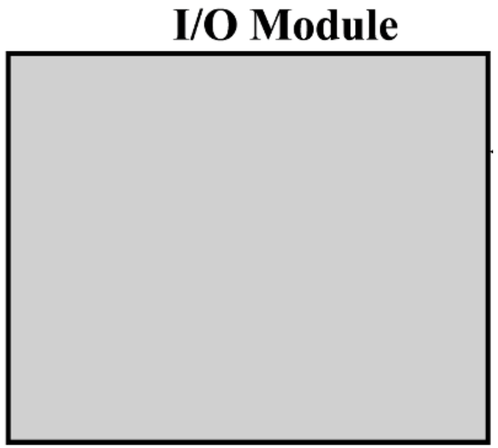
A system for software programming: I/O

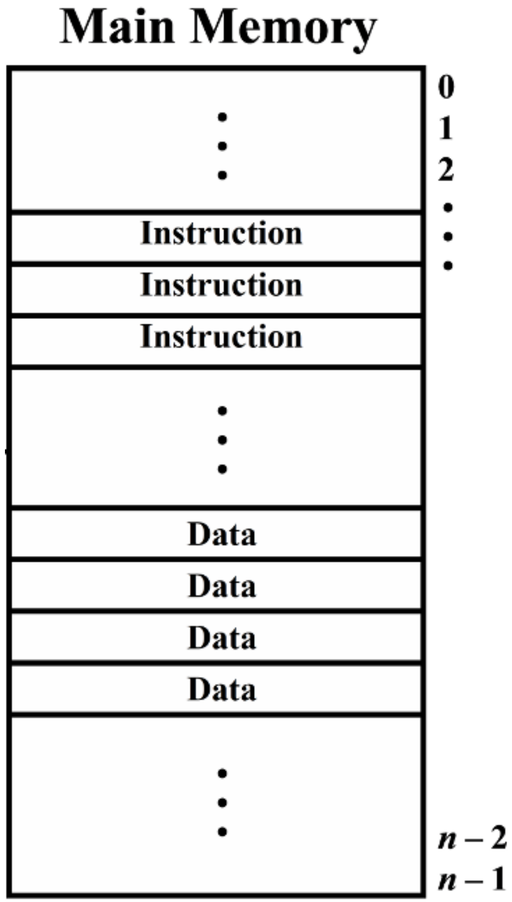


An **Input/Output module**(I/O) is needed to enter the input (data and software) and return outputs (results)

A system for software programming: memory





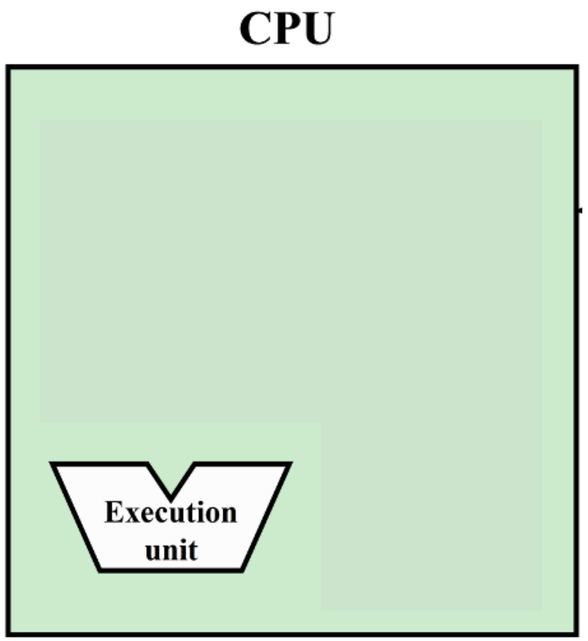


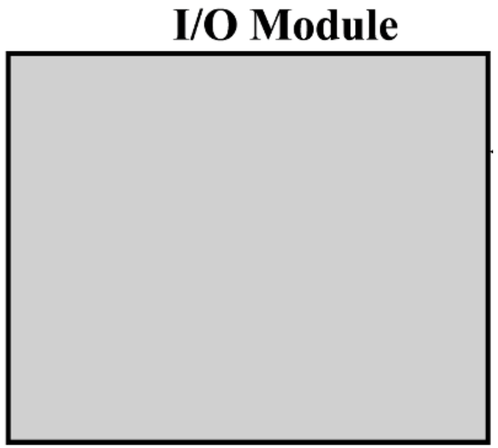
A **memory** is in charge of saving:

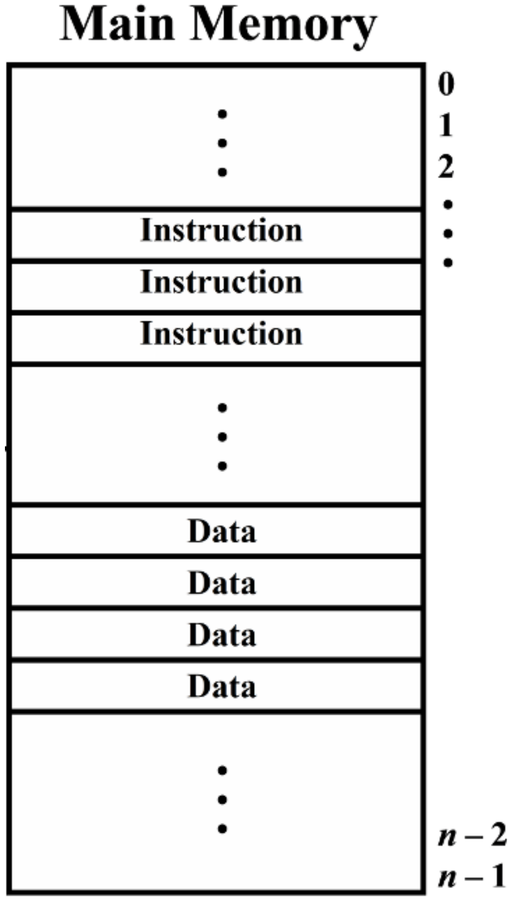
- Instructions
- Input data
- Temporary data

The memory allows to access data/instructions during the execution

A system for software programming: memory(2)



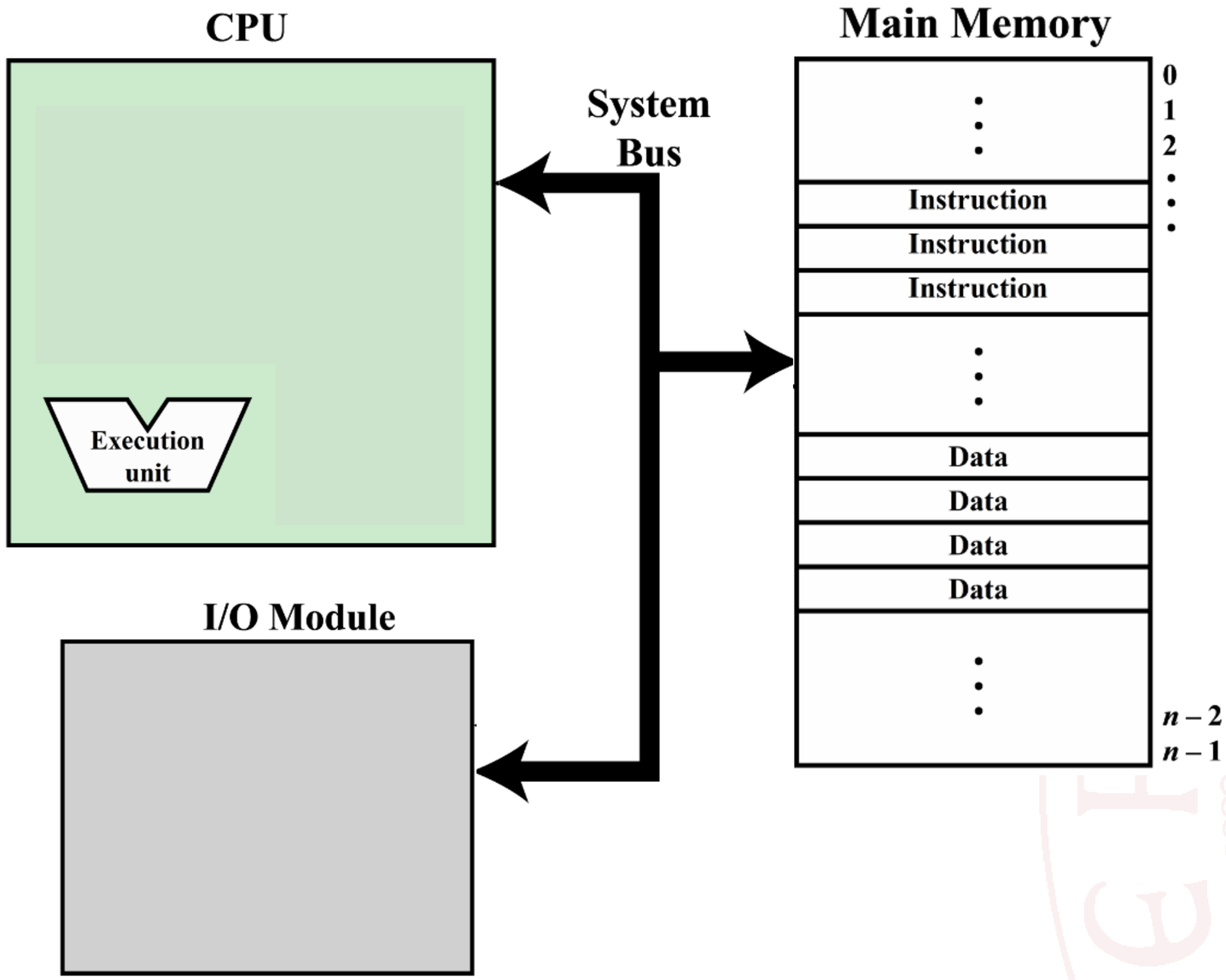




The memory is divided into **cells**

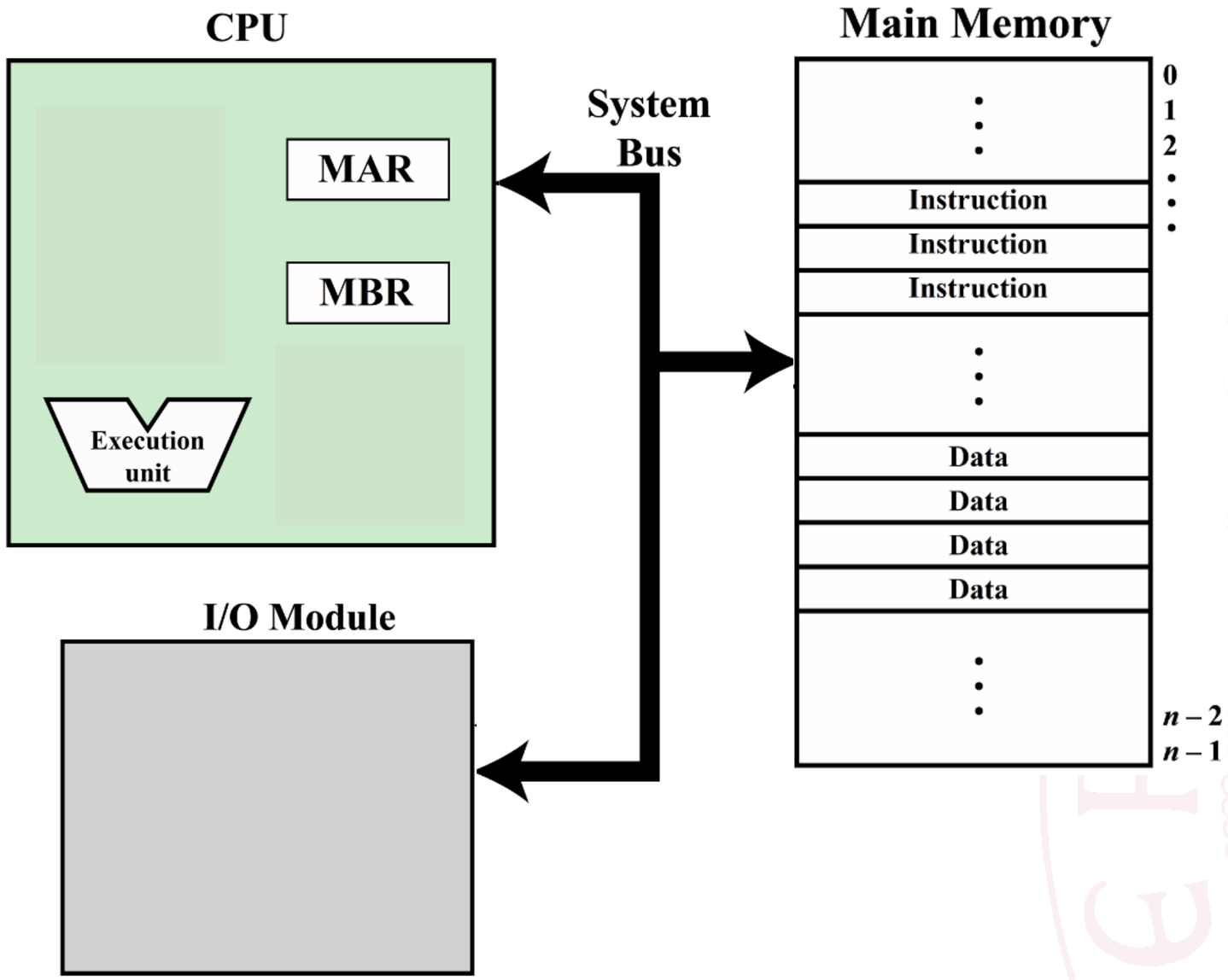
Each cell is accesible via an **address**

A system for software programming: bus



A **bus** connects the several modules

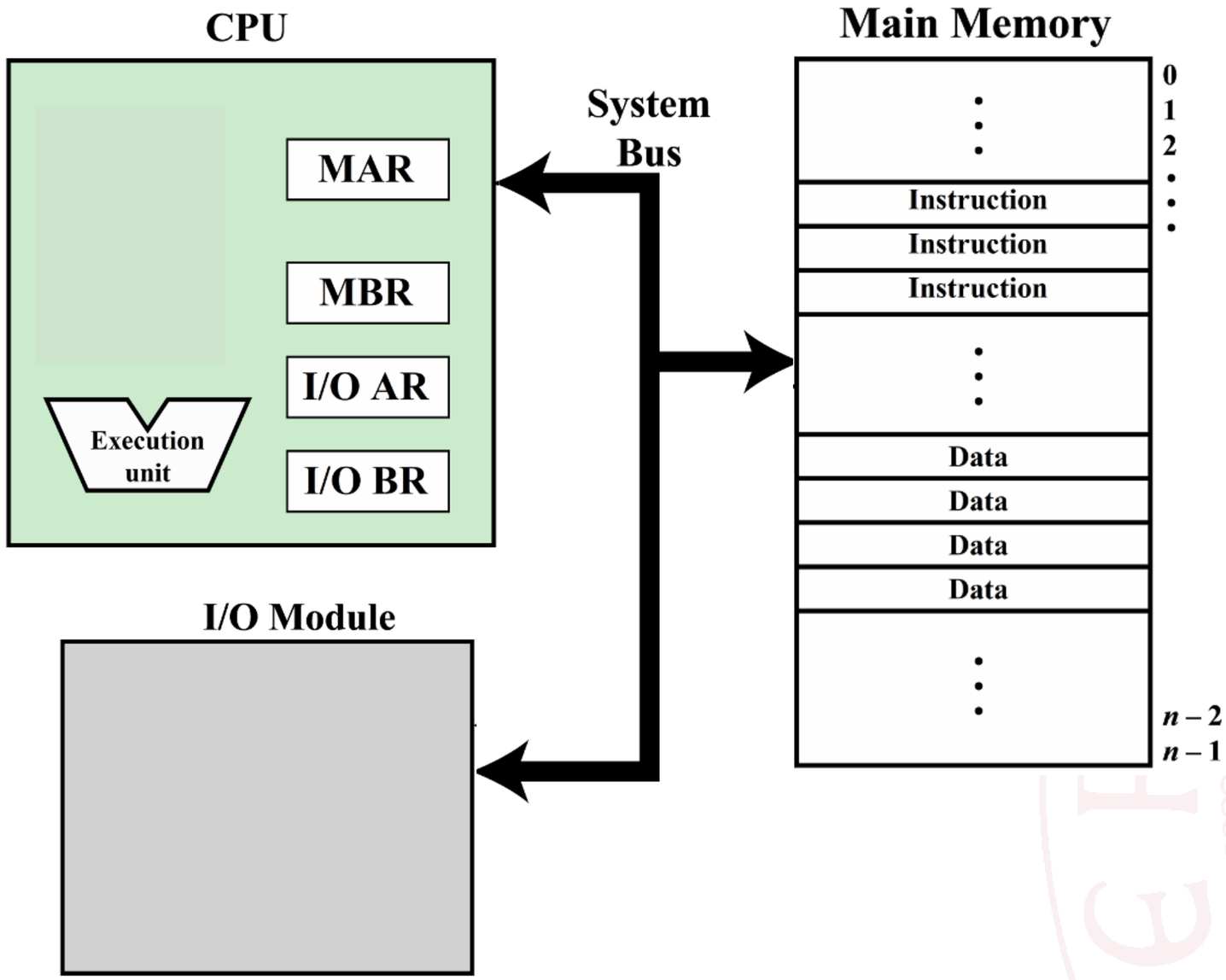
CPU and memory



The CPU reads or writes data/istructions in the memory through two registers:

- **MAR**: memory address register — Where?
- **MBR**: memory buffer register — What?

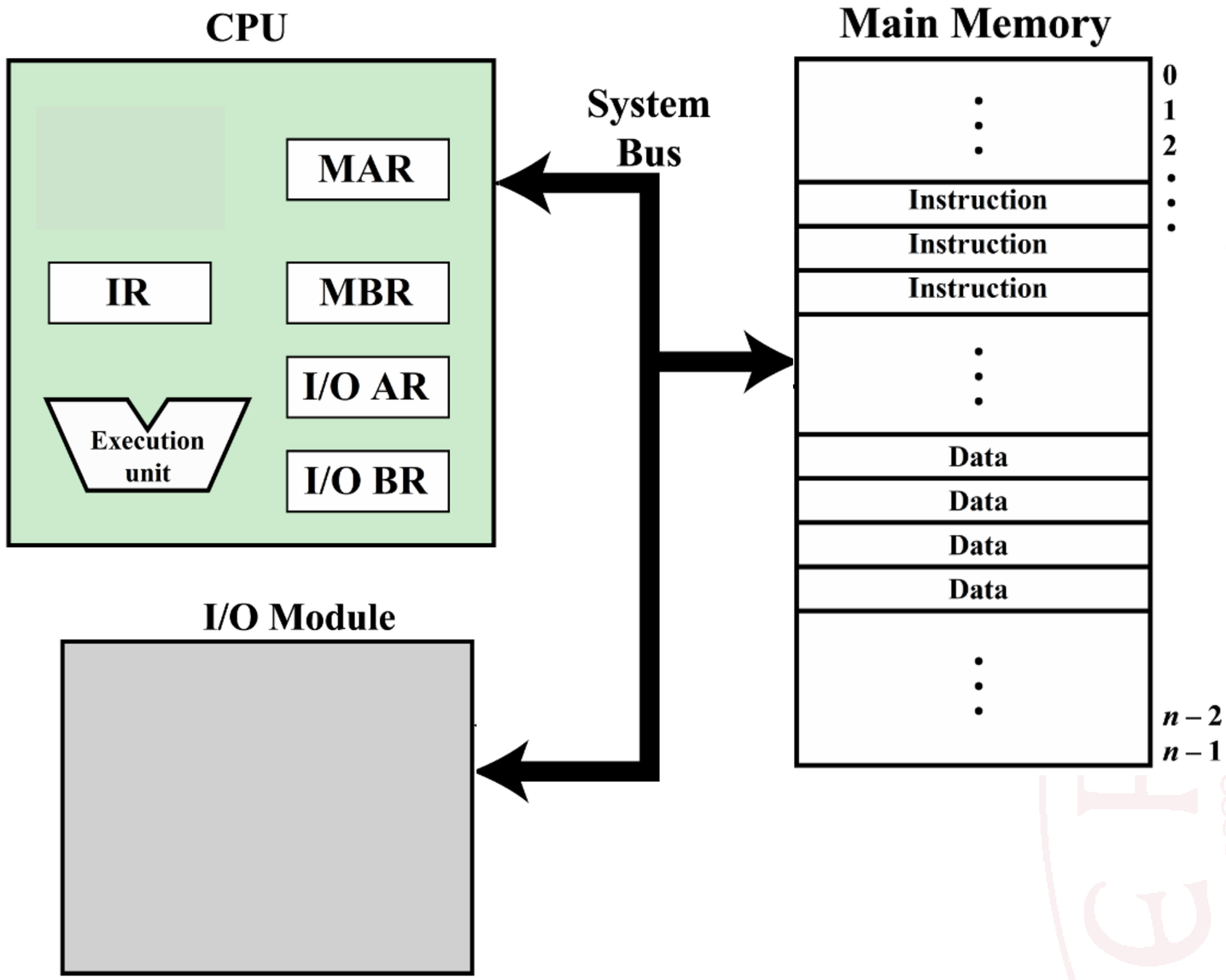
CPU and I/O



The CPU reads or writes data from I/O through two registers:

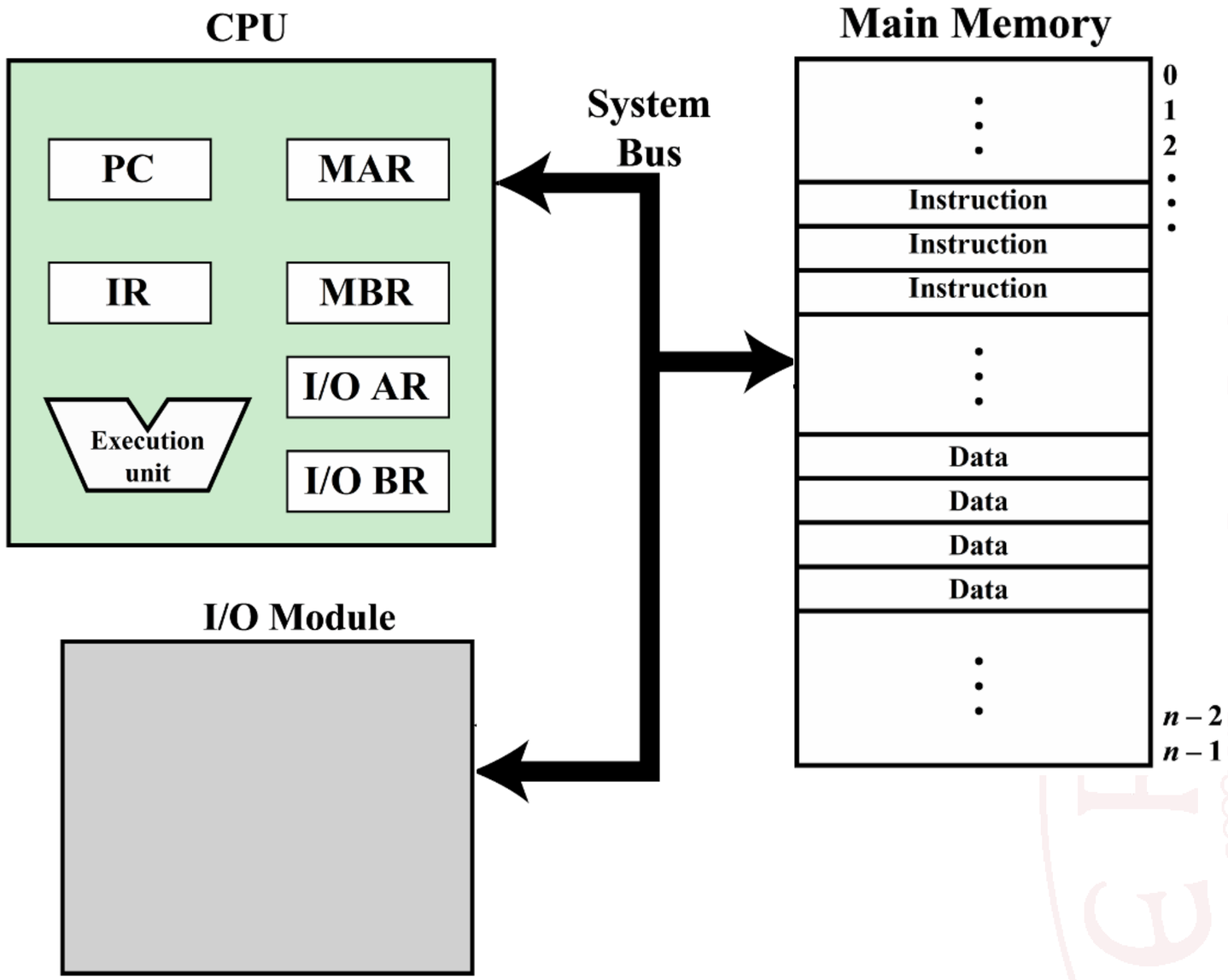
- **I/O AR**: I/O address register
- **I/O BR**: I/O buffer register

CPU and instructions



The CPU keeps a local copy of the instruction to execute in the register **IR** (Instruction Register)

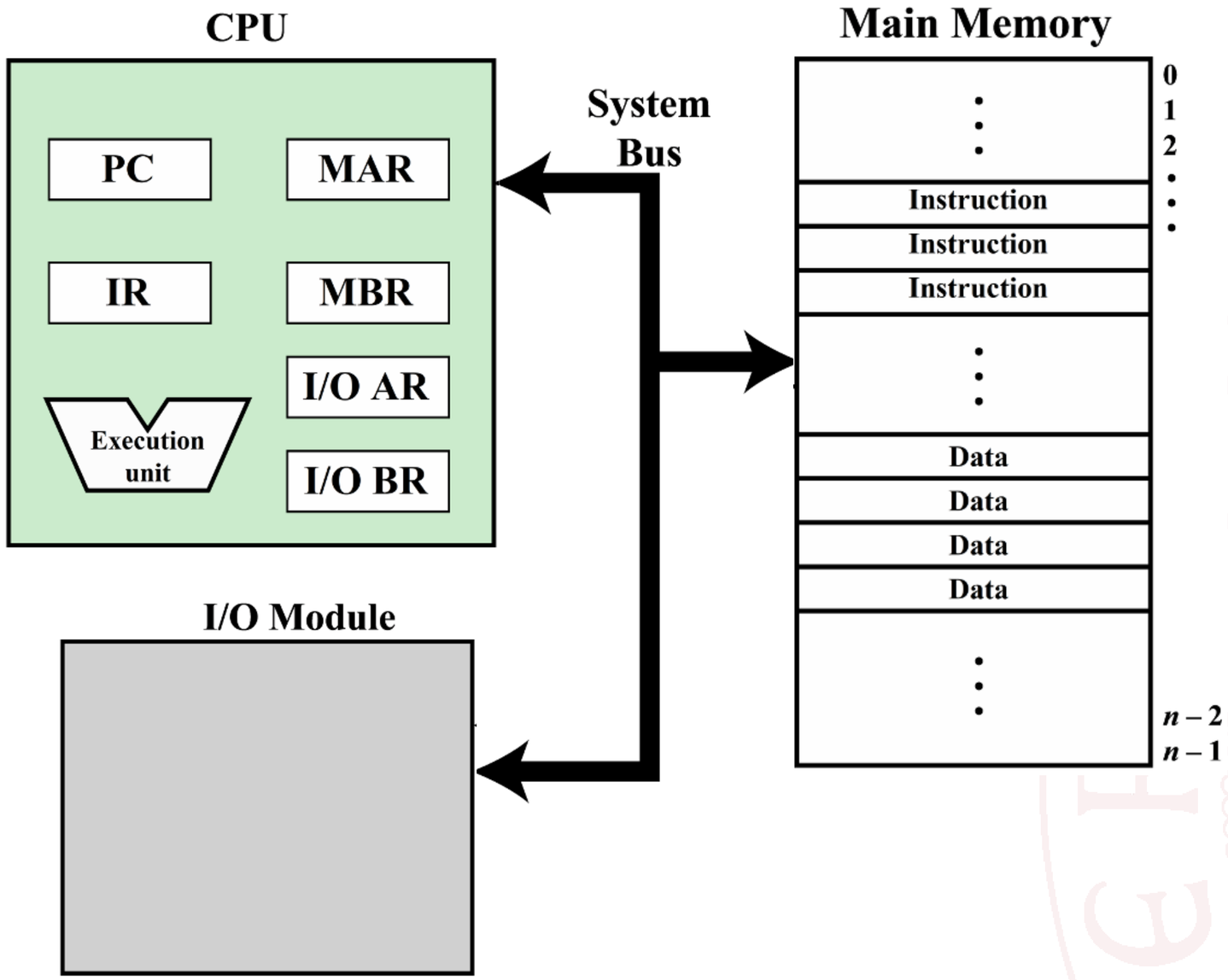
CPU and instructions (2)



The CPU must know which instruction to execute.

The register **PC** (program counter) keeps in memory the address of the next instruction to be executed

Von Neumann architecture

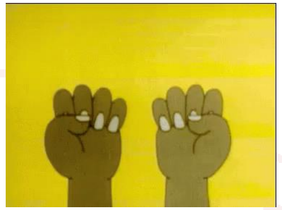


- Data and instructions share the same memory
  - **Stored-program concept**, introduced by Von Neumann
  - Previously, the program was saved on the hardware or on other devices
- Memory cells can save data or instructions
- A program in memory can be changed

The Arithmetic/Logic Unit (ALU)

- The ALU is a combinational circuit that performs a set of basic arithmetic and logic microoperations.

Let's see how it is organized internally:
two main components



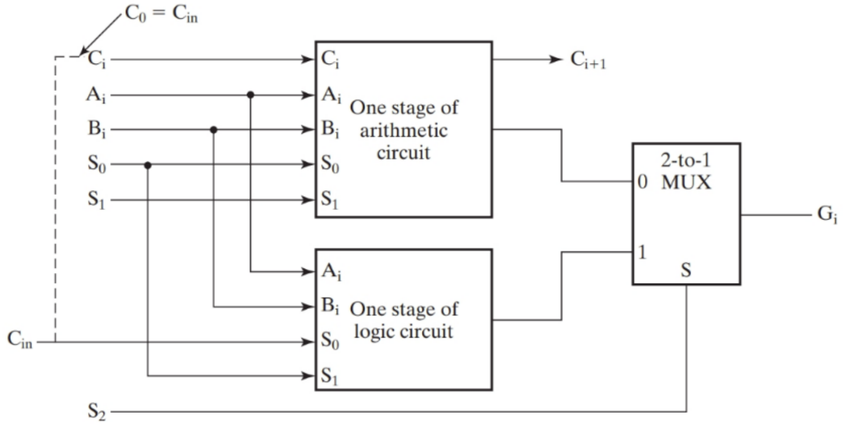


FIGURE 8-7
One Stage of ALU

input A and B (n bits)

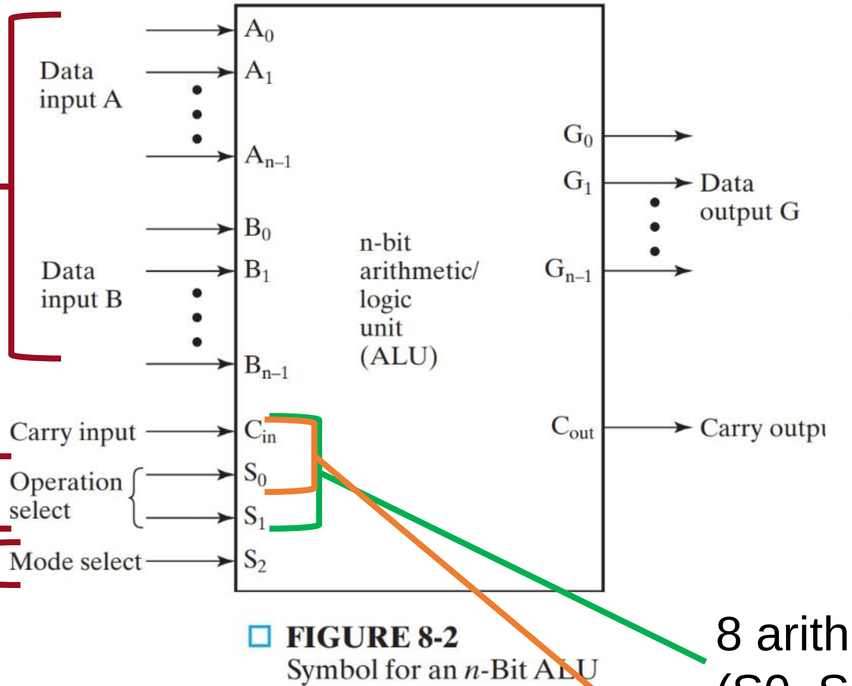


FIGURE 8-2
Symbol for an *n*-Bit ALU

Given k selection lines, it is possible to specify up to 2^k distinct operations

Mode select arithmetic or logic

8 arithmetic microoperations (S0, S1, Cin and S2=0)

4 logic microoperations (S0, Cin and S2=1)

ALU: Arithmetic Circuit

- Parallel adder with a number of full-adder circuits connected in cascade
- By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations.

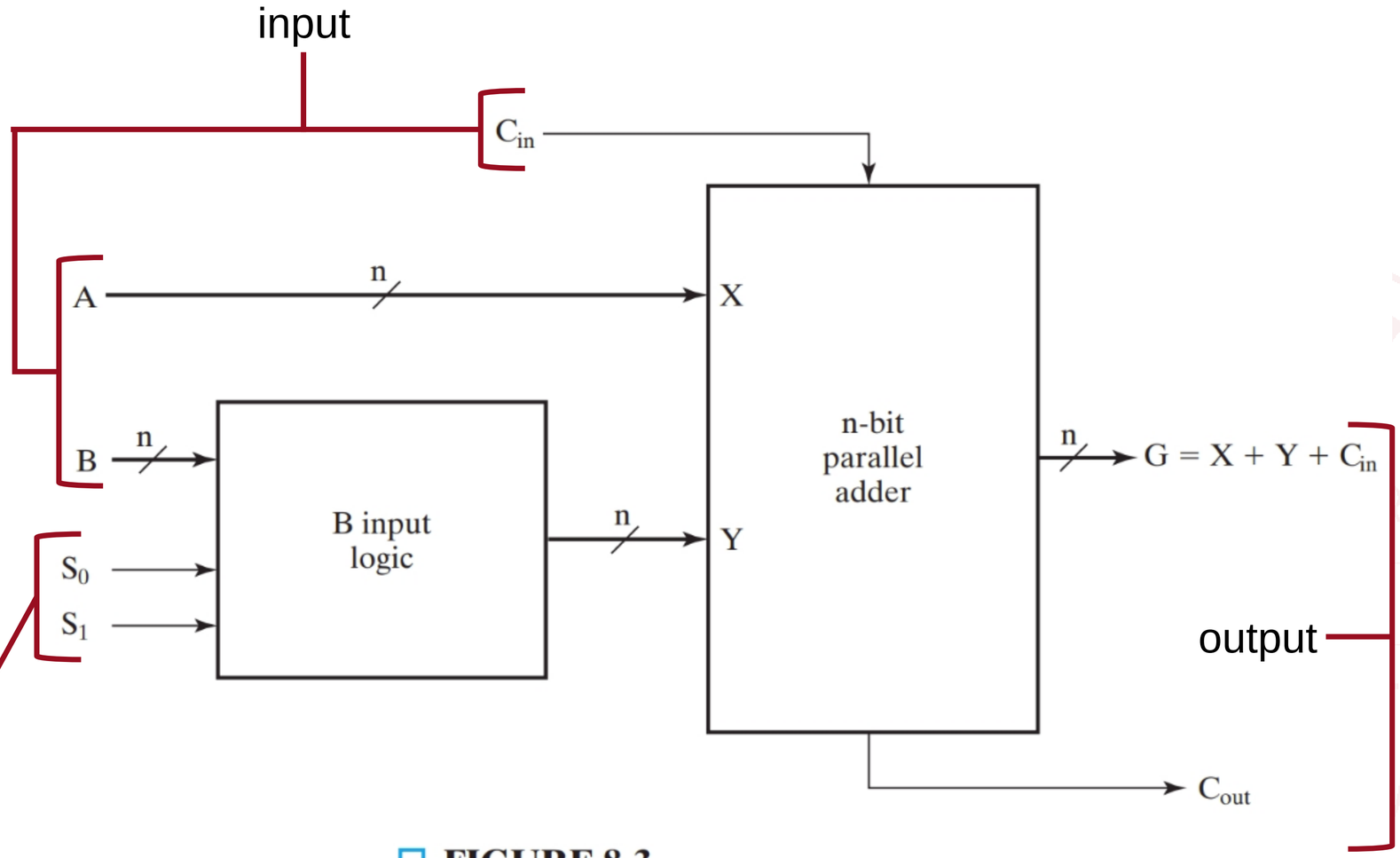


FIGURE 8-3
Block Diagram of an Arithmetic Circuit

ALU: Arithmetic Circuit (2)

- Parallel adder with a number of full-adder circuits connected in cascade
- By controlling the input Y through the two selection lines, it is possible to obtain different arithmetic operations.

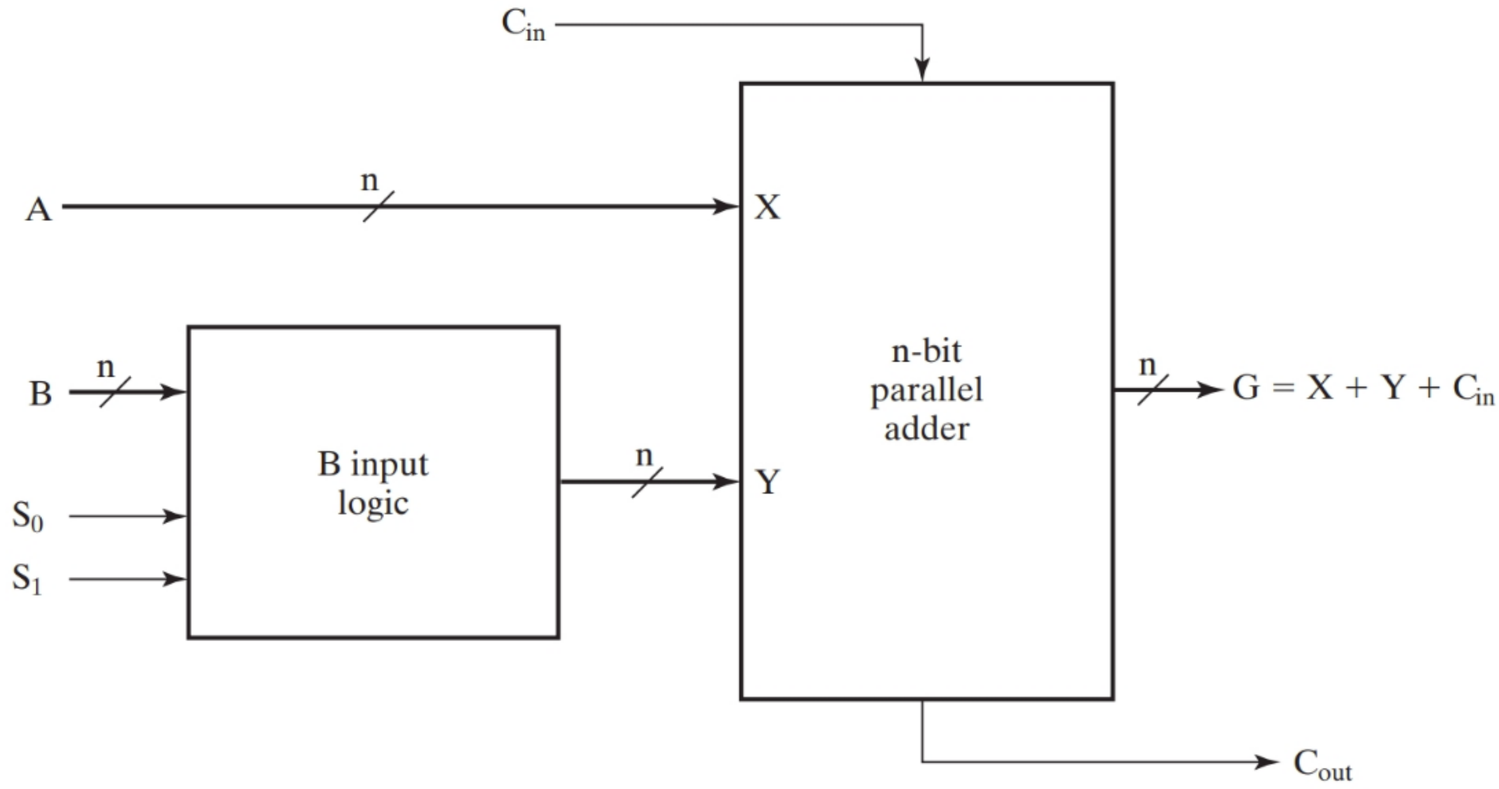


FIGURE 8-3
Block Diagram of an Arithmetic Circuit

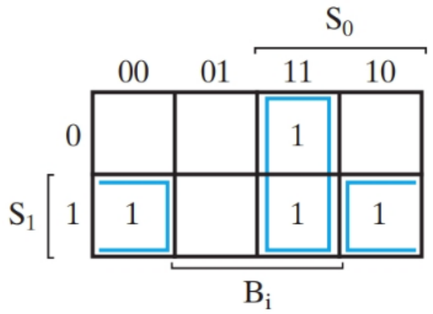
TABLE 8-1
Function Table for Arithmetic Circuit

Select		Input	$G = (A + Y + C_{in})$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)

ALU: Arithmetic Circuit (3)

Inputs			Output	
S_1	S_0	B_i	Y_i	
0	0	0	0	$Y_i = 0$
0	0	1	0	
0	1	0	0	$Y_i = B_i$
0	1	1	1	
1	0	0	1	$Y_i = \overline{B}_i$
1	0	1	0	
1	1	0	1	$Y_i = 1$
1	1	1	1	

(a) Truth table



(b) Map simplification:
$Y_i = B_i S_0 + \overline{B}_i S_1$

FIGURE 8-4
B Input Logic for One Stage of Arithmetic Circuit

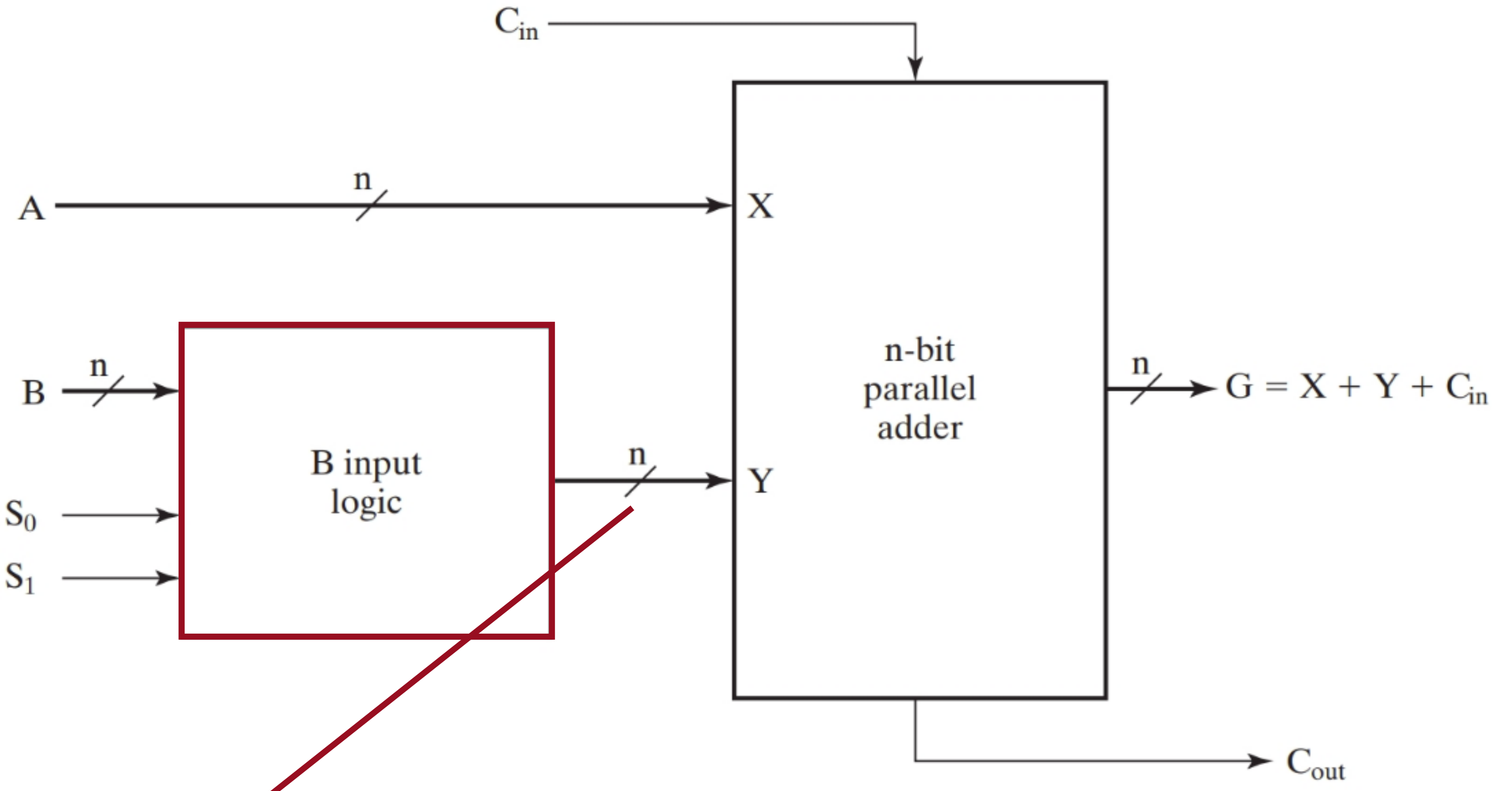


FIGURE 8-3
Block Diagram of an Arithmetic Circuit

This logic corresponds to a 2-to-1 multiplexer with B_i on the select input and S_1 and S_0 on the data input

$$Y_i = B_i S_0 + \overline{B_i} S_1$$

ALU: Arithmetic Circuit (4)

Input Logic
(input: B, S0, S1
output: Y)

$$Y_i = B_i S_0 + \overline{B_i} S_1$$

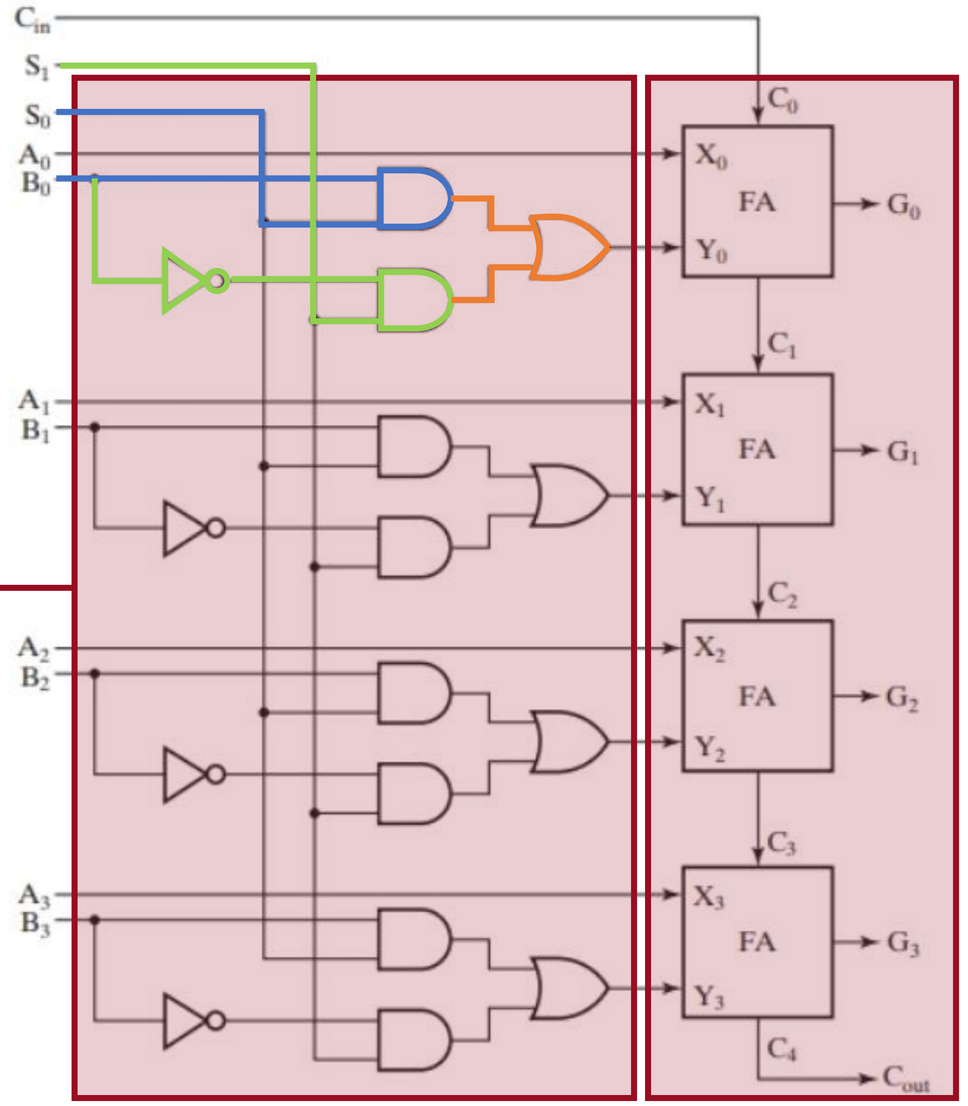
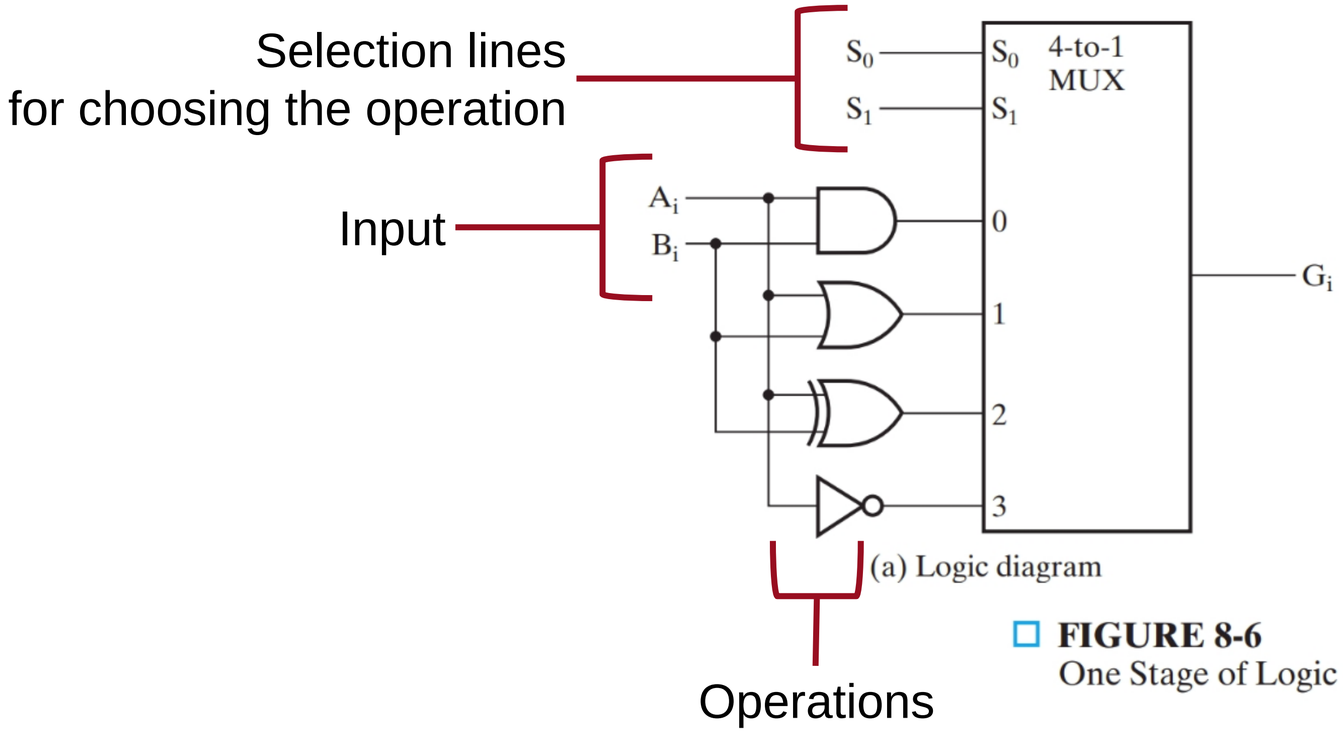


FIGURE 8-5
Logic Diagram of a 4-Bit Arithmetic Circuit

Parallel adder
(input: A, Cin, Y
output: G, Cout)

ALU: Logic Circuit

- Manipulate the bits of the operands giving bitwise operations
- Four commonly used logic operations: AND, OR, XOR (exclusive-OR) and NOT
- Other more complex operations can be conveniently derived from them.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT

(b) Function table

FIGURE 8-6
One Stage of Logic Circuit

ALU: Arithmetic/Logic Unit

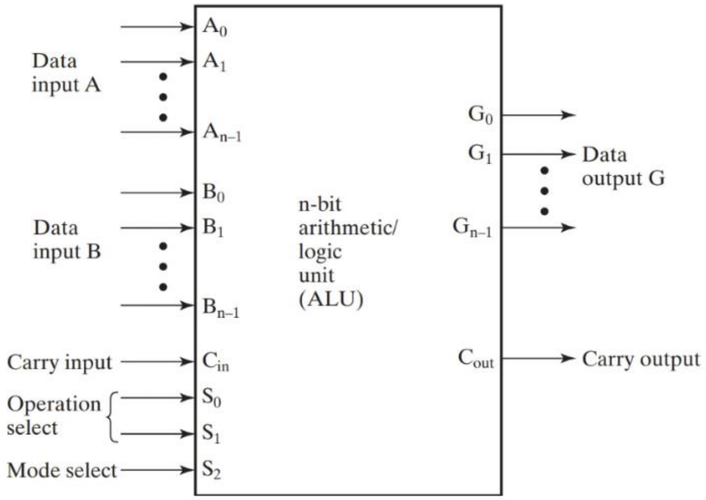


FIGURE 8-2
Symbol for an *n*-Bit ALU

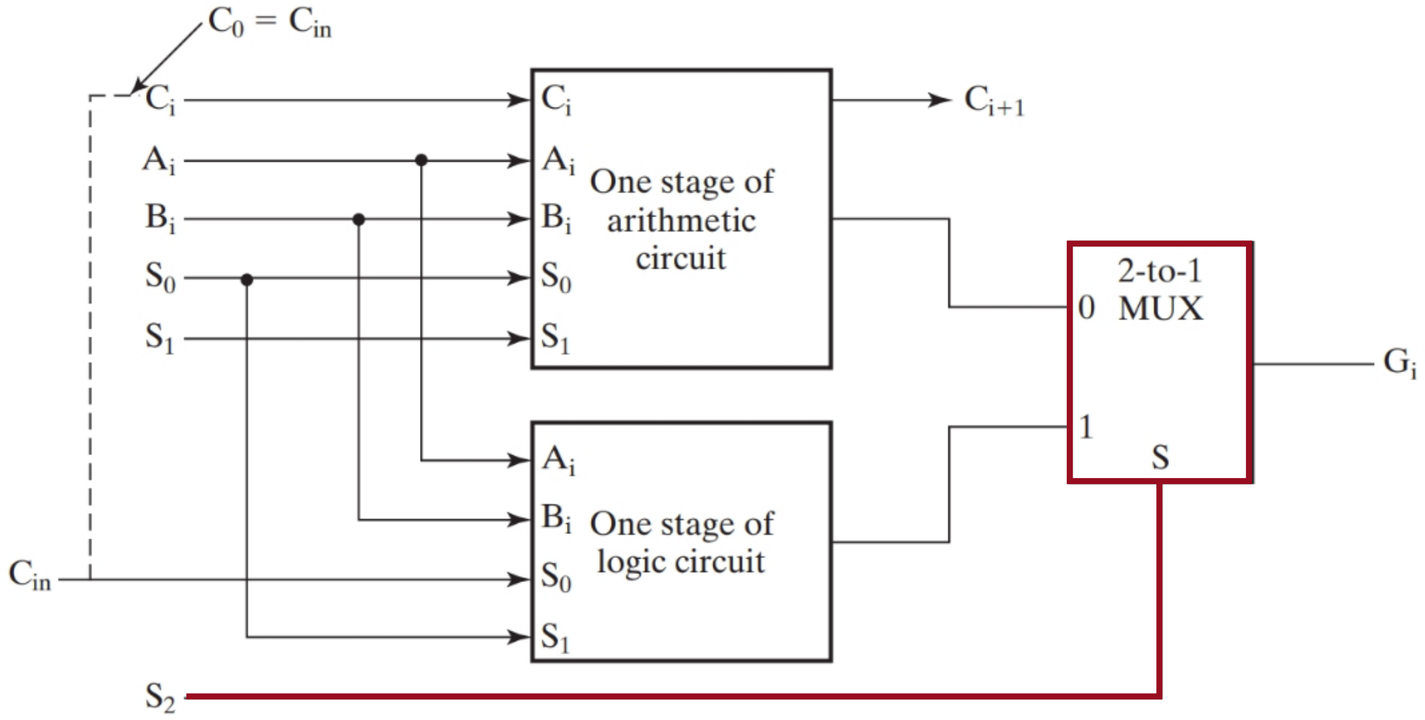


FIGURE 8-7
One Stage of ALU

Selection of the arithmetic or logic circuit (S2)

ALU: Arithmetic/Logic Unit

TABLE 8-2
Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)

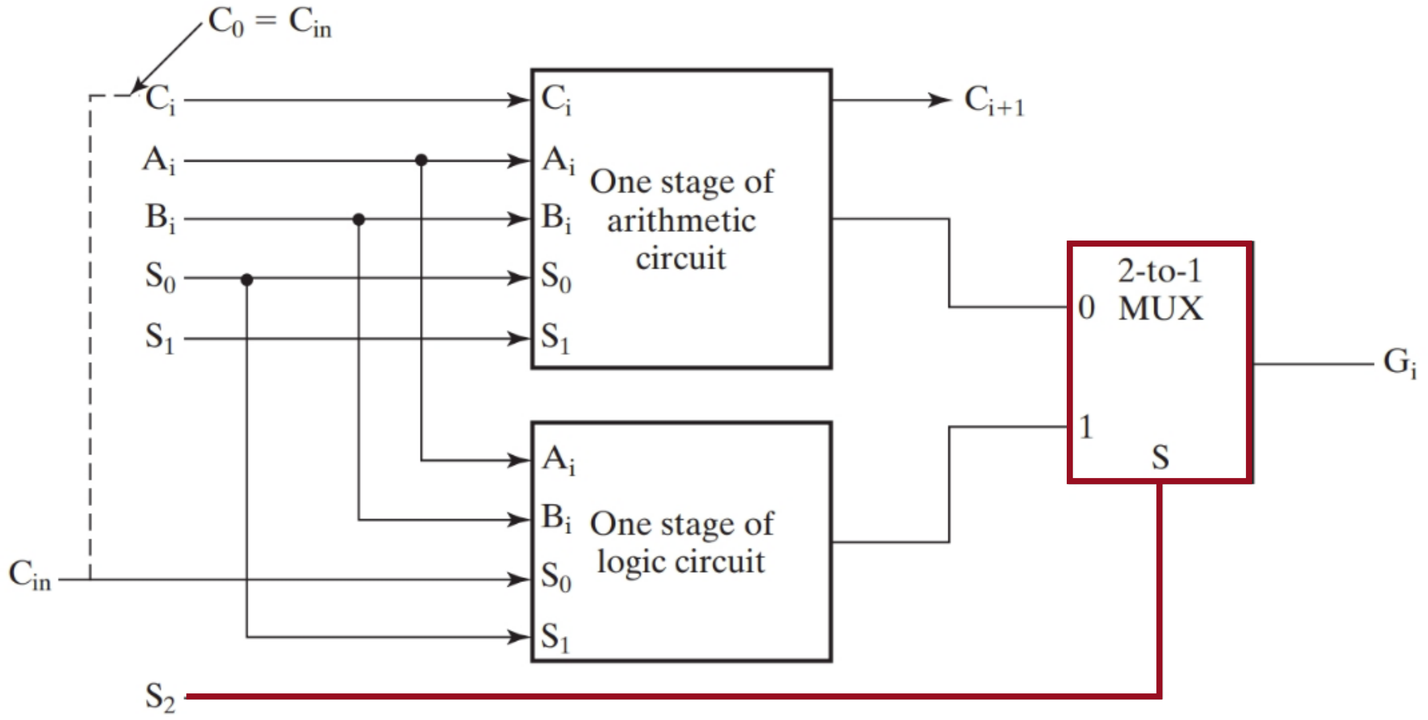


FIGURE 8-7
One Stage of ALU

Selection input S1 has no effect during the logic operations and is marked with X to indicate that its value may be either 0 or 1

Shifter

- It is often useful to shift the data by n bits positions in datapath applications
- With this purpose, a Barrell Shifter shifts or rotates the input data bits by a specified number of bit positions in a single clock cycle

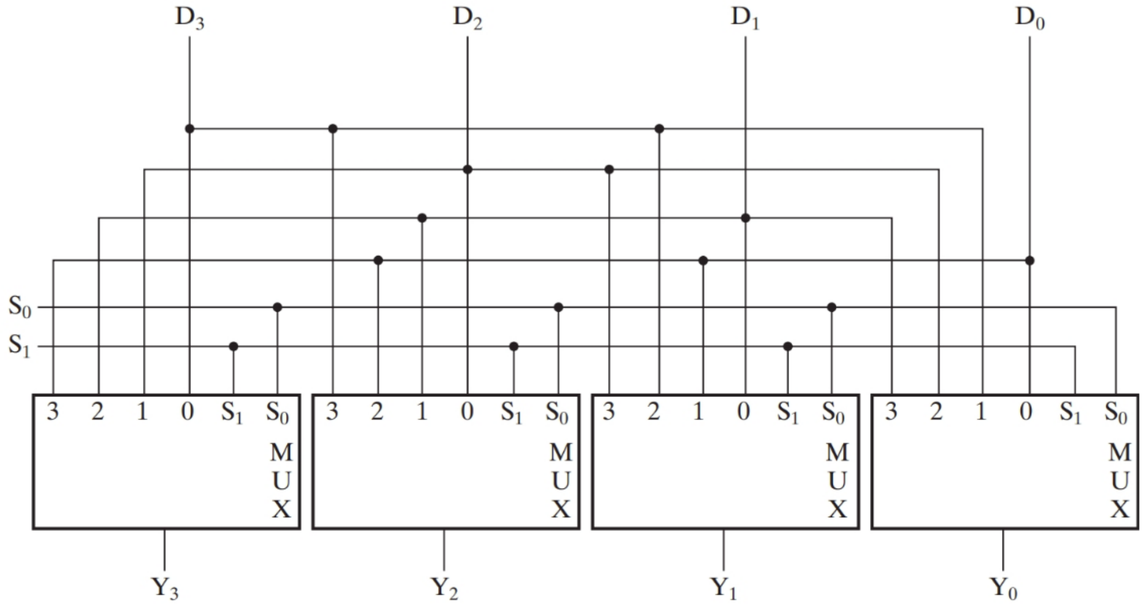
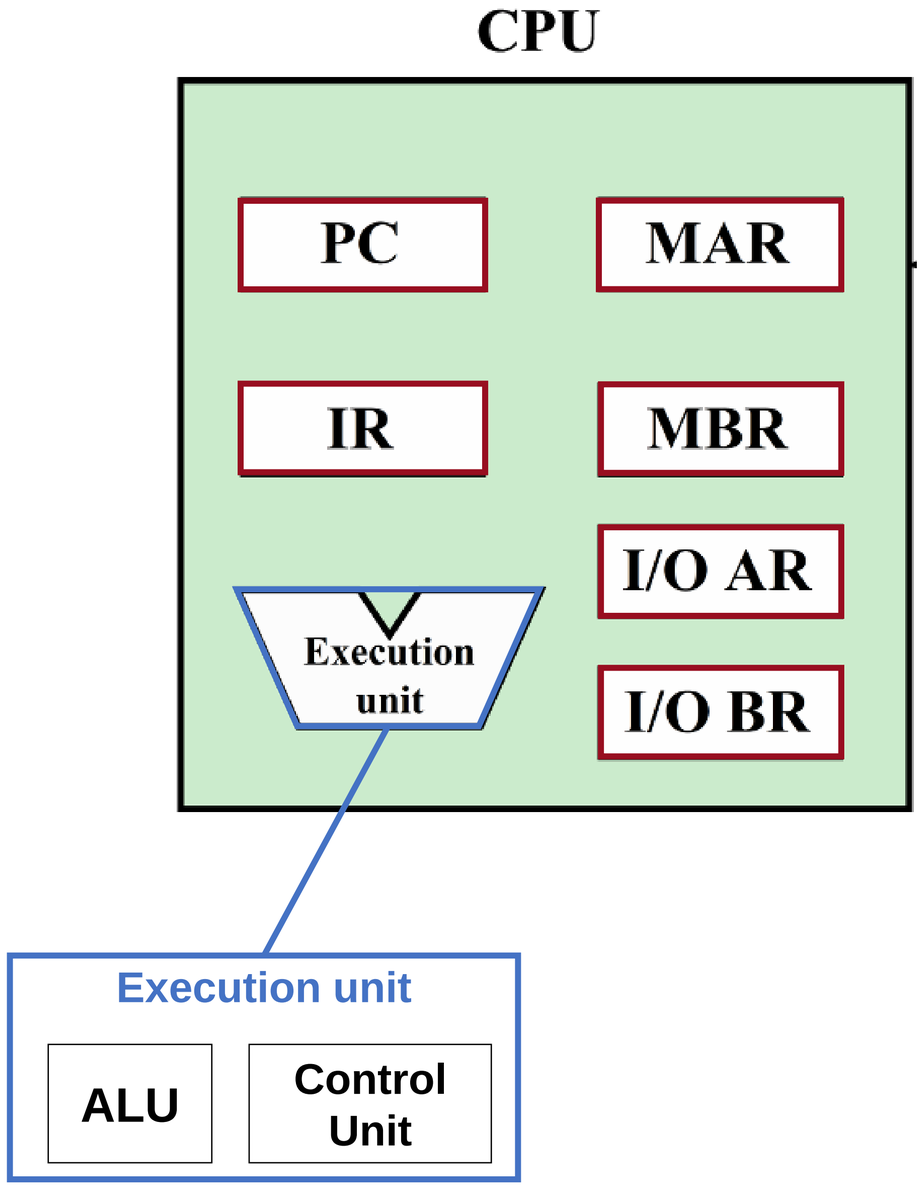


FIGURE 8-9
4-Bit Barrel Shifter

TABLE 8-3
Function Table for 4-Bit Barrel Shifter

Select		Output				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	Operation
0	0	D_3	D_2	D_1	D_0	No rotation
0	1	D_2	D_1	D_0	D_3	Rotate one position
1	0	D_1	D_0	D_3	D_2	Rotate two positions
1	1	D_0	D_3	D_2	D_1	Rotate three positions

Registers



Registers allows data storage inside the CPU

Reading/writing access to the registers is faster than memory access.

There is a set of other general registers to store data (e.g., R0-R7)

Example of registers in a generic datapath

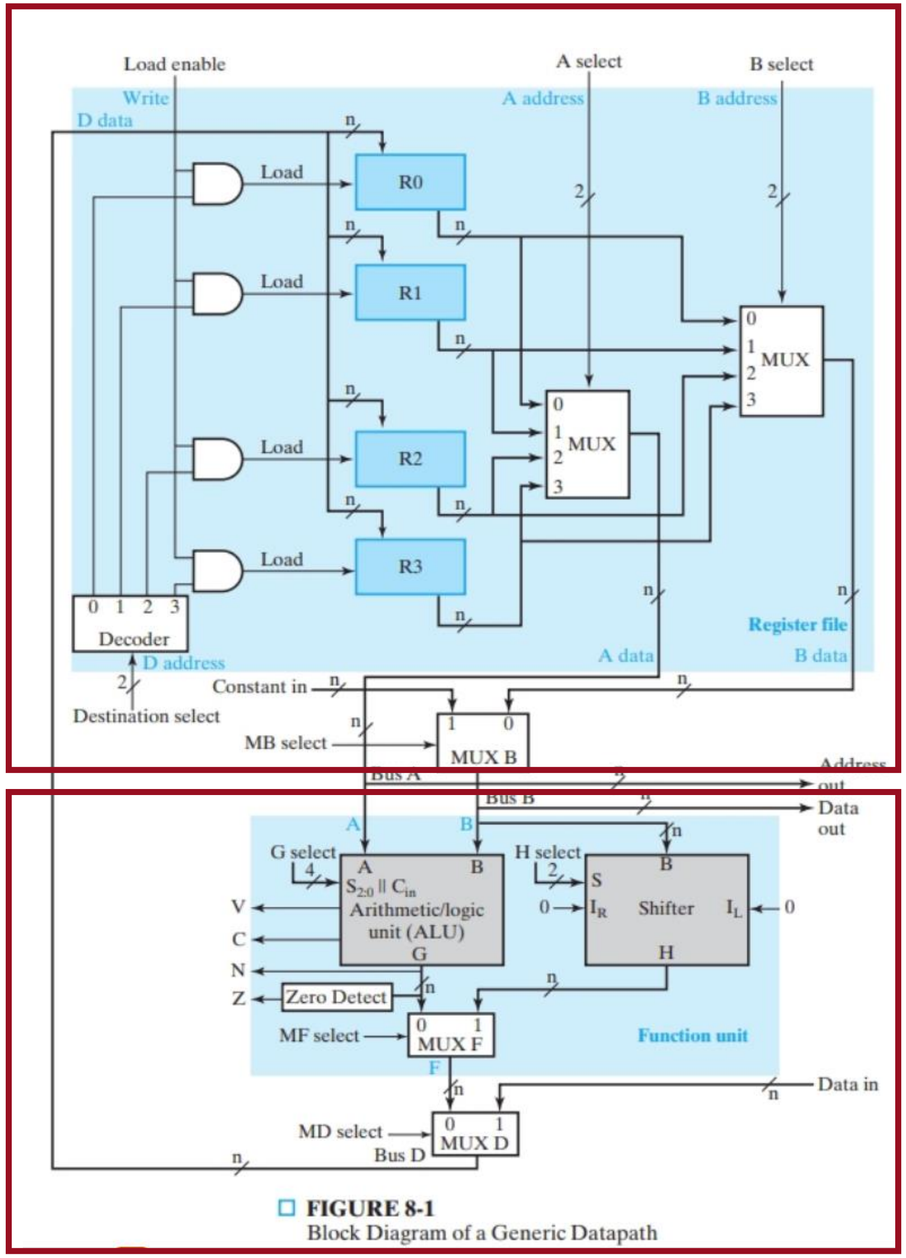


FIGURE 8-1
Block Diagram of a Generic Datapath

Register file

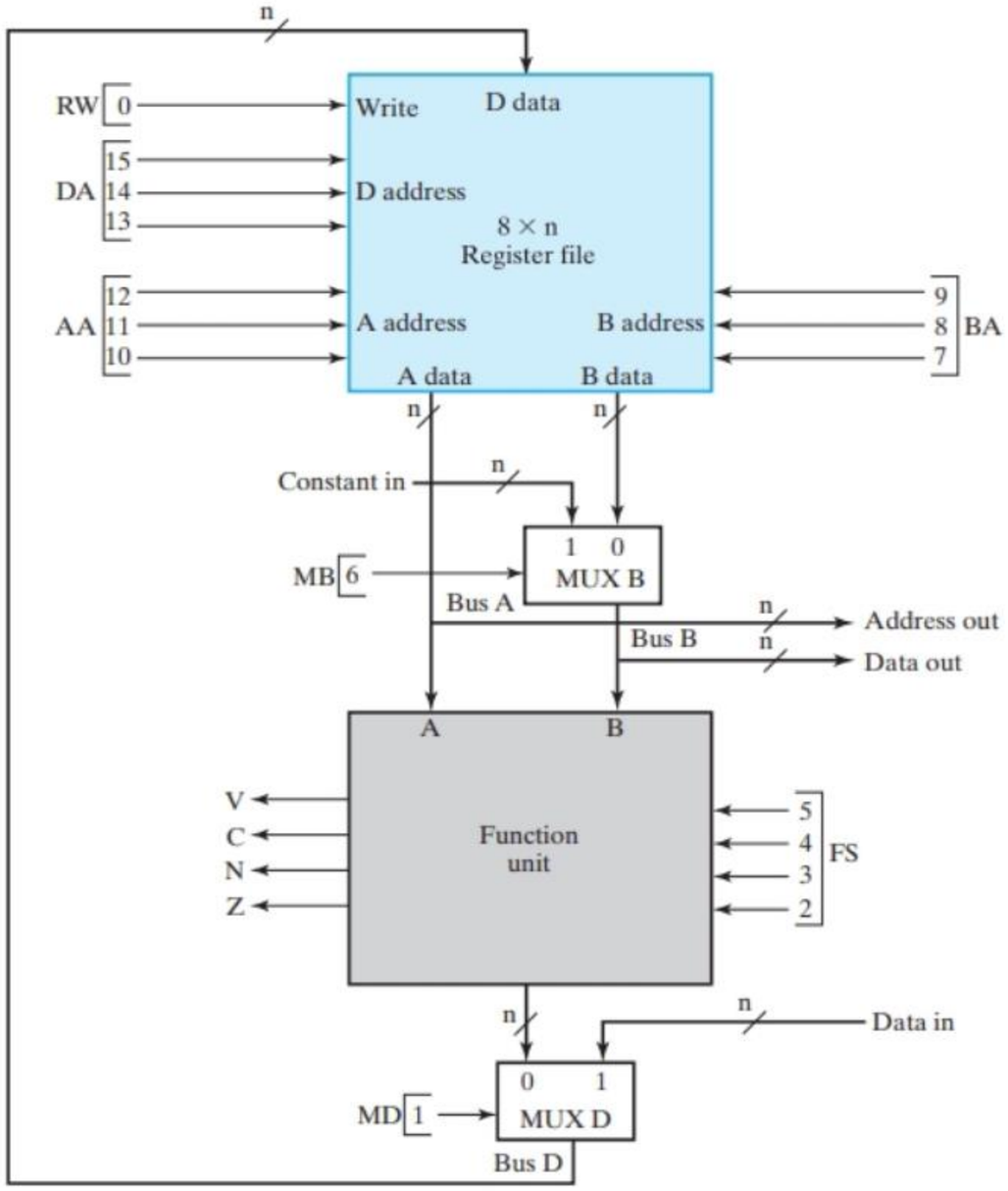
- 4 n-bits registers (R0-R3)
- 2 MUX for the input A e B of the ALU and of the shifter
- *A select* e *B select* select the inputs from registers R0-R3

Function unit

Datapath representation

- A typical datapath has more than four registers and they may be organized into a *register file*
- The size of the register file is $2^m \times n$, where m is the number of register address bits and n is the number of bits per register.
- The selection of input, operations, output is executed through a **control word** that identifies them uniquely
- The operations are executed in a single clock cycle

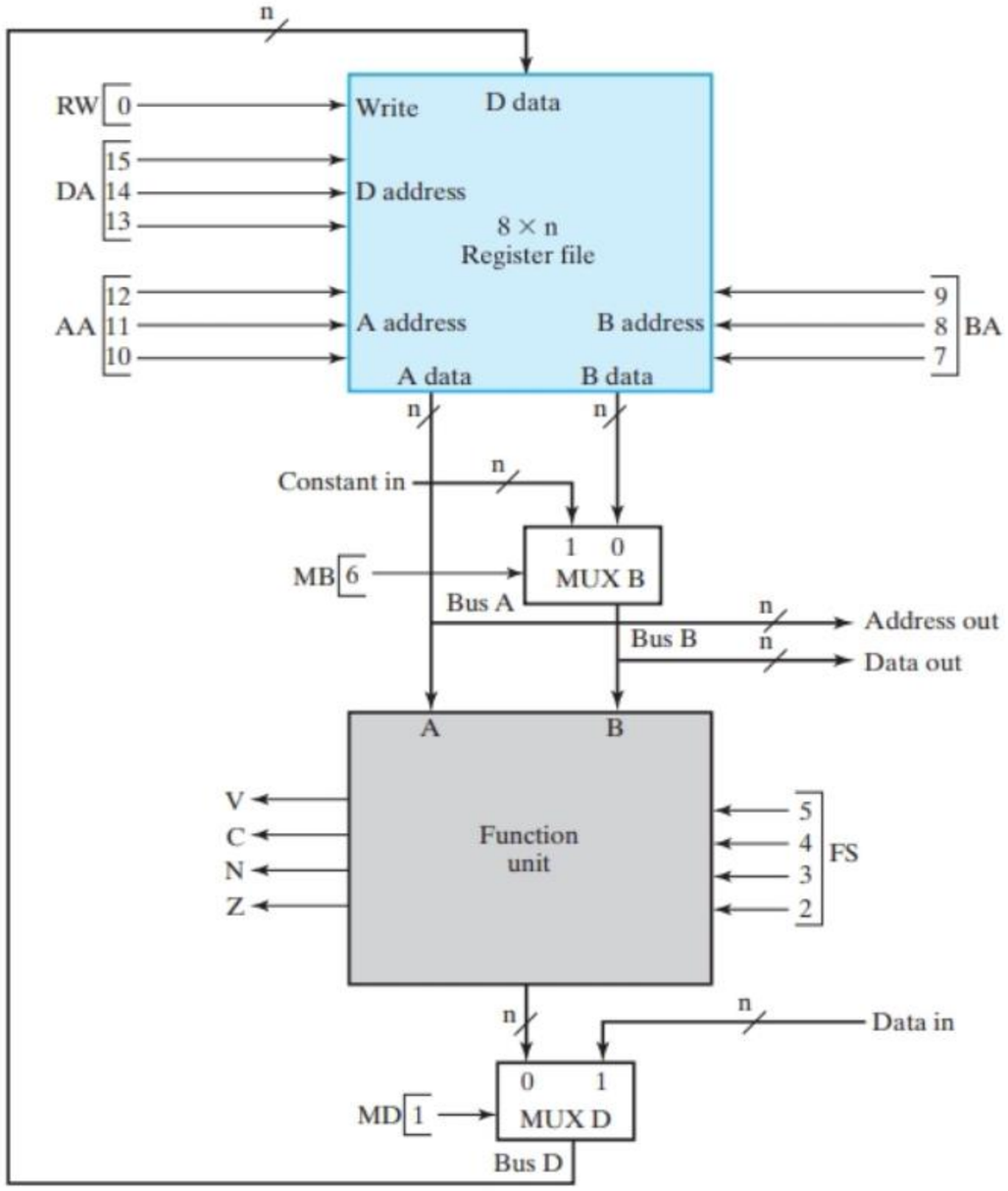
We will see it next lecture



(a) Block diagram

Datapath representation

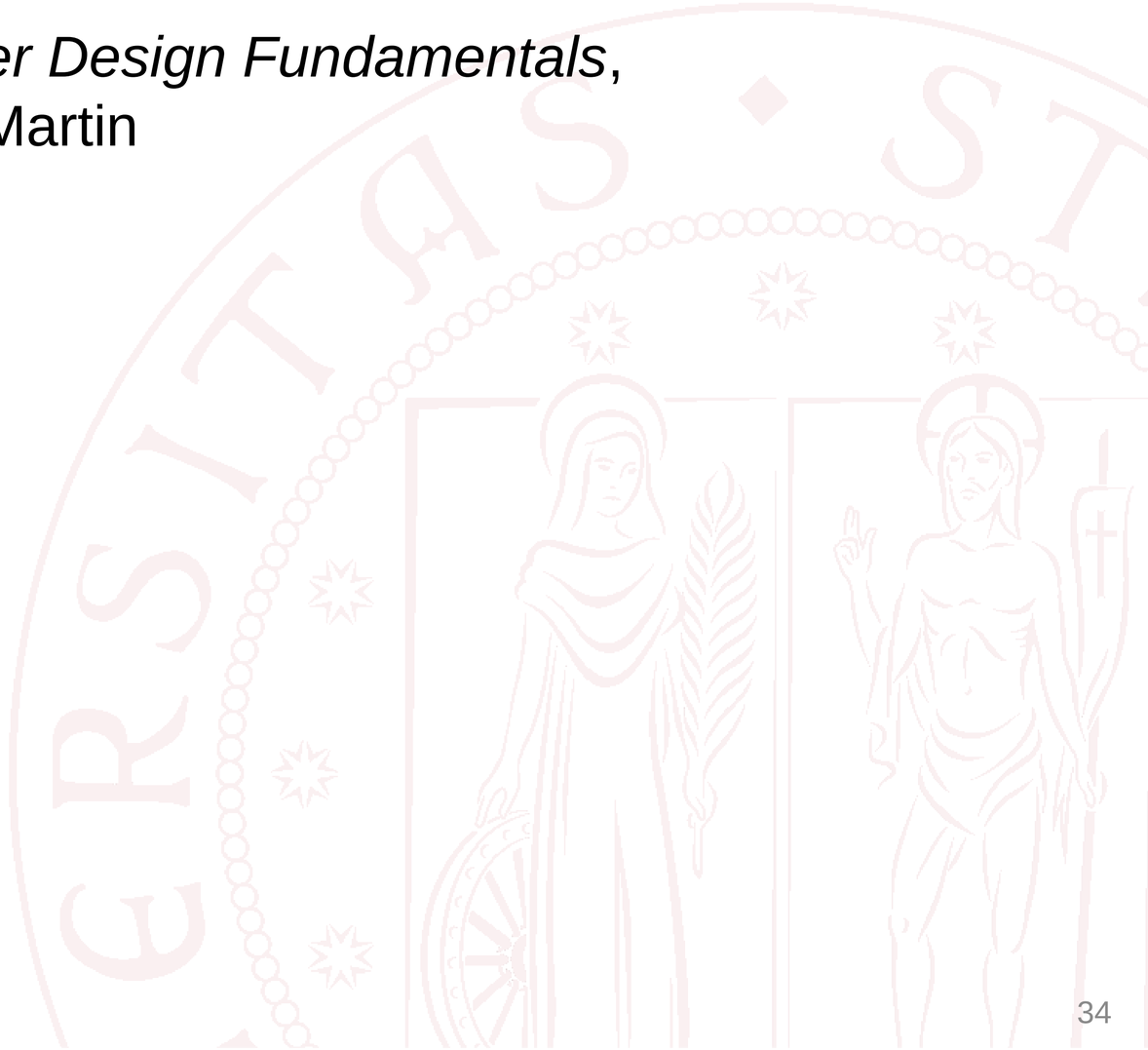
- A typical datapath has more than four registers and they may be organized into a *register file*
- The size of the register file is $2^m \times n$, where m is the number of register address bits and n is the number of bits per register.
- The selection of input, operations, output is executed through a **control word** that identifies them uniquely
- The operations are executed in a single clock cycle



(a) Block diagram

Disclaimer

Figures from *Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano | Kime | Martin



Questions