LECTURE 1

Introduction to Digital Systems. Combinational Circuits. Digital Integrated Circuits.

Lecture 1

Dana Utebayeva

Outline

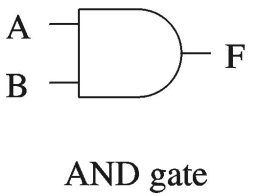
- Basic concepts
  - Simple gates
  - Completeness
- Logic functions
  - Expressing logic functions
  - Equivalence
- Boolean algebra
  - Boolean identities
  - Logical equivalence
- Logic Circuit Design Process
- Deriving logical expressions
  - Sum-of-products form
  - Product-of-sums form
- Simplifying logical expressions
  - Algebraic manipulation
  - Karnaugh map method
  - Quine-McCluskey method
- Generalized gates
- Multiple outputs
- Implementation using other gates (NAND and XOR)

Introduction

- Hardware consists of a few simple building blocks
  - These are called *logic gates*
    - AND, OR, NOT, …
    - NAND, NOR, XOR, …
- Logic gates are built using transistors
    - NOT gate can be implemented by a single transistor
    - AND gate requires 3 transistors
- Transistors are the fundamental devices
    - Pentium consists of 3 million transistors
    - Compaq Alpha consists of 9 million transistors
    - Now we can build chips with more than 100 million transistors

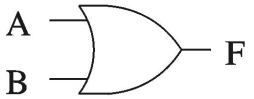
Basic Concepts

- Simple gates
  - AND
  - OR
  - NOT
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination
- Other methods
  - Logic expressions
  - Logic diagrams



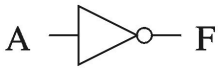
AND gate

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



OR gate

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



NOT gate

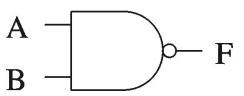
A	F
0	1
1	0

Logic symbol

Truth table

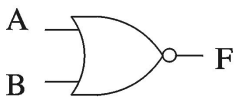
Basic Concepts (cont'd)

- Additional useful gates
  - NAND
  - NOR
  - XOR
- NAND = AND + NOT
- NOR = OR + NOT
- XOR implements exclusive-OR function
- NAND and NOR gates require only 2 transistors
  - AND and OR need 3 transistors!



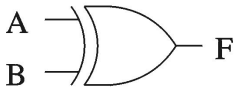
NAND gate

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



NOR gate

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



XOR gate

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Logic symbol

Truth table

Basic Concepts (cont'd)

- Number of functions
  - With N logical variables, we can define
    2^{2N} functions
  - Some of them are useful
    - AND, NAND, NOR, XOR, …
  - Some are not useful:
    - Output is always 1
    - Output is always 0
  - "Number of functions" definition is useful in proving completeness property

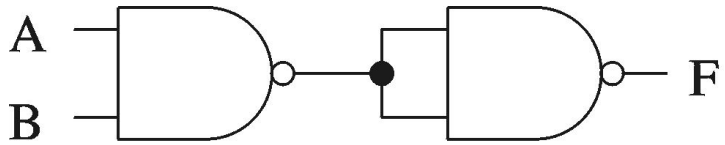
Basic Concepts (cont'd)

- Complete sets
  - A set of gates is complete
    - if we can implement any logical function using only the type of gates in the set
      - You can uses as many gates as you want
  - Some example complete sets
    - {AND, OR, NOT}
    - {AND, NOT}
    - {OR, NOT}
    - {NAND}
    - {NOR}

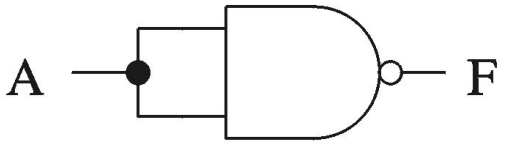
    Not a ~~minimal~~ complete set
  - Minimal complete set
      - A complete set with no redundant elements.

Basic Concepts (cont'd)

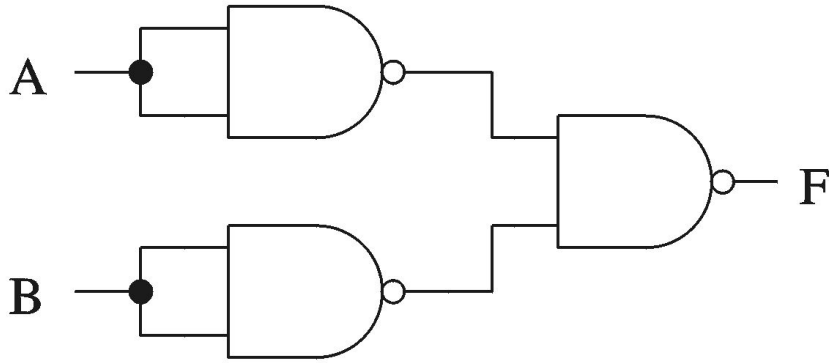
- Proving NAND gate is universal



AND gate



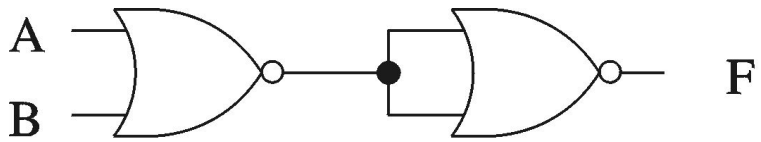
NOT gate



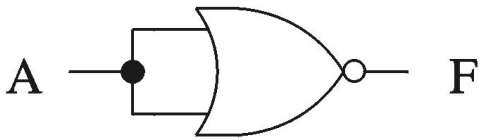
OR gate

Basic Concepts (cont'd)

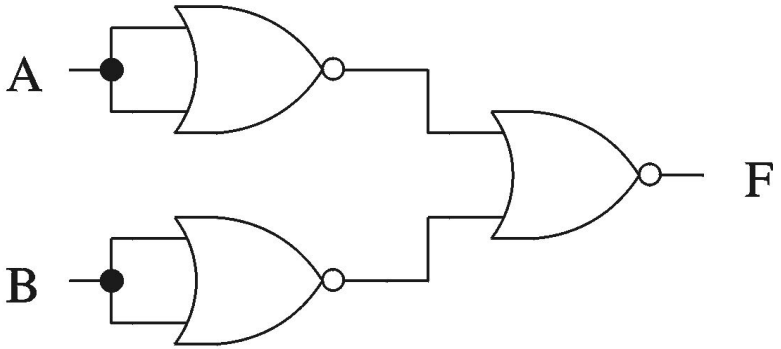
- Proving NOR gate is universal



OR gate



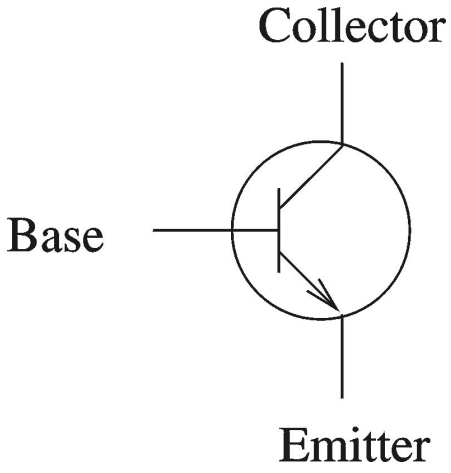
NOT gate



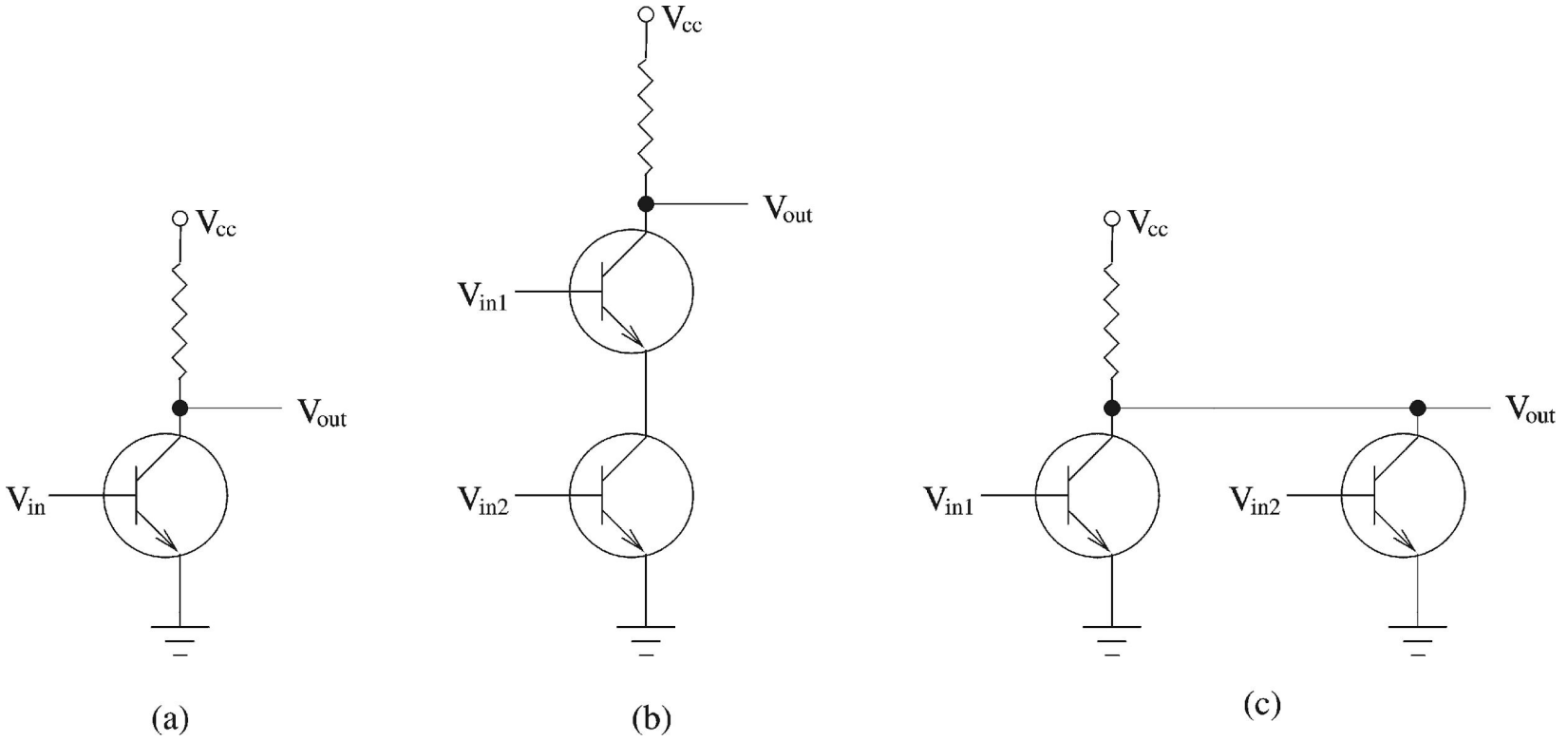
AND gate

Logic Chips

- Basic building block:
  - Transistor
- Three connection points
  - Base
  - Emitter
  - Collector
- Transistor can operate
  - Linear mode
    - Used in amplifiers
  - Switching mode
    - Used to implement digital circuits



Logic Chips (cont'd)



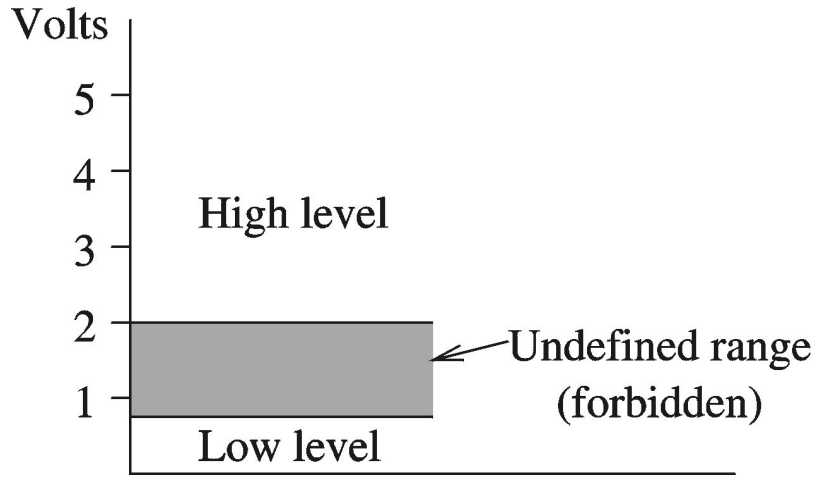
NOT

NAND

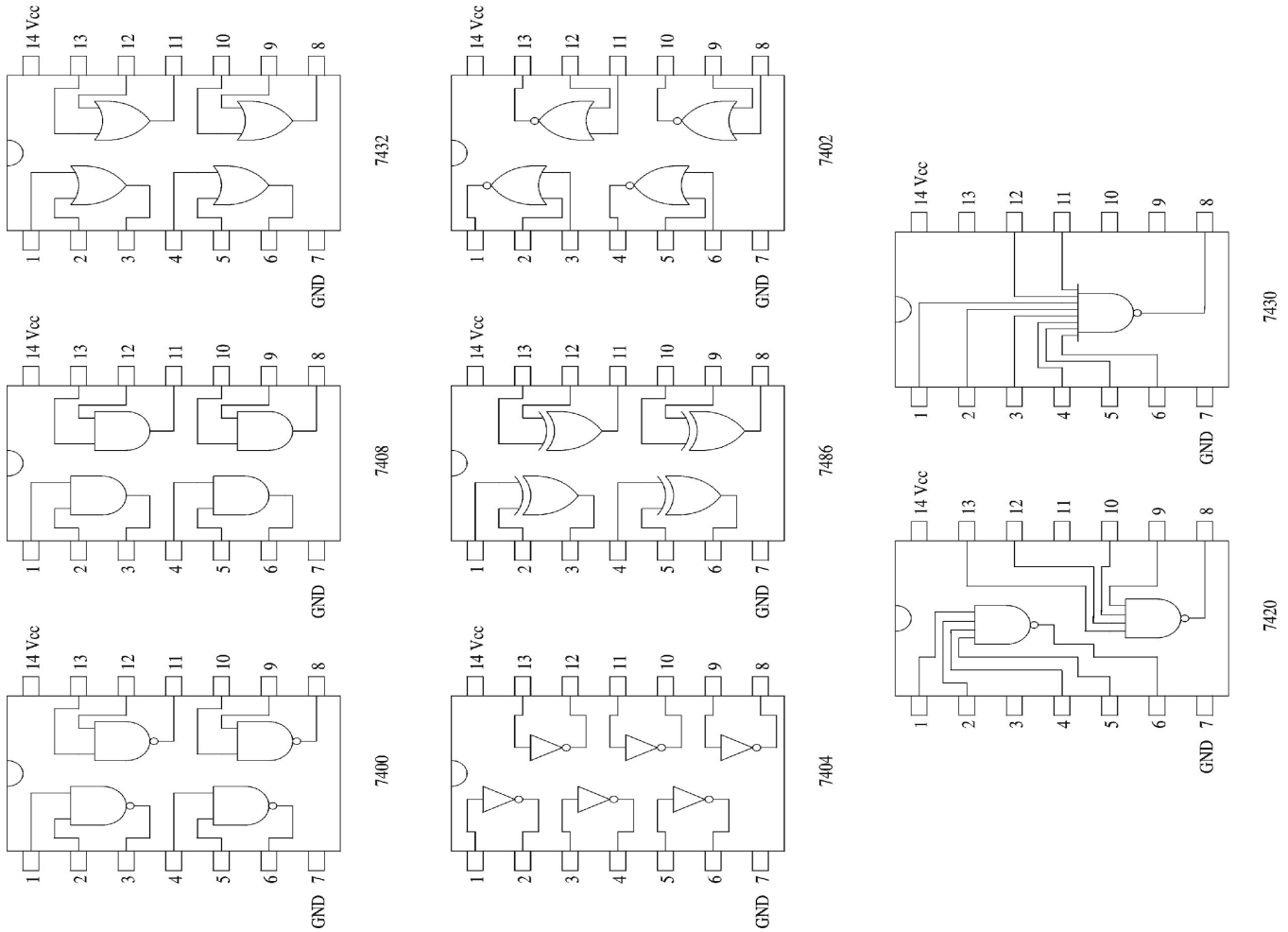
NOR

Logic Chips (cont'd)

- Low voltage level: < 0.4V
- High voltage level: > 2.4V
- Positive logic:
  - Low voltage represents 0
  - High voltage represents 1
- Negative logic:
  - High voltage represents 0
  - Low voltage represents 1
- Propagation delay
  - Delay from input to output
  - Typical value: 5-10 ns



Logic Chips (cont'd)



Logic Chips (cont'd)

- Integration levels
  - SSI (small scale integration)
    - Introduced in late 1960s
    - 1-10 gates (previous examples)
  - MSI (medium scale integration)
    - Introduced in late 1960s
    - 10-100 gates
  - LSI (large scale integration)
    - Introduced in early 1970s
    - 100-10,000 gates
  - VLSI (very large scale integration)
    - Introduced in late 1970s
    - More than 10,000 gates

Logic Functions

- Logical functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical form
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

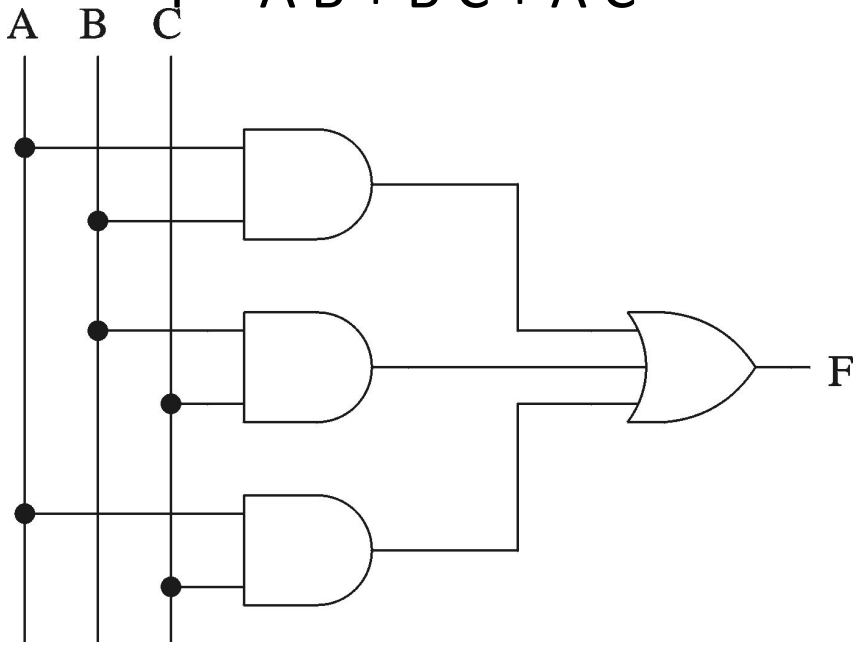
Logic Functions (cont'd)

3-input majority function

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

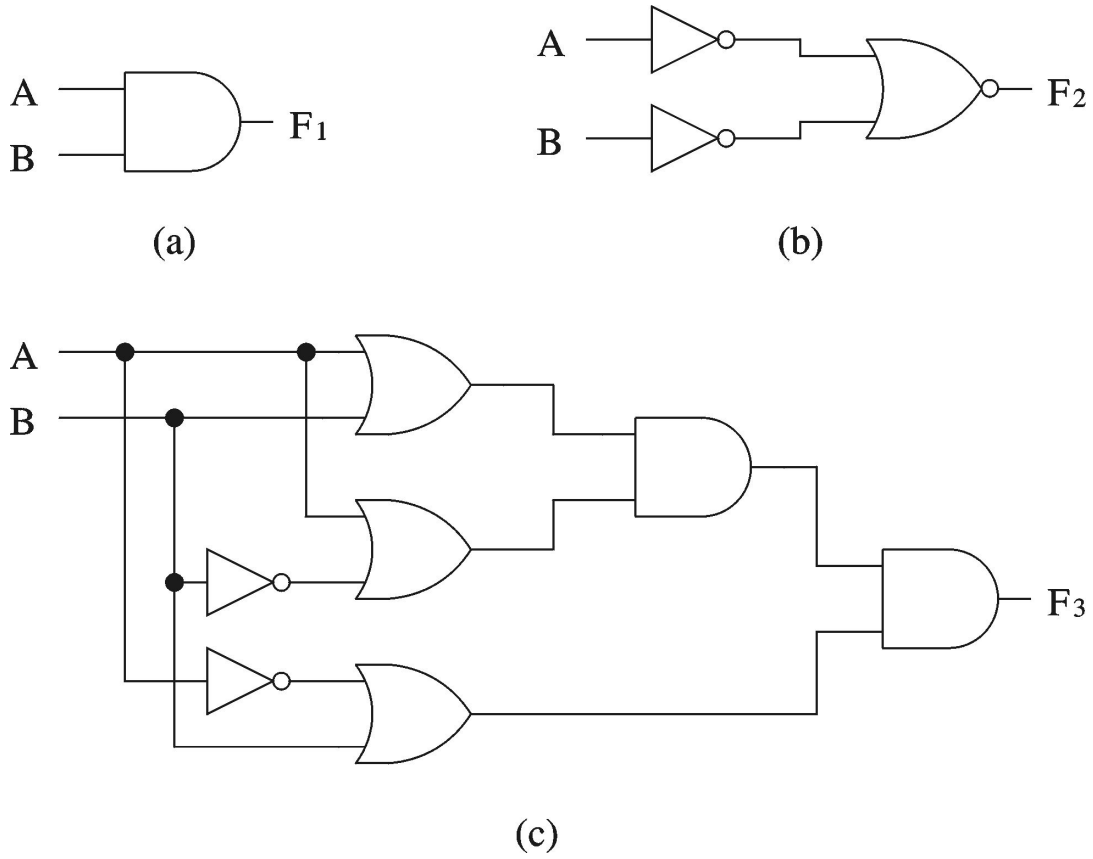
- Logical expression form

$$F = A\,B + B\,C + A\,C$$



Logical Equivalence

- All three circuits implement F = A B function

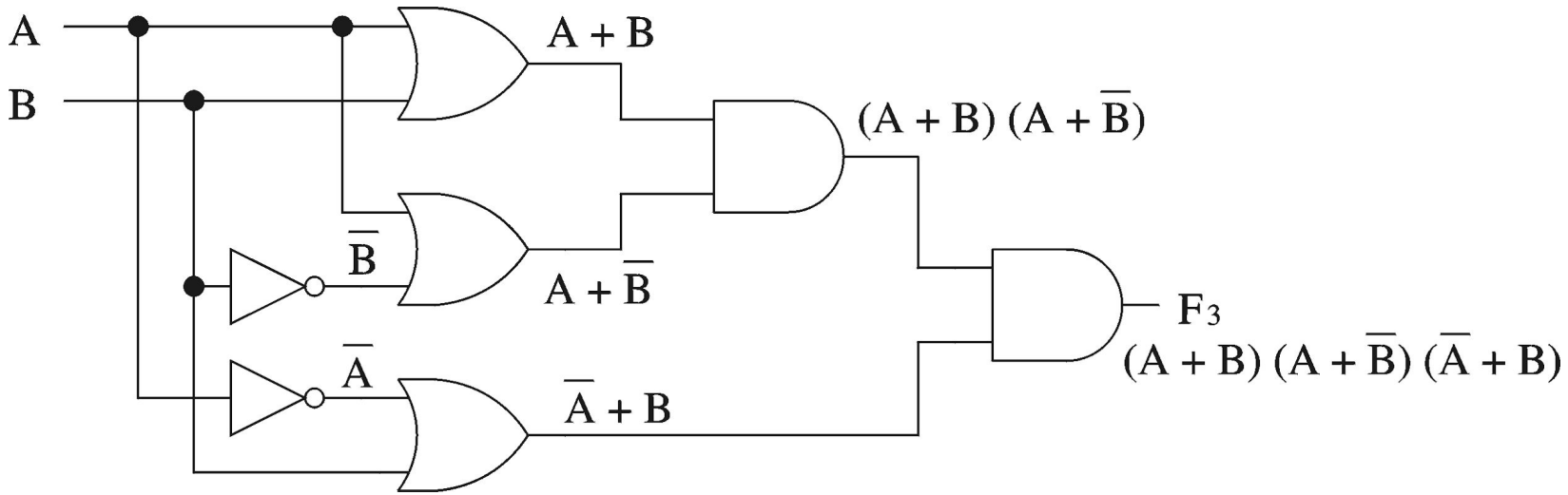


Logical Equivalence (cont'd)

- Proving logical equivalence of two circuits
  - Derive the logical expression for the output of each circuit
  - Show that these two expressions are equivalent
    - Two ways:
      - You can use the truth table method
        - For every combination of inputs, if both expressions yield the same output, they are equivalent
        - Good for logical expressions with small number of variables
      - You can also use algebraic manipulation
        - Need Boolean identities

Logical Equivalence (cont'd)

- Derivation of logical expression from a circuit
  - Trace from the input to output
    - Write down intermediate logical expressions along the path



Logical Equivalence (cont'd)

- Proving logical equivalence: Truth table method

A	B	F1 = A B	F3 = (A + B) (A + B) (A + B)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Thanks for your attention

LECTURE 2

FUNDAMENTALS OF LOGICAL DESIGN

Dana Utebayeva

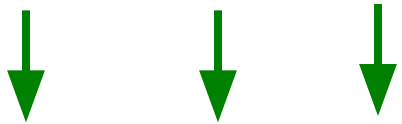
SIS 2
"Binary systems"

d.utebayeva@iitu.edu.kz

DECIMAL TO BINARY CONVERSION

Convert Decimal Number to a Binary Number:

Decimal 7392



Binary 1110011100000

QUIZ for SIS Project 1

Conversion of Decimal Number to a Binary Number:

- Dana Zh. Utebayeva: https://docs.google.com/forms/d/e/1FAIpQLSeKQMn0v_6GEpZ-ROuIxY7bV…
- опубликовано в 13507 Fundamentals of Logic Design (Утебаева Д..) 2021-2022/1 или Общий в Tuesday, September 14, 2021 12:19:40 PM

●

FUNDAMENTALS OF LOGICAL DESIGN

Dana Utebayeva

Lecture 2

"Number systems and Codes"

d.utebayeva@iitu.edu.kz

Goal of the lecture **is to be familiar with number systems and code in digital electronics.**

Outline of Lecture

- Counting in Decimal and Binary
- Place Value
- Binary to Decimal Conversion
- Decimal to Binary Conversion
- Electronic Translators
- Hexadecimal Numbers
- Octal Numbers

Number systems

$$N(b) = \{a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots + a_1 b^1 + a_0 b^0\}$$

Radix and subscript

$$N(b) = a_{n-1}\, a_{n-2} \ldots a_1\, a_0$$

COUNTING IN DECIMAL AND BINARY

Number System -

Code using symbols that refer to a number of items.

Decimal Number System -

Uses ten symbols (base 10 system)

Binary System -

Uses two symbols (base 2 system)

Generalized approach of Number systems

A number with a decimal point is represented by series of coefficients

$$a_5 a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3}$$

PLACE VALUE

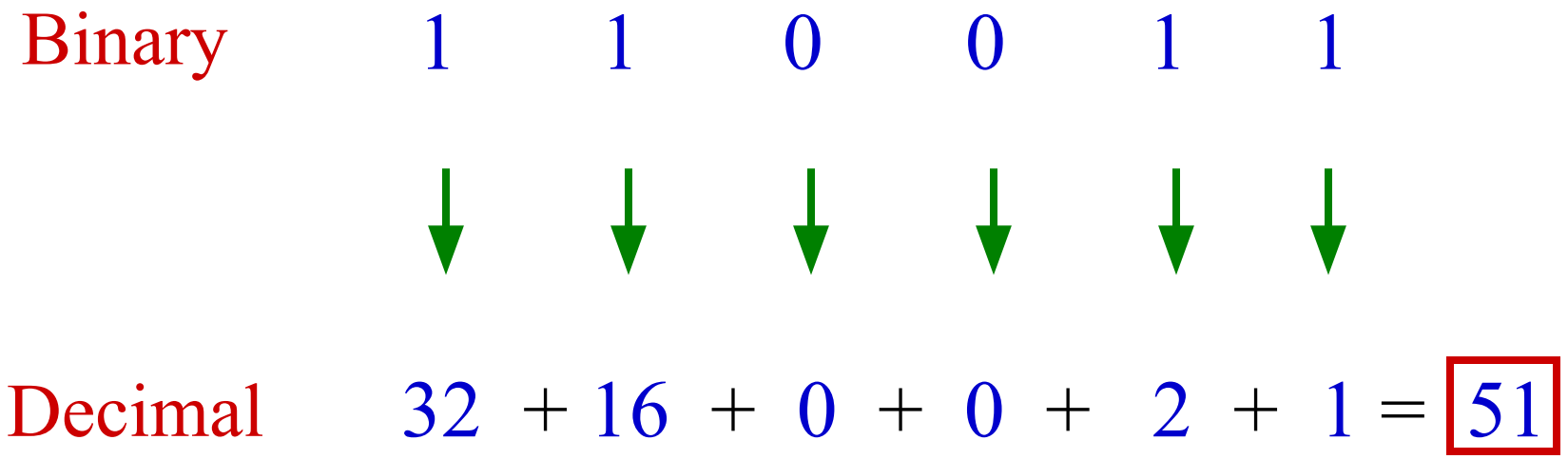
- **Numeric value of symbols in different positions.**
- ***Example* - Place value in binary system:**

Place Value	8s	4s	2s	1s
Binary	Yes	Yes	No	No
Number	1	1	0	0

RESULT: Binary 1100 = decimal 8 + 4 + 0 + 0 = decimal 12

BINARY TO DECIMAL CONVERSION

Convert Binary Number 110011 to a Decimal Number:



TEST

Convert the following binary numbers into decimal numbers:

Binary 1001 = 9

Binary 1111 = 15

Binary 0010 = 2

DECIMAL TO BINARY CONVERSION

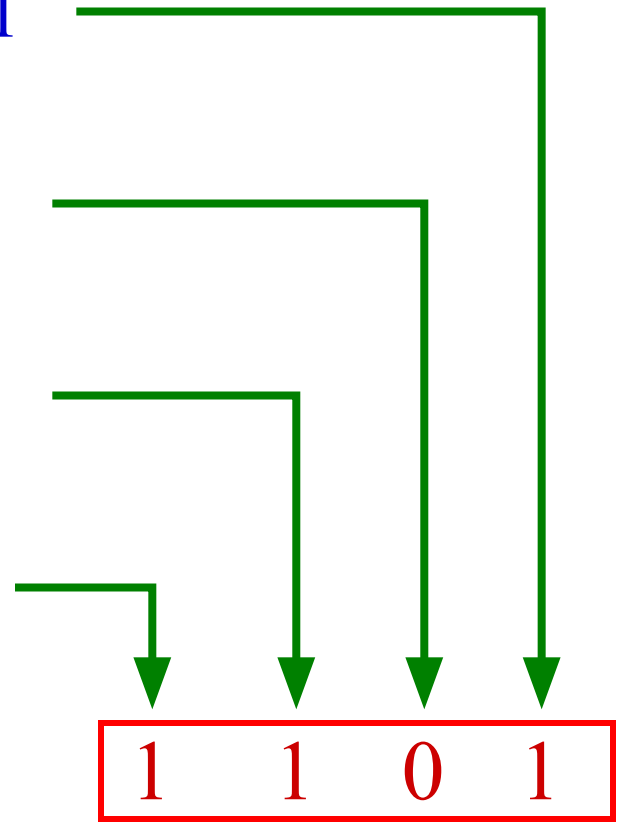
Divide by 2 Process

Decimal # 13 ÷ 2 = 6 remainder 1

6 ÷ 2 = 3 remainder 0

3 ÷ 2 = 1 remainder 1

1 ÷ 2 = 0 remainder 1



TEST

Convert the following decimal numbers into binary:

Decimal 11 = 1011

Decimal 4 = 0100

Decimal 17 = 10001

ELECTRONIC TRANSLATORS

Devices that convert from decimal to binary numbers and from binary to decimal numbers.

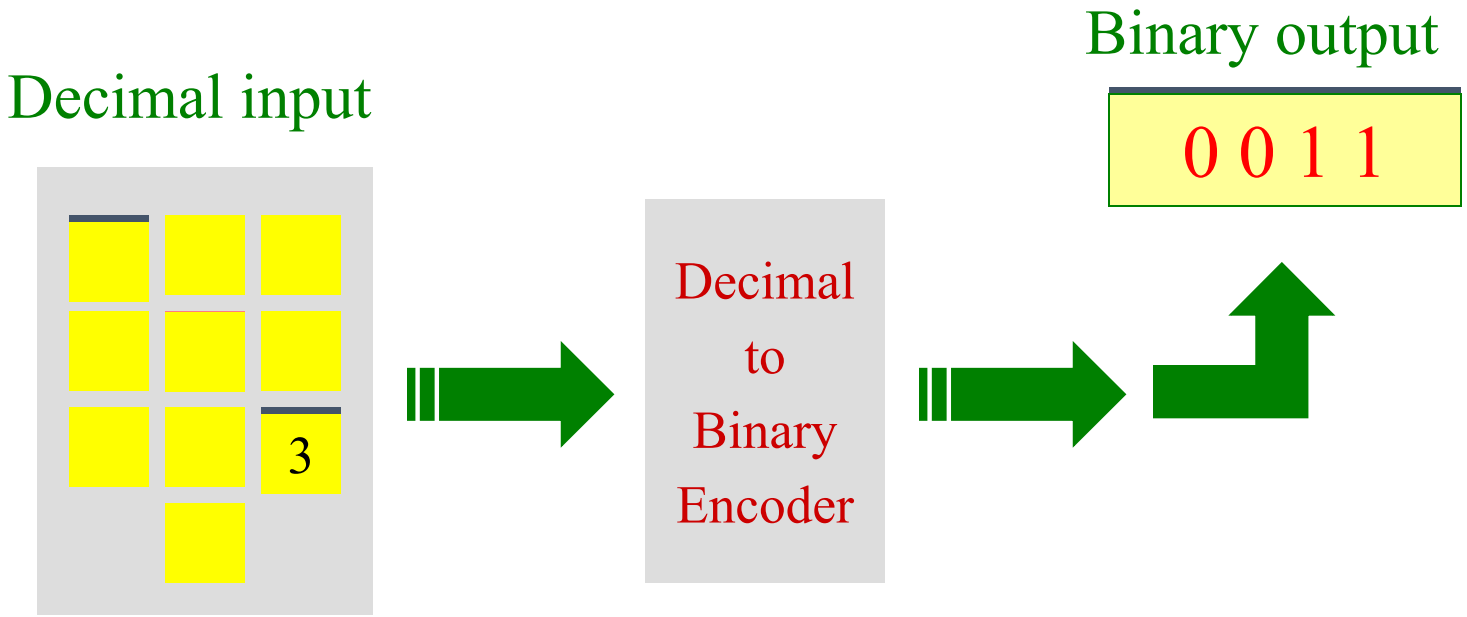
Encoders -

translates from decimal to binary

Decoders -

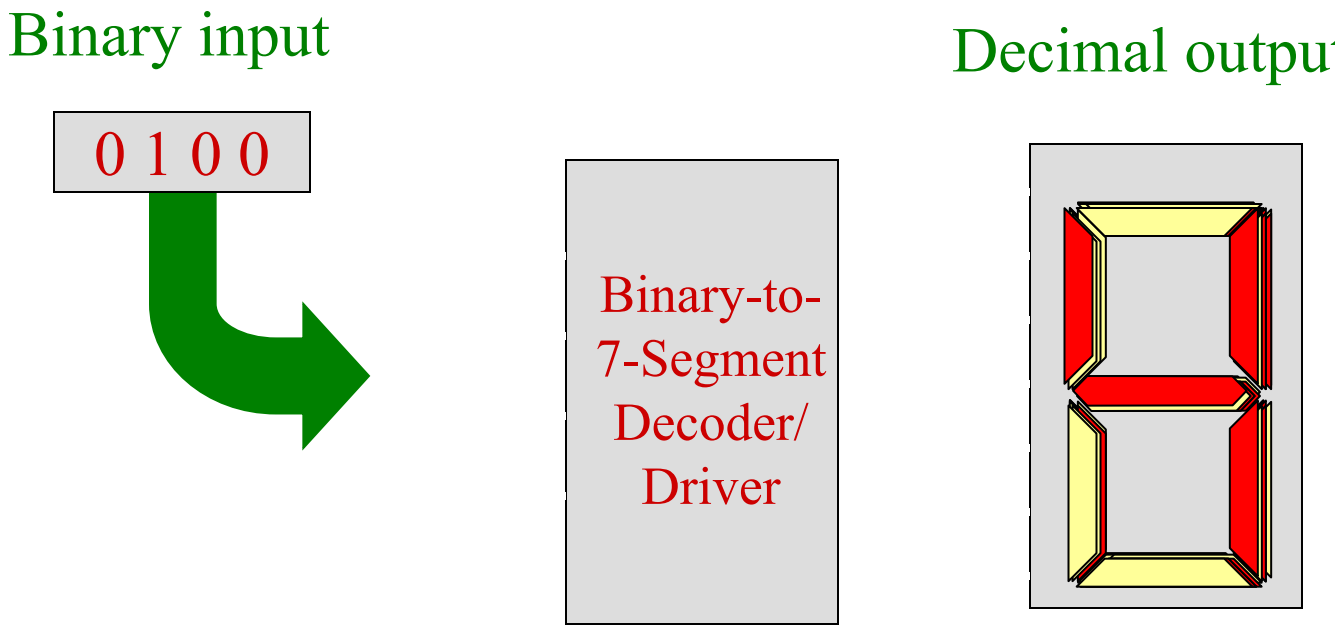
translates from binary to decimal

ELECTRONIC ENCODER - DECIMAL TO BINARY



- Encoders are available in IC form.
- This encoder translates from decimal input to binary (BCD) output.

ELECTRONIC DECODING: BINARY TO DECIMAL



- Electronic decoders are available in IC form.
- This decoder translates from binary to decimal.
- Decimals are shown on an 7-segment LED display.
- This decoder also drives the 7-segment display.

HEXADECIMAL NUMBER SYSTEM

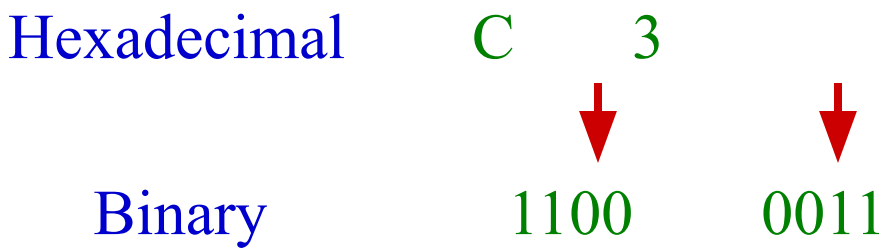
Uses 16 symbols -Base 16 System

0-9, A, B, C, D, E, F

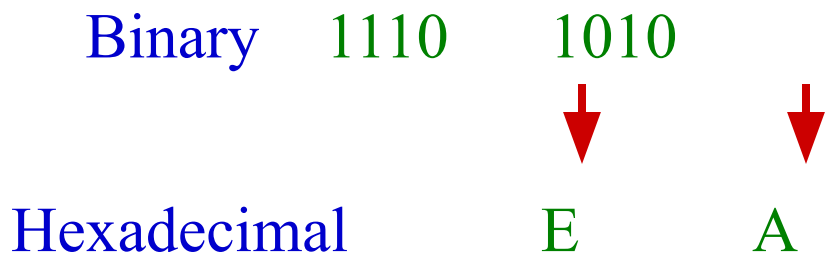
Decimal	Binary	Hexadecimal
1	0001	1
9	1001	9
10	1010	A
15	1111	F
16	10000	10

HEXADECIMAL AND BINARY CONVERSIONS

- Hexadecimal to Binary Conversion



- Binary to Hexadecimal Conversion

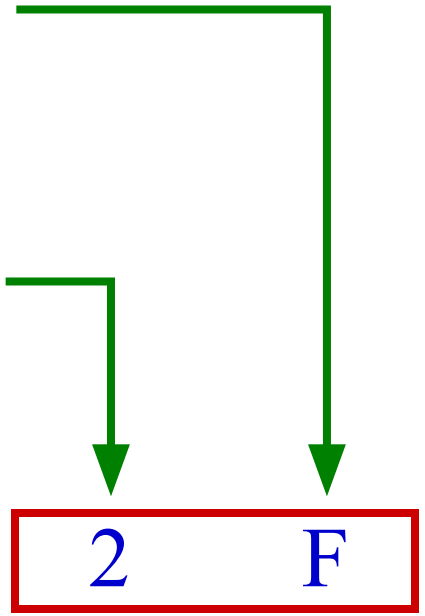


DECIMAL TO HEXADECIMAL CONVERSION

Divide by 16 Process

Decimal # 47 ÷ 16 = 2 remainder 15

2 ÷ 16 = 0 remainder 2



HEXADECIMAL TO DECIMAL CONVERSION

Convert hexadecimal number 2DB to a decimal number

Place	256s	16s	1s
Value Hexadecimal	2	D	B
	(256 x 2)	(16 x 13)	(1 x 11)
Decimal	512 +	208 +	11 = 731

TEST

Convert Hexadecimal number A6 to Binary

A6 = 1010 0110 (Binary)

Convert Hexadecimal number 16 to Decimal

16 = 22 (Decimal)

Convert Decimal 63 to Hexadecimal

63 = 3F (Hexadecimal)

OCTAL NUMBERS

Uses 8 symbols -Base 8 System

0, 1, 2, 3, 4, 5, 6, 7

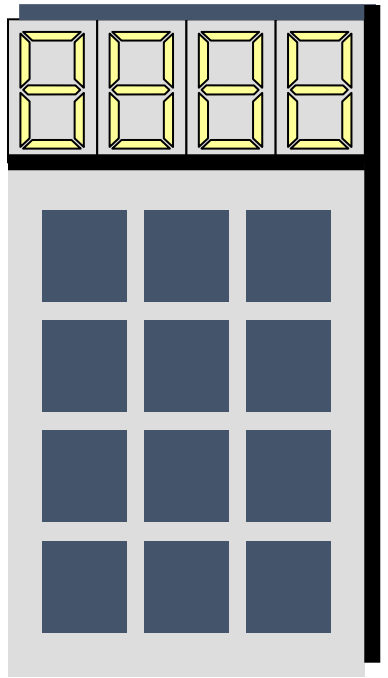
Decimal	Binary	Octal
1	001	1
6	110	6
7	111	7
8	001 000	10
9	001 001	11

PRACTICAL SUGGESTION ON NUMBER SYSTEM CONVERSIONS

- Use a scientific calculator

- Most scientific calculators have DEC, BIN, OCT, and HEX modes and can either convert between codes or perform arithmetic in different number systems.

- Most scientific calculators also have other functions that are valuable in digital electronics such as AND, OR, NOT, XOR, and XNOR logic functions.



SIS project 1

- 1) first page (Names, Title: “Binary systems”)
- 2) Outline (План)
- 3) Part I: (images from your “конспект”)
- 4) Part II: Upload/insert your screens from your Quiz
- 5) Part: Assignments: screens from your copybook
- 6) About calculators

Deadline: Monday till 18.oo (20.09.2021)

Attendance for Lecture 2

- Dana Zh. Utebayeva: https://docs.google.com/forms/d/e/1FAIpQLSfotGeOUvqjykd78SYCT_WEU…
- опубликовано в 13507 Fundamentals of Logic Design (Утебаева Д..) 2021-2022/1 или Общий в Tuesday, September 14, 2021 1:46:25 PM

LECTURE 3-4

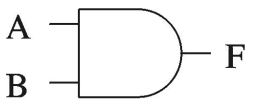
Outline

- Binary numbers
  - Logic States
- Implementation
  - The Buffer Logic Gate using n-p-n transistors
  - Logic Gates using transistors
- Logic functions
  - Expressing logic functions
  - Building block diagrams
- Boolean algebra
  - Boolean algebra laws
- Logic Circuit Design Process
- Deriving logical expressions
  - Sum-of-products form
  - Product-of-sums form
- Generalized gates
- Multiple outputs
- Implementation using other gates (NAND and XOR)

POS – product of sums

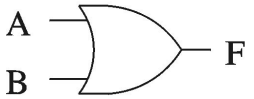
Logic Gates

- Simple gates
  - AND
  - OR
  - NOT
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination
- Other methods
  - Logic expressions
  - Logic diagrams



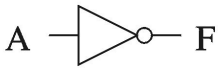
AND gate

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



OR gate

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



NOT gate

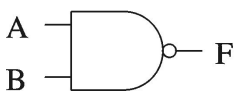
A	F
0	1
1	0

Logic symbol

Truth table

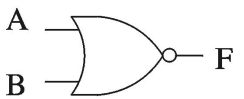
Basic Concepts (cont'd)

- Additional useful gates
  - NAND
  - NOR
  - XOR
- NAND = AND + NOT
- NOR = OR + NOT
- XOR implements exclusive-OR function
- NAND and NOR gates require only 2 transistors
  - AND and OR need 3 transistors!



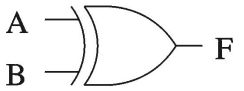
NAND gate

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



NOR gate

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



XOR gate

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Logic symbol

Truth table

Basic Concepts (cont'd)

- Number of functions
  - With N logical variables, we can define
    2^{2N} functions
  - Some of them are useful
    - AND, NAND, NOR, XOR, …
  - Some are not useful:
    - Output is always 1
    - Output is always 0
  - "Number of functions" definition is useful in proving completeness property

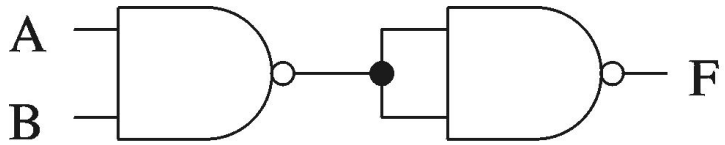
Basic Concepts (cont'd)

- Complete sets
  - A set of gates is complete
    - if we can implement any logical function using only the type of gates in the set
      - You can uses as many gates as you want
  - Some example complete sets
    - {AND, OR, NOT}
    - {AND, NOT}
    - {OR, NOT}
    - {NAND}
    - {NOR}

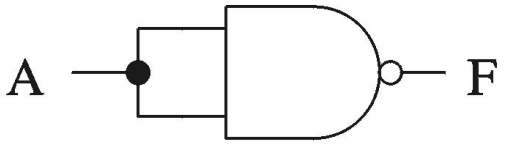
    Not a ~~minimal~~ complete set
  - Minimal complete set
      - A complete set with no redundant elements.

Basic Concepts (cont'd)

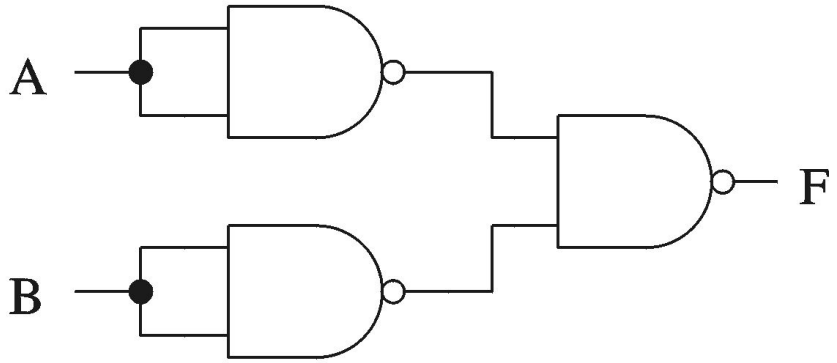
- Proving NAND gate is universal



AND gate



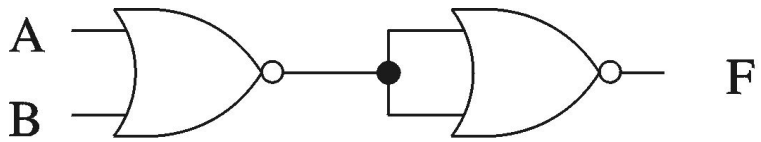
NOT gate



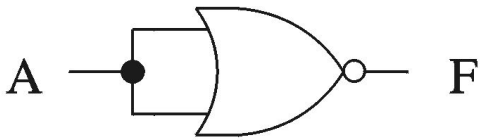
OR gate

Basic Concepts (cont'd)

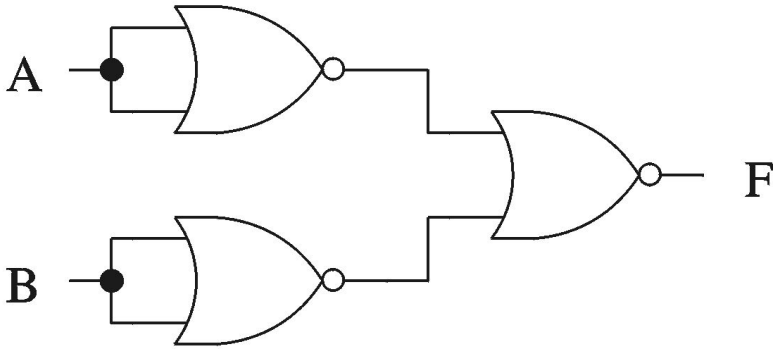
- Proving NOR gate is universal



OR gate



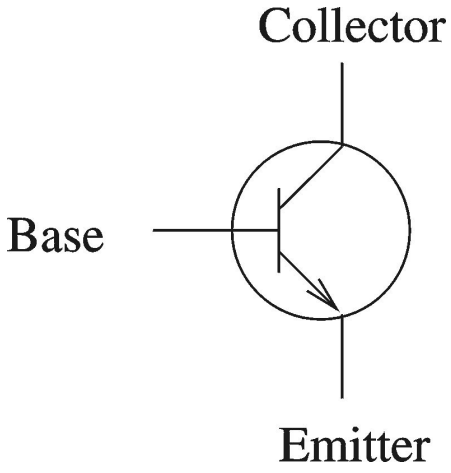
NOT gate



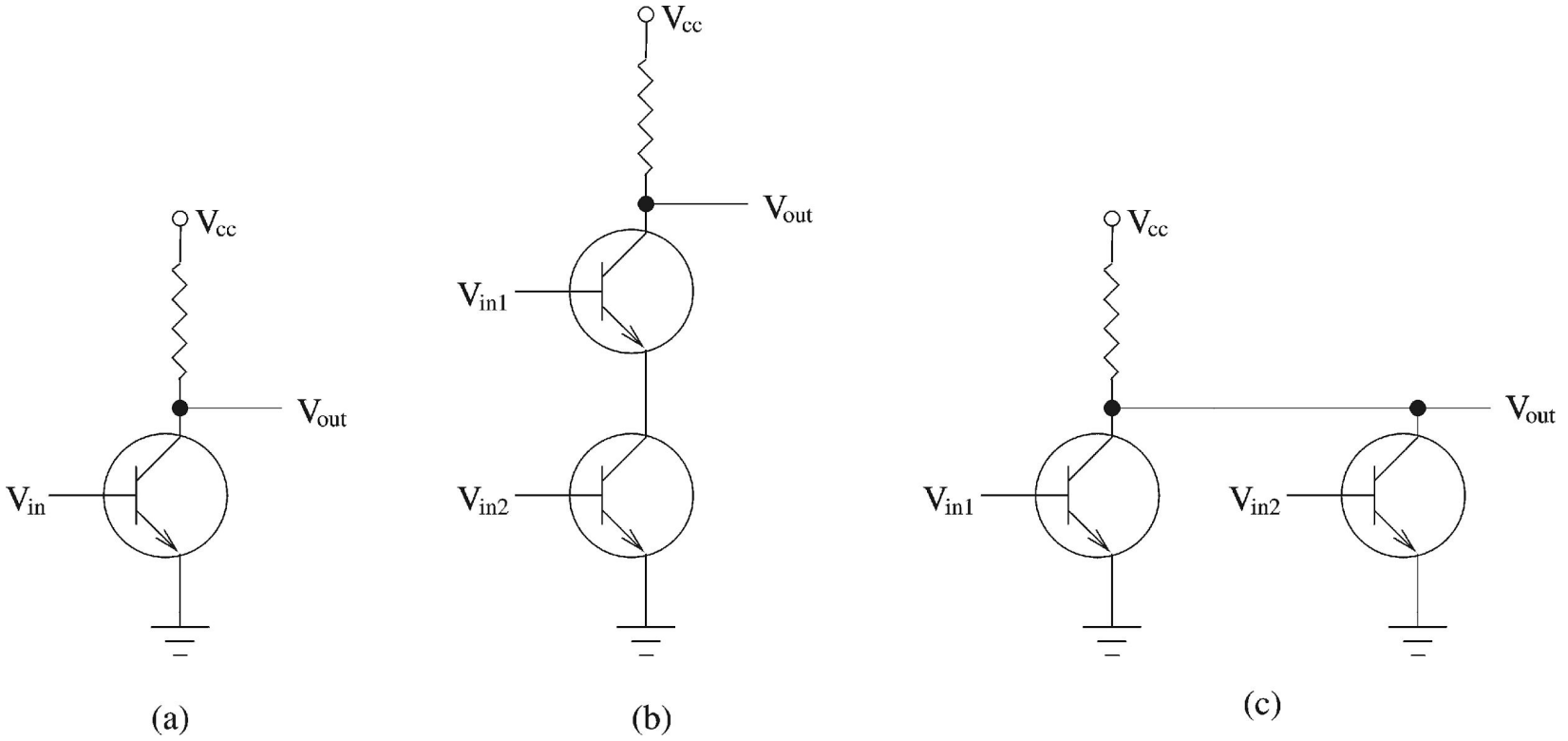
AND gate

Logic Chips

- Basic building block:
  - Transistor
- Three connection points
  - Base
  - Emitter
  - Collector
- Transistor can operate
  - Linear mode
    - Used in amplifiers
  - Switching mode
    - Used to implement digital circuits



Logic Chips (cont'd)



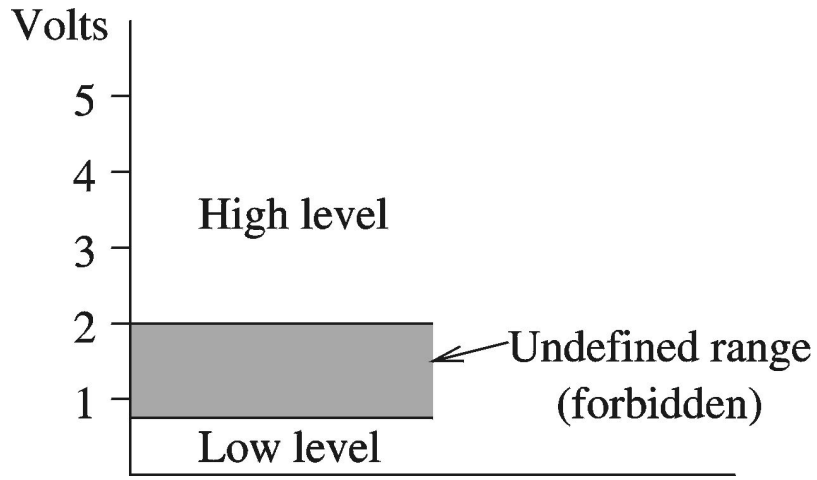
NOT

NAND

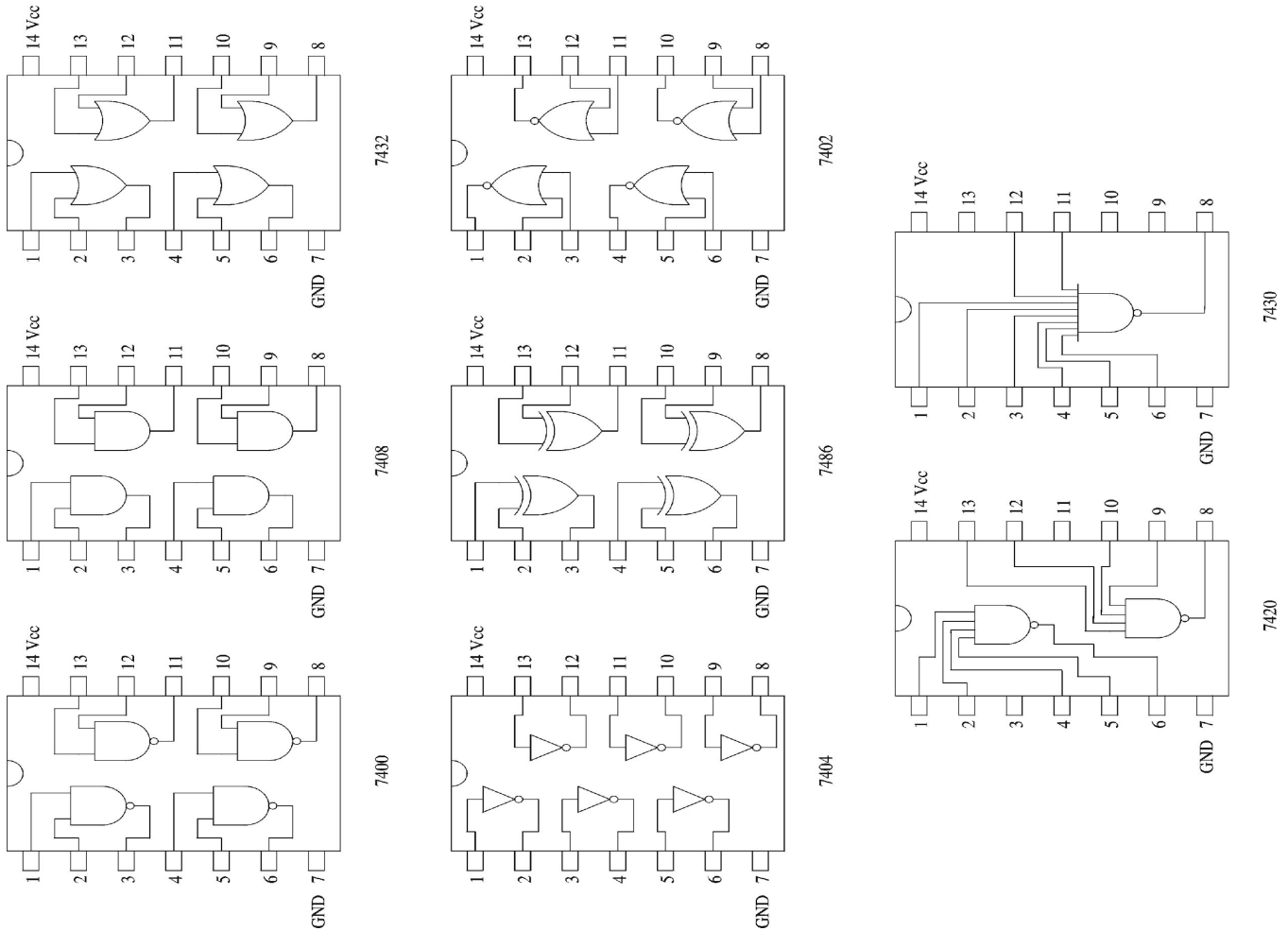
NOR

Logic Chips (cont'd)

- Low voltage level: < 0.4V
- High voltage level: > 2.4V
- Positive logic:
  - Low voltage represents 0
  - High voltage represents 1
- Negative logic:
  - High voltage represents 0
  - Low voltage represents 1
- Propagation delay
  - Delay from input to output
  - Typical value: 5-10 ns



Logic Chips (cont'd)



Logic Chips (cont'd)

- Integration levels
  - SSI (small scale integration)
    - Introduced in late 1960s
    - 1-10 gates (previous examples)
  - MSI (medium scale integration)
    - Introduced in late 1960s
    - 10-100 gates
  - LSI (large scale integration)
    - Introduced in early 1970s
    - 100-10,000 gates
  - VLSI (very large scale integration)
    - Introduced in late 1970s
    - More than 10,000 gates

Logic Functions

- Logical functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical form
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

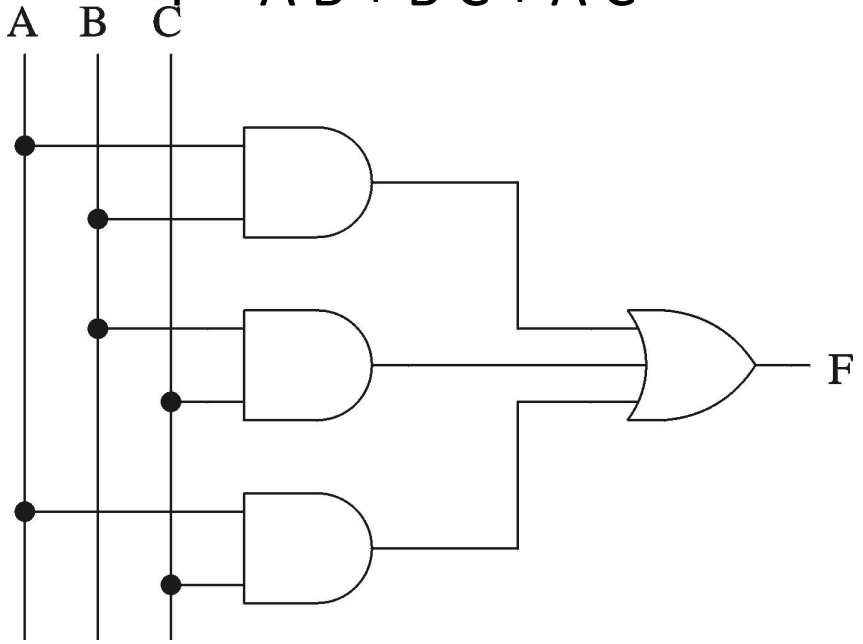
Logic Functions (cont'd)

3-input majority function

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

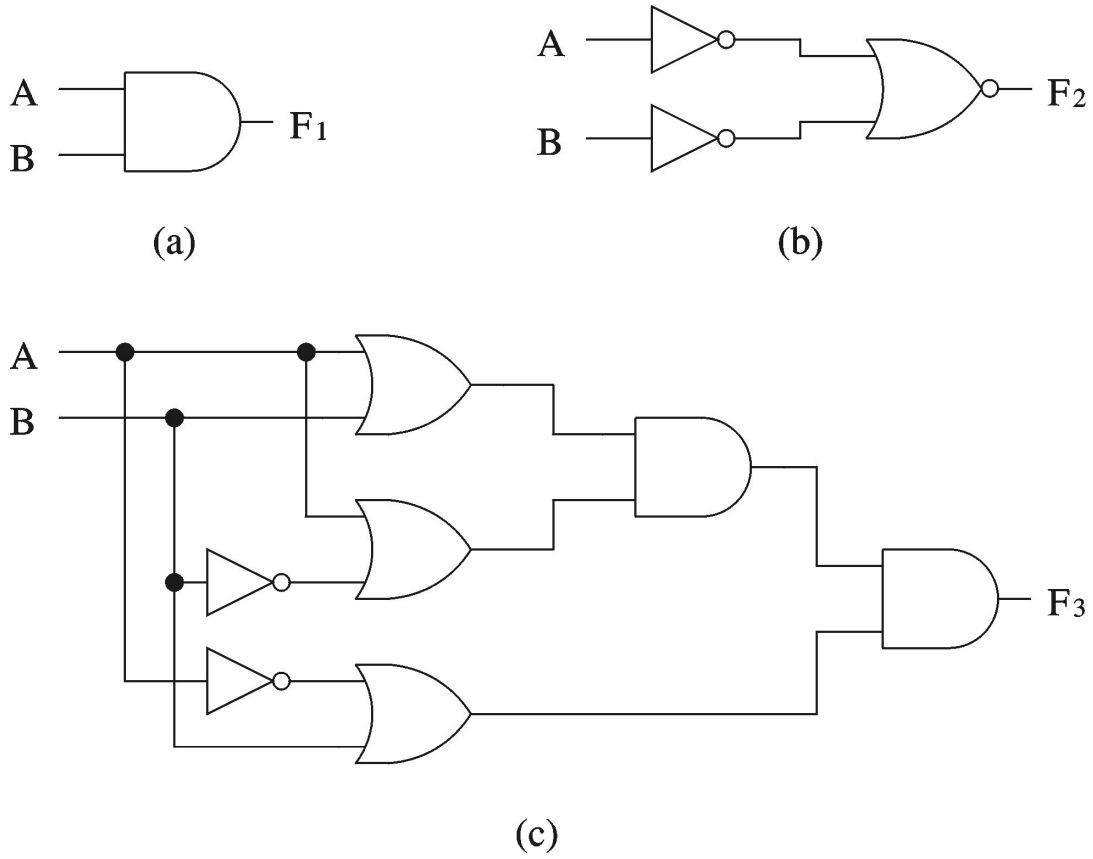
- Logical expression form

$$F = A\,B + B\,C + A\,C$$



Logical Equivalence

- All three circuits implement F = A B function

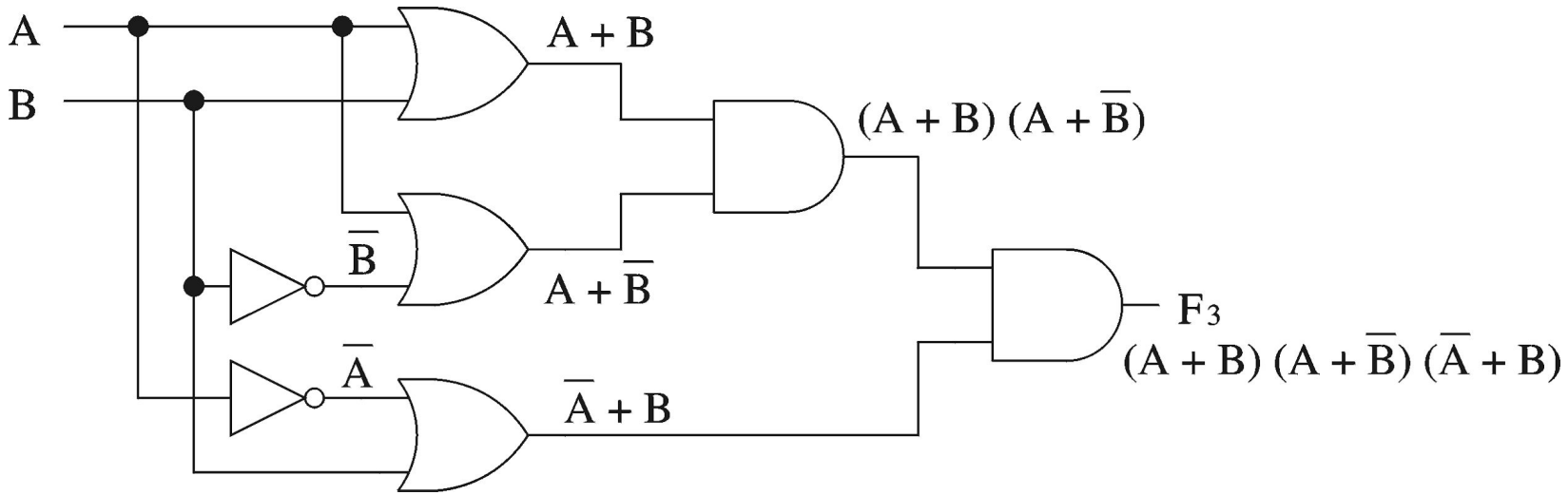


Logical Equivalence (cont'd)

- Proving logical equivalence of two circuits
  - Derive the logical expression for the output of each circuit
  - Show that these two expressions are equivalent
    - Two ways:
      - You can use the truth table method
        - For every combination of inputs, if both expressions yield the same output, they are equivalent
        - Good for logical expressions with small number of variables
      - You can also use algebraic manipulation
        - Need Boolean identities

Logical Equivalence (cont'd)

- Derivation of logical expression from a circuit
  - Trace from the input to output
    - Write down intermediate logical expressions along the path



Logical Equivalence (cont'd)

- Proving logical equivalence: Truth table method

A	B	F1 = A B	F3 = (A + B) (A + B) (A + B)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Thanks for your attention

LECTURE 4

#3 Boolean Algebra and Digital Logic Gates.

#4 Combinational logic design. Completely and Incompletely Specified Logic Functions. Design of a combinational Circuits.

Lecture 3 - 4

Dana Utebayeva

Outline

- Binary numbers
  - Logic States
- Implementation
  - The Buffer Logic Gate using n-p-n transistors
  - Logic Gates using transistors
- Logic functions
  - Expressing logic functions
  - Building block diagrams
- Boolean algebra
  - Boolean algebra laws
- Logic Circuit Design Process
- Deriving logical expressions
  - Sum-of-products form
  - Product-of-sums form
- Generalized gates
- Multiple outputs
- Implementation using other gates (NAND and XOR)

Basic Concepts (cont'd)

- Number of functions
  - With N logical variables, we can define
    2^{2N} functions
  - Some of them are useful
    - AND, NAND, NOR, XOR, …
  - Some are not useful:
    - Output is always 1
    - Output is always 0
  - "Number of functions" definition is useful in proving completeness property

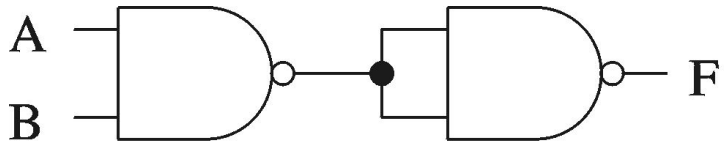
Basic Concepts (cont'd)

- Complete sets
  - A set of gates is complete
    - if we can implement any logical function using only the type of gates in the set
      - You can uses as many gates as you want
  - Some example complete sets
    - {AND, OR, NOT}
    - {AND, NOT}
    - {OR, NOT}
    - {NAND}
    - {NOR}

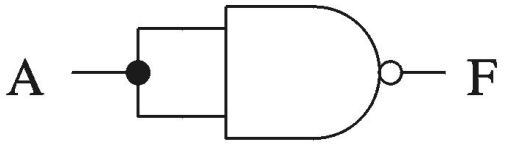
    Not a ~~minimal~~ complete set
  - Minimal complete set
      - A complete set with no redundant elements.

Basic Concepts (cont'd)

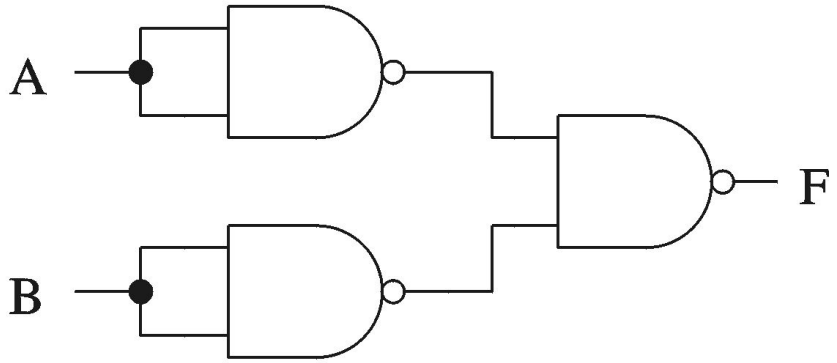
- Proving NAND gate is universal



AND gate



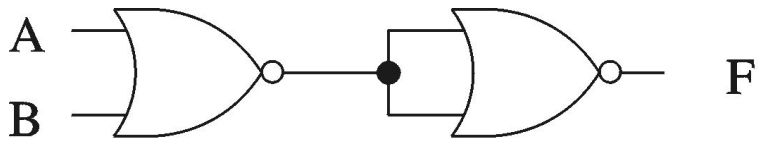
NOT gate



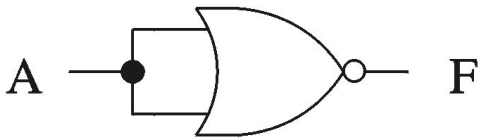
OR gate

Basic Concepts (cont'd)

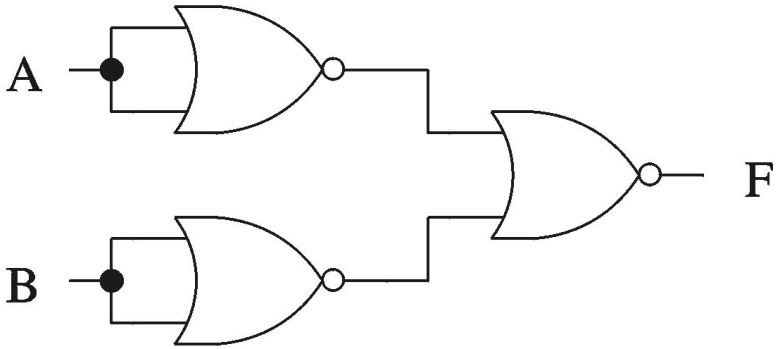
- Proving NOR gate is universal



OR gate



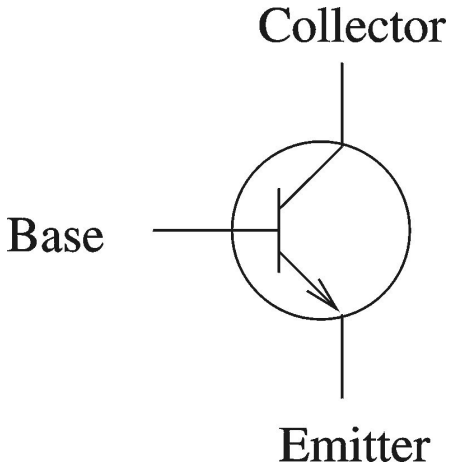
NOT gate



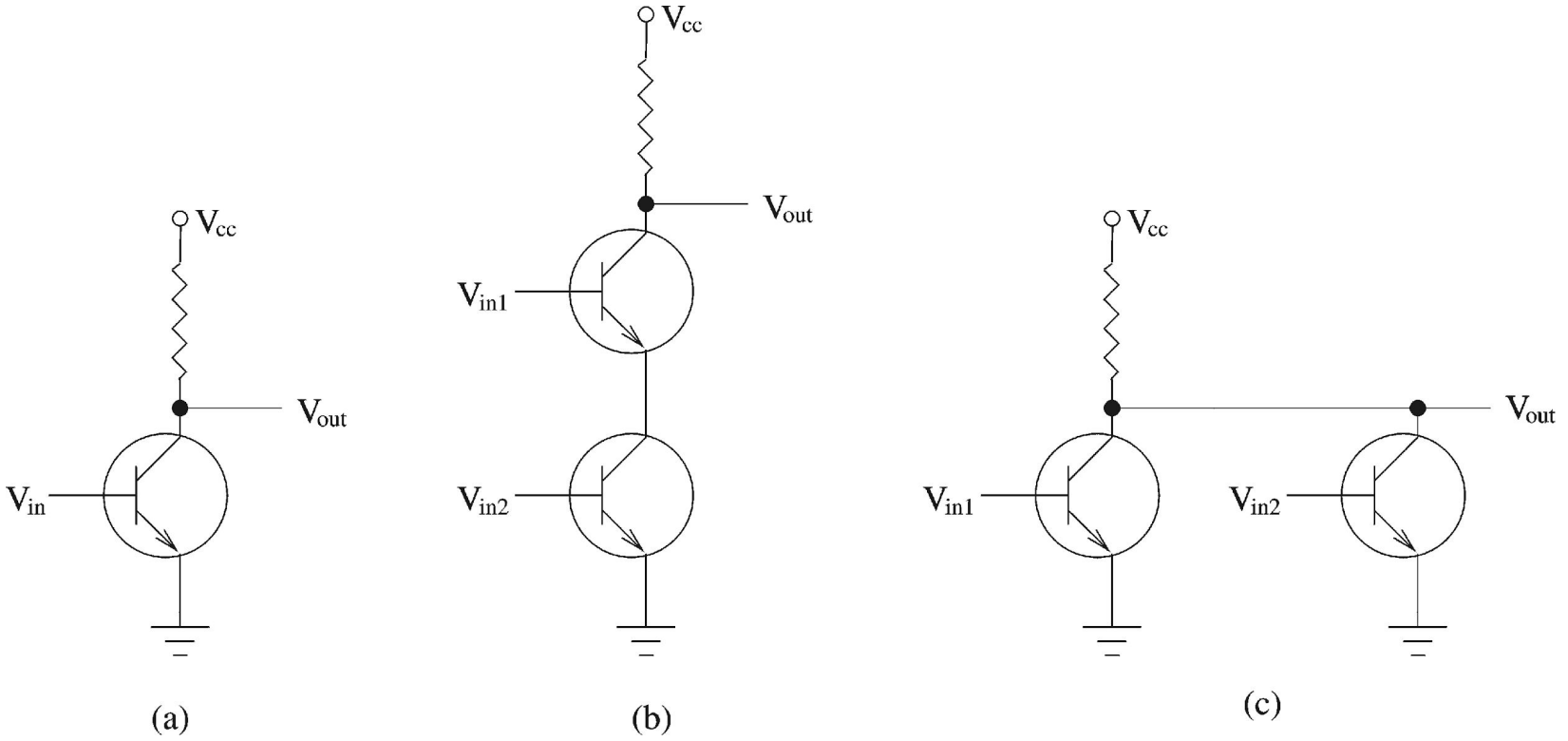
AND gate

Logic Chips

- Basic building block:
  - Transistor
- Three connection points
  - Base
  - Emitter
  - Collector
- Transistor can operate
  - Linear mode
    - Used in amplifiers
  - Switching mode
    - Used to implement digital circuits



Logic Chips (cont'd)



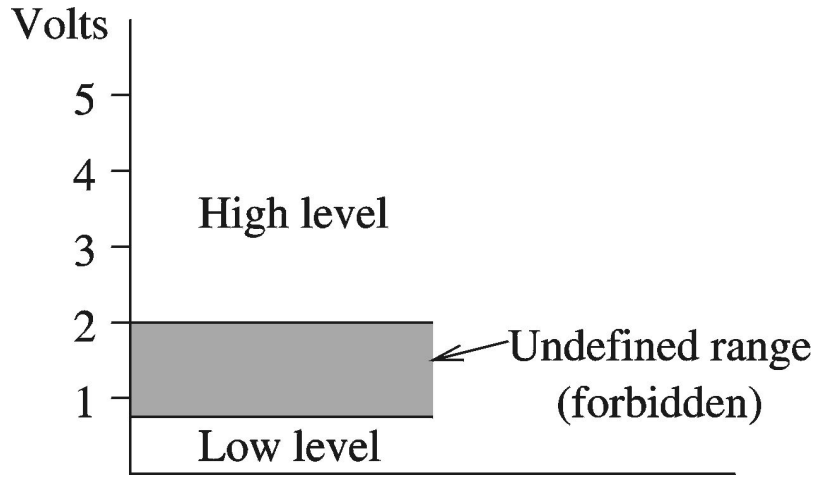
NOT

NAND

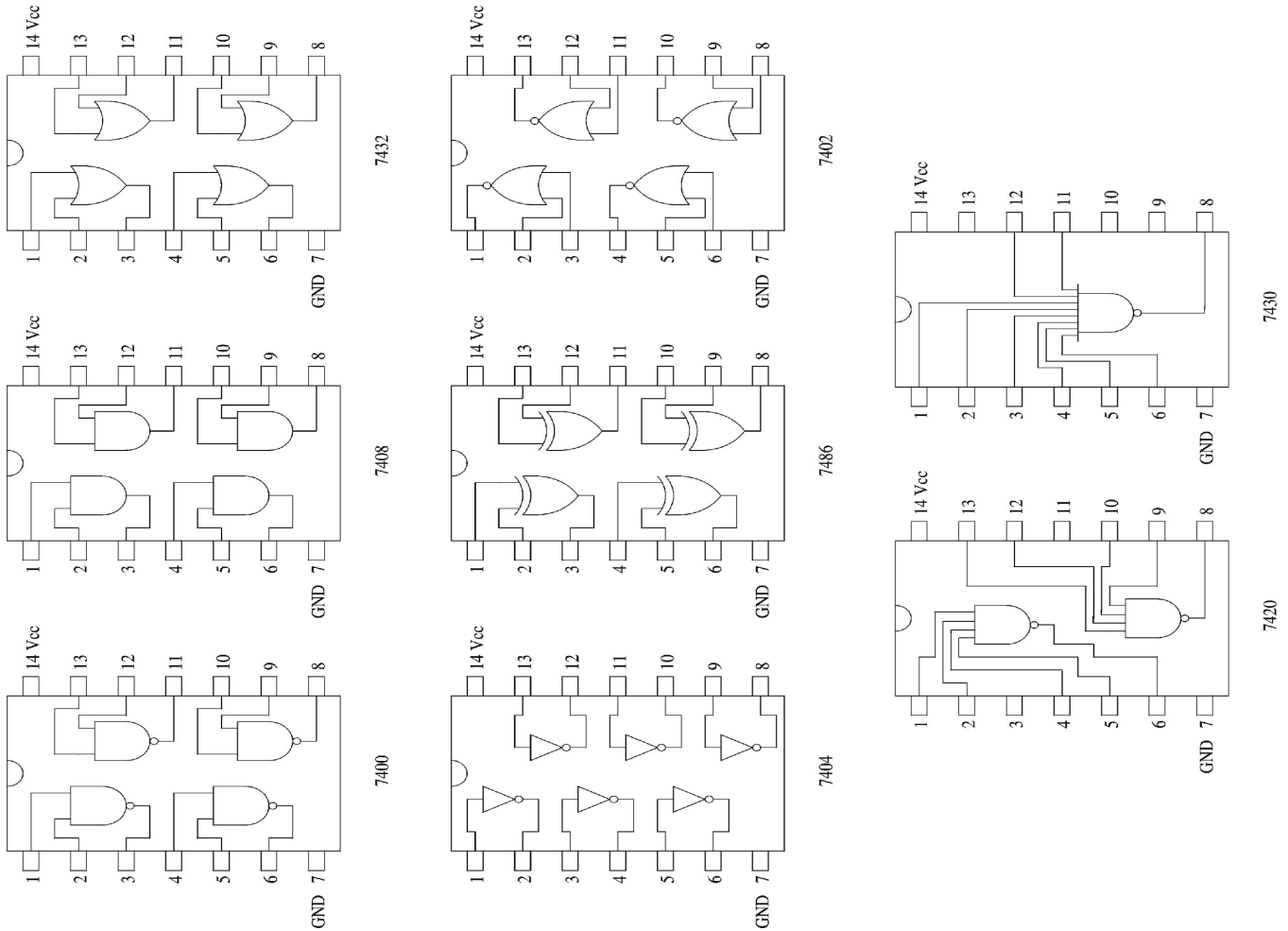
NOR

Logic Chips (cont'd)

- Low voltage level: < 0.4V
- High voltage level: > 2.4V
- Positive logic:
  - Low voltage represents 0
  - High voltage represents 1
- Negative logic:
  - High voltage represents 0
  - Low voltage represents 1
- Propagation delay
  - Delay from input to output
  - Typical value: 5-10 ns



Logic Chips (cont'd)



Logic Chips (cont'd)

- Integration levels
  - SSI (small scale integration)
    - Introduced in late 1960s
    - 1-10 gates (previous examples)
  - MSI (medium scale integration)
    - Introduced in late 1960s
    - 10-100 gates
  - LSI (large scale integration)
    - Introduced in early 1970s
    - 100-10,000 gates
  - VLSI (very large scale integration)
    - Introduced in late 1970s
    - More than 10,000 gates

Logic Functions

- Logical functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical form
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

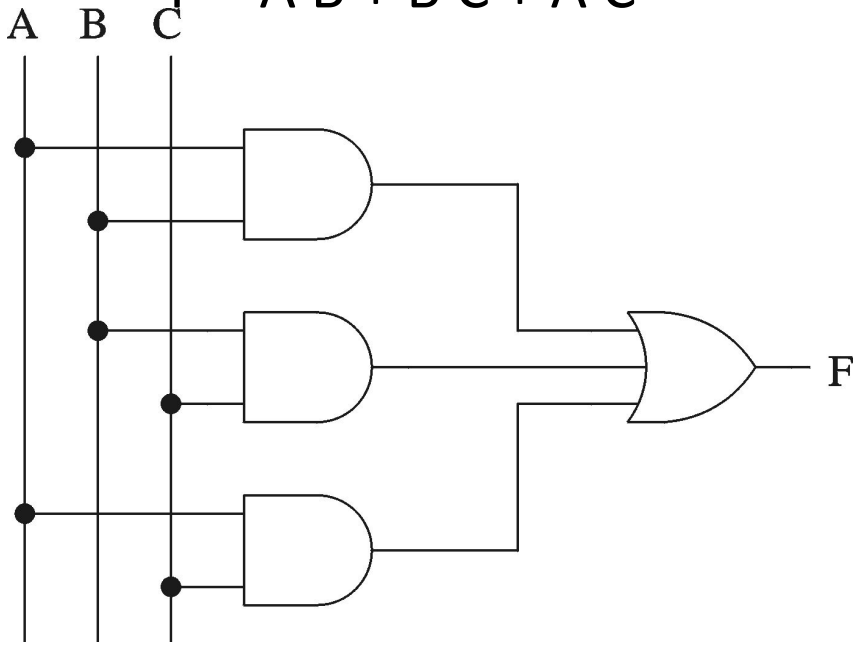
Logic Functions (cont'd)

3-input majority function

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

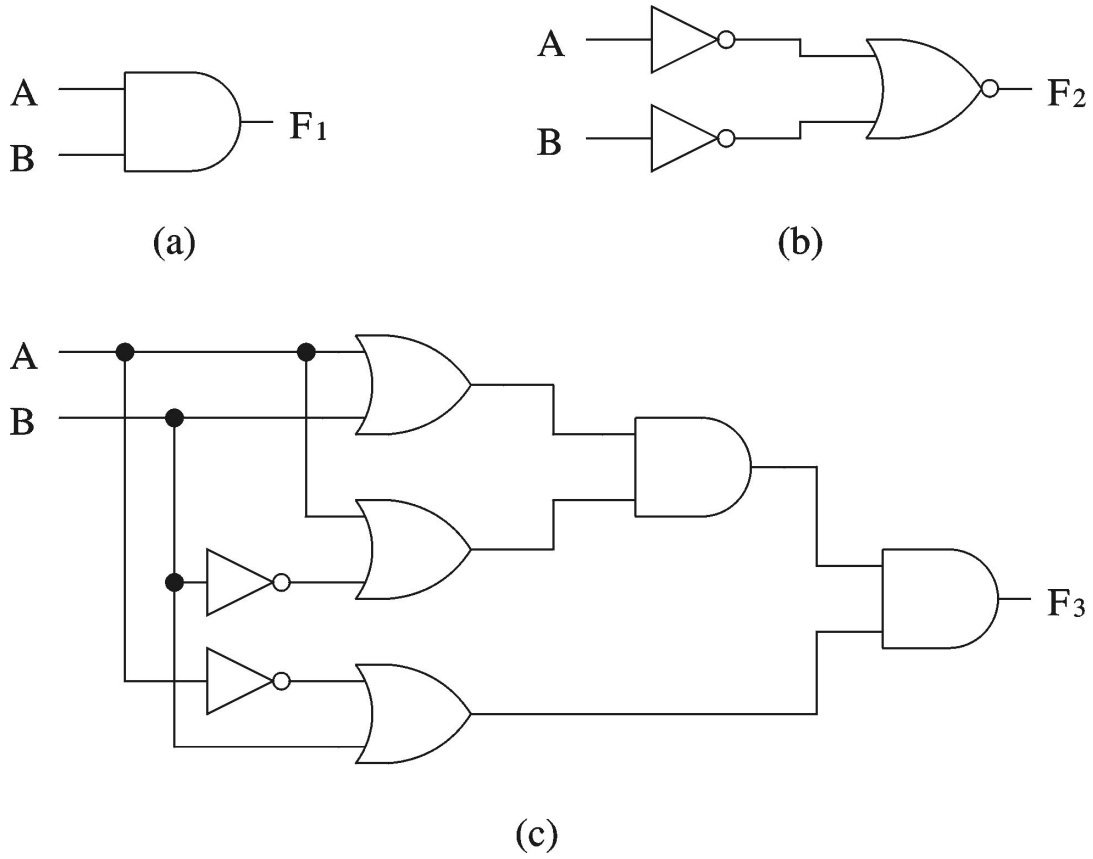
- Logical expression form

$$F = A\,B + B\,C + A\,C$$



Logical Equivalence

- All three circuits implement F = A B function

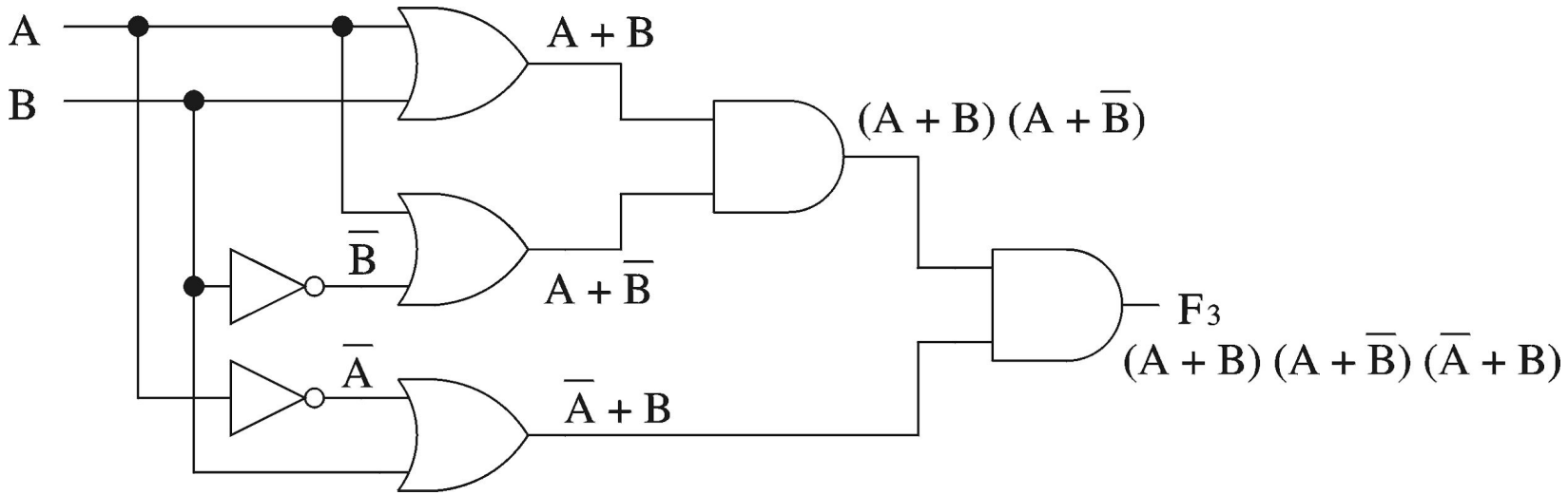


Logical Equivalence (cont'd)

- Proving logical equivalence of two circuits
  - Derive the logical expression for the output of each circuit
  - Show that these two expressions are equivalent
    - Two ways:
      - You can use the truth table method
        - For every combination of inputs, if both expressions yield the same output, they are equivalent
        - Good for logical expressions with small number of variables
      - You can also use algebraic manipulation
        - Need Boolean identities

Logical Equivalence (cont'd)

- Derivation of logical expression from a circuit
  - Trace from the input to output
    - Write down intermediate logical expressions along the path



Logical Equivalence (cont'd)

- Proving logical equivalence: Truth table method

A	B	F1 = A B	F3 = (A + B) (A + B) (A + B)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Thanks for your attention

LECTURE 5

«CPC»: SIS, Practice class and Lab class assignments explanation

#5 Combinational and Sequential Circuit. Adders. Subtractors. Comparators.

Lecture 5

Dana Utebayeva

Outline

- **«CPC» SIS assignments explanation**
  - Practice class
  - Lab class
  - SIS assignments explanation
- Boolean Identities/Postulates/laws
  - Expressing logic functions
  - Building block diagrams
- Simplification Using Boolean Identities

- Standard Representations
  - Minterm
  - Maxterm
- Combinational and Sequential circuits
- Adders
- Comparators
- Full Subtractor

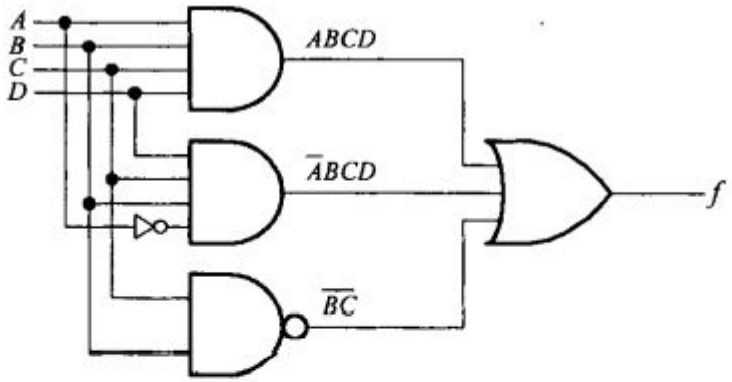
Boolean Identities / Postulates / laws
Boolean Laws

Here is a list of Boolean identities that are useful in simplifying Boolean expressions:

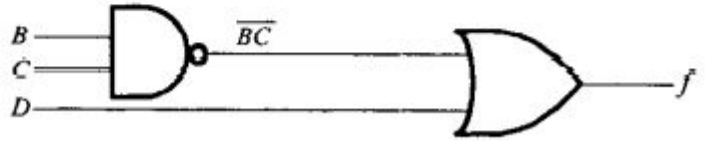
1. a) $A + 0 = A$ b) $A \cdot 1 = A$
2. a) $A + 1 = 1$ b) $A \cdot 0 = 0$
3. a) $A + A = A$ b) $A \cdot A = A$
4. a) $A + \overline{A} = 1$ b) $A \cdot \overline{A} = 0$
5. a) $(\overline{\overline{A}}) = A$
6. Commutative Law:
   a) $A + B = B + A$ b) $A \cdot B = B \cdot A$
7. Associative Law:
   a) $A + (B + C) = (A + B) + C$ b) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
8. Distributive Law:
   a) $A \cdot (B + C) = A \cdot B + A \cdot C$ b) $A + B \cdot C = (A + B) \cdot (A + C)$
9. DeMorgan's Theorem:
   a) $\overline{A + B} = \overline{A} \cdot \overline{B}$ b) $\overline{A \cdot B} = \overline{A} + \overline{B}$

Problem 2

(a) Implementation of f= ABCD + ABCD + BC(b) implementation of the simplified function f= BC + DImplementation of Boolean function using logic gates



(a) Implementation of $f = ABCD + \bar{A}BCD + \overline{BC}$



(b) implementation of the simplified function $f = \overline{BC} + D$

Implementation of Boolean function using logic gates

$$f = ABCD + \overline{A}BCD + \overline{\overline{BC}}$$

Simplification Using Boolean Identities

$$f = ABCD + \overline{A}BCD + \overline{\overline{BC}}$$

f= ABCD + ABCD + BC

Complement of a Boolean Function

$$(A + B + C)' =$$

(A + B + C)

$$
\begin{aligned}
(A + B + C)' &= (A + X)' && \text{let } B + C = X \\
&= A'X' && \text{by theorem 5(a) (DeMorgan)} \\
&= A' \cdot (B + C)' && \text{substitute } B + C = X \\
&= A' \cdot (B'C') && \text{by theorem 5(a) (DeMorgan)} \\
&= A'B'C' && \text{by theorem 4(b) (associative)}
\end{aligned}
$$

Complement of a Boolean Function

$$f = \overline{C}(AB + \overline{A}\,\overline{B}D + \overline{A}B\overline{D})$$

Boolean Algebra and Digital Logic Gates

$$\overline{f} = \overline{\overline{C}(AB + \overline{A}\,\overline{B}D + \overline{A}B\overline{D})}$$

$$= \overline{\overline{C}} + \overline{(AB + \overline{A}\,\overline{B}D + \overline{A}B\overline{D})}$$

$$= C + \left(\overline{AB} \cdot \overline{\overline{A}\,\overline{B}D} \cdot \overline{\overline{A}B\overline{D}}\right)$$

$$= C + (\overline{A} + \overline{B})(A + B + \overline{D})(A + \overline{B} + D)$$

By taking the dual and complementing each literal, we have:

The dual of f: $\overline{C} + (A + B)(\overline{A} + \overline{B} + D)(\overline{A} + B + \overline{D})$

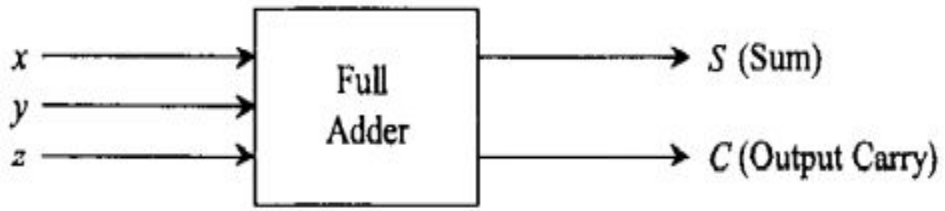
Complementing each literal: $C + (\overline{A} + \overline{B})(A + B + \overline{D})(A + \overline{B} + D) = \overline{f}$

f= C'(AB + A'B'D + A'BD')

f= C(AB + ABD + ABD)

Combinational and Sequential circuits

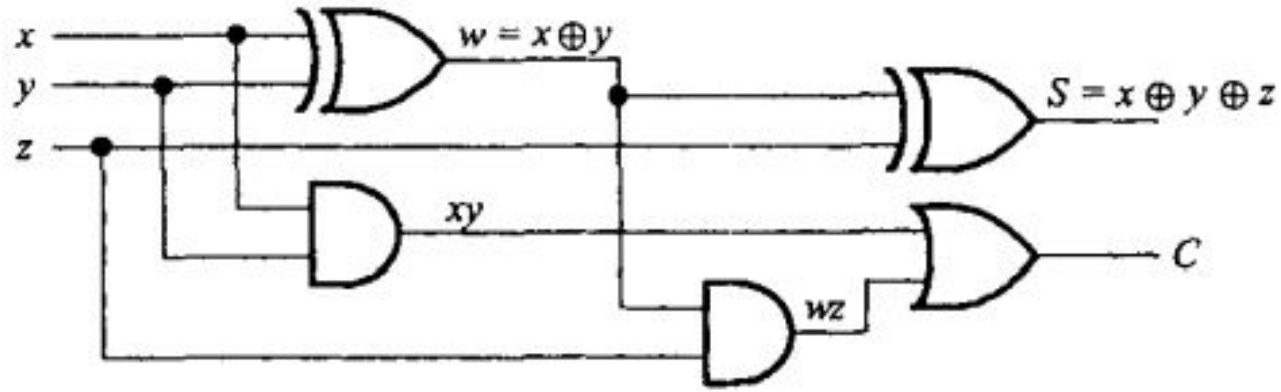
Full Adder



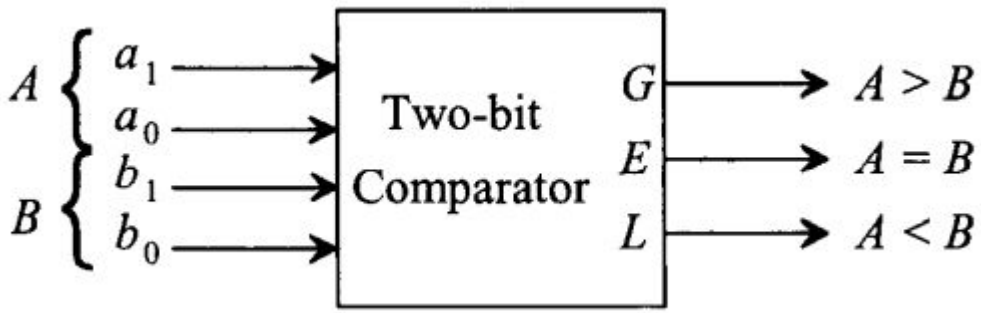
Truth Table of a Full Adder

Inputs			Outputs		Decimal Value
x	y	z	C	S	
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

Logic Diagram of Full Adder



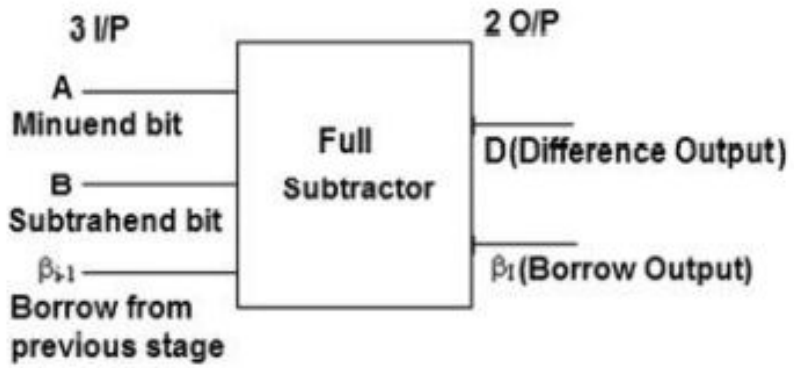
Two-bit Comparator



Truth Table for the 2-Bit Comparator

Inputs				Outputs		
a_1	a_0	b_1	b_0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Full Subtractor



Logic Chips (cont'd)

- Integration levels
  - SSI (small scale integration)
    - Introduced in late 1960s
    - 1-10 gates (previous examples)
  - MSI (medium scale integration)
    - Introduced in late 1960s
    - 10-100 gates
  - LSI (large scale integration)
    - Introduced in early 1970s
    - 100-10,000 gates
  - VLSI (very large scale integration)
    - Introduced in late 1970s
    - More than 10,000 gates

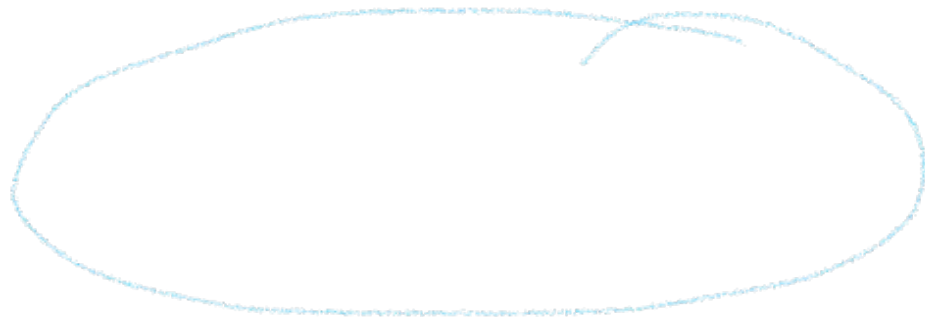
LECTURE 6

Multiplexer design Procedure and applications

Lecture 6

Utebayeva dana


A
B
S



Truth table for MUX

Truth table for 4-to-1 MUX

LECTURE 7

Demultiplexers and their Applications

Lecture 7, By dana Utebayeva

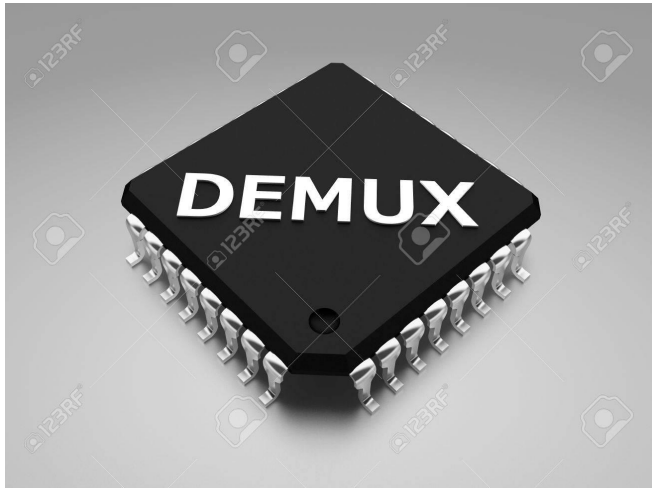
DEMUX DEMULTIPLEXER

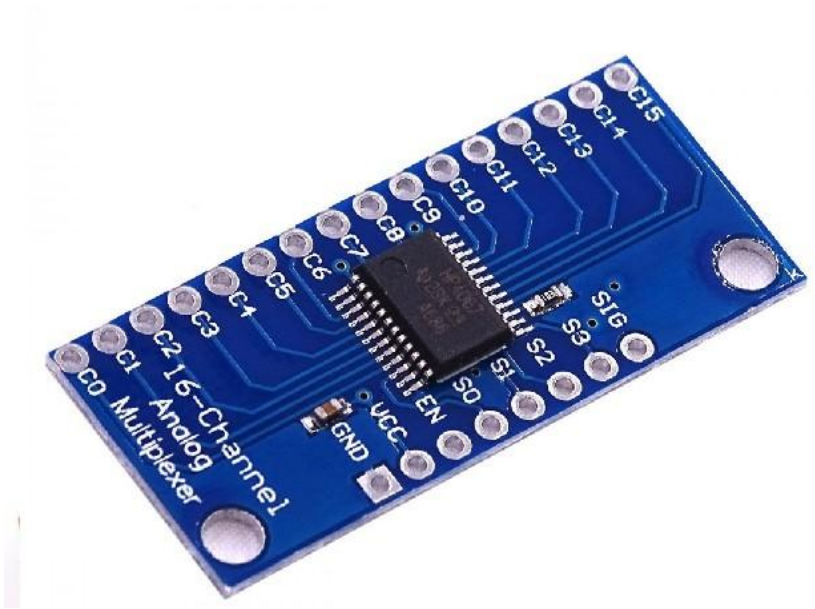
74LS137 Datasheet

3 Line to 8 Line Demultiplexer

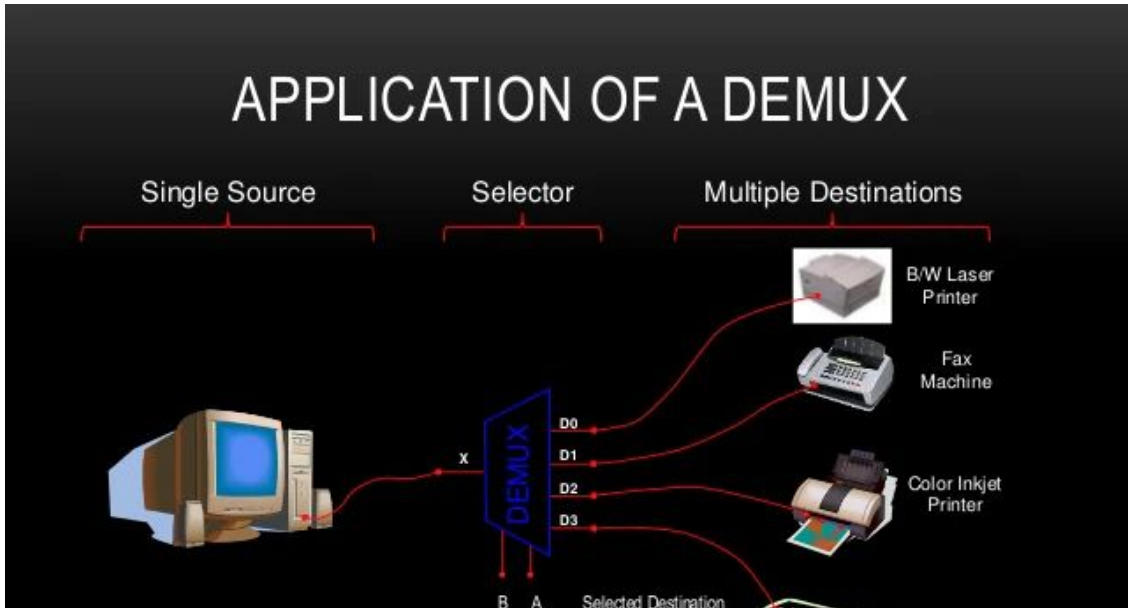
CIRCUITS DIY

MUX and DEMUX applications

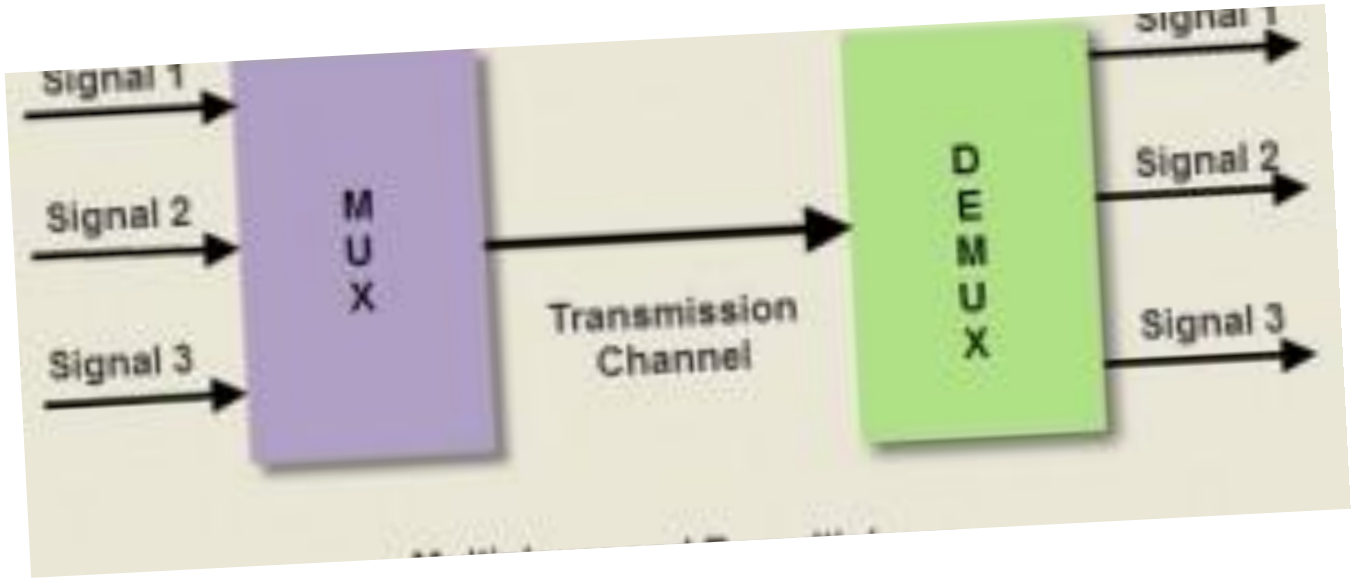




PINOY STORE



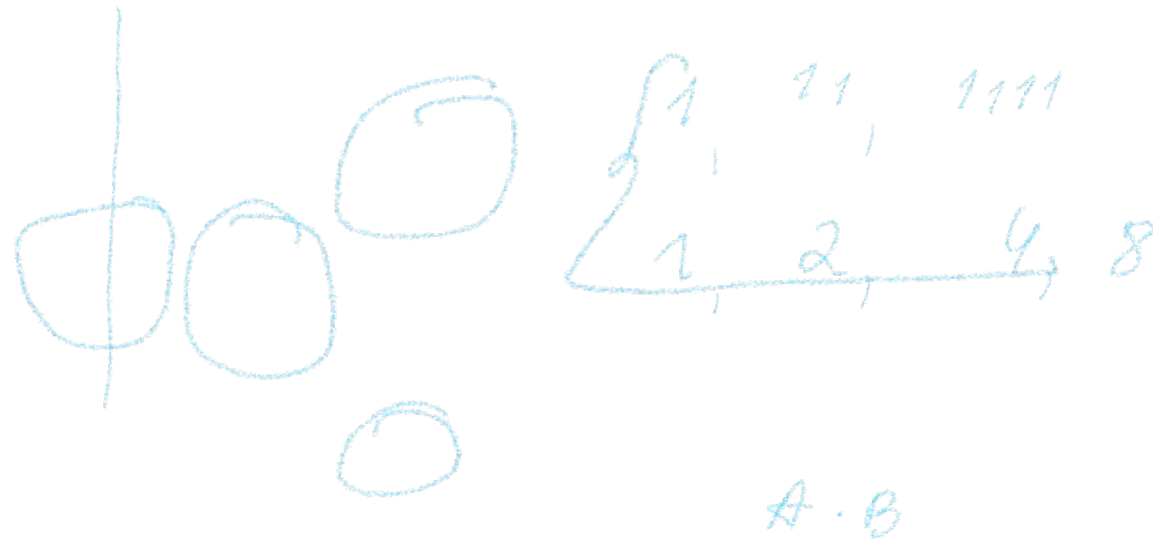
MUX and DEMUX applications

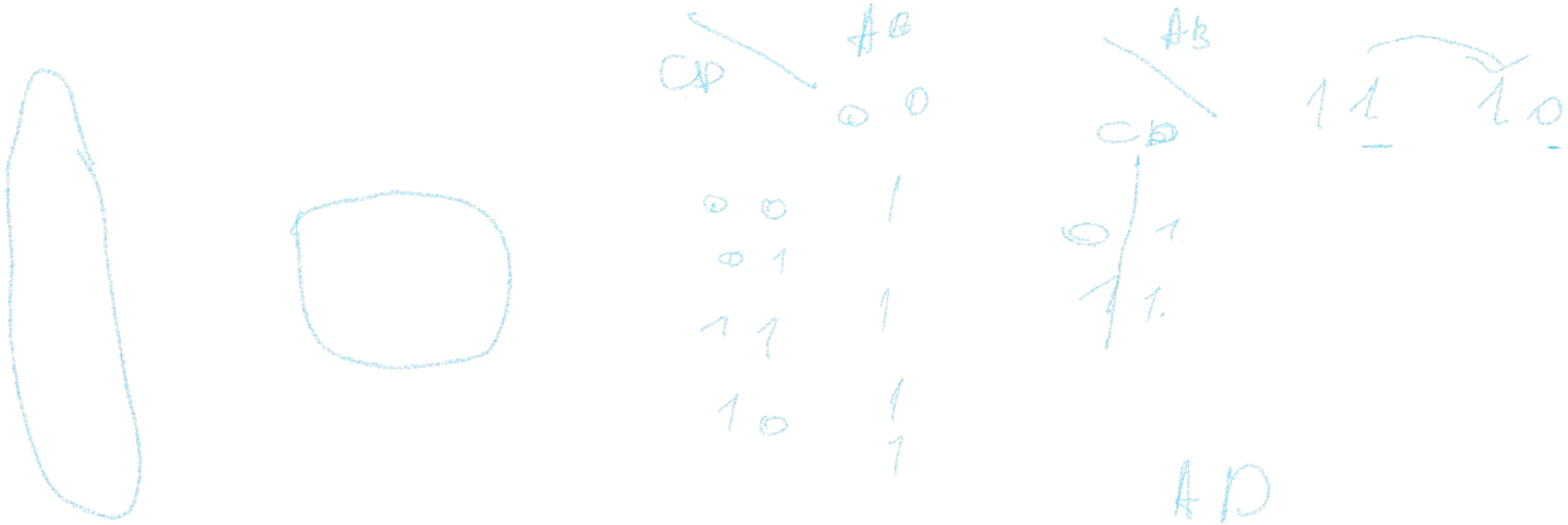

Signal 1
Signal 2
Signal 3
MUX
Transmission Channel
DEMUX
Signal 1
Signal 2
Signal 3

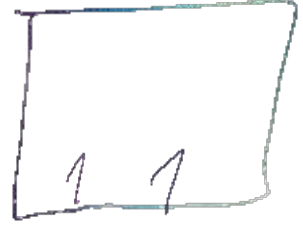
LECTURE 8 K-MAP K MAP KMAP

Introduction to Karnaugh map (k-map)

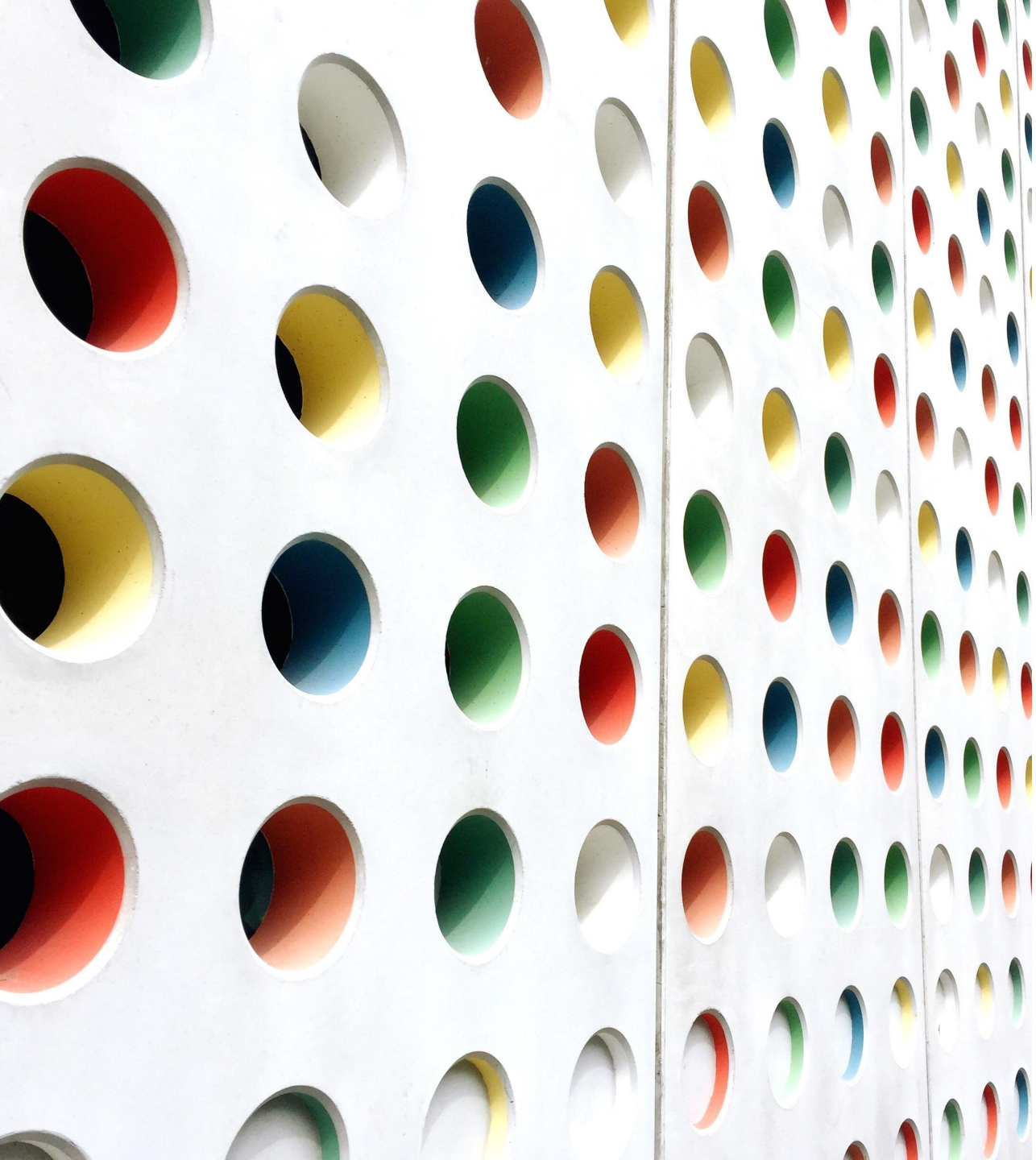
CPC (SIS) by Dana Utebayeva


1
11
111
1
1
2
4
8
A · B




1 1

LECTURE 9

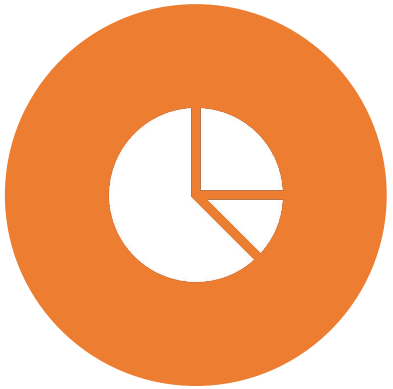


Sequential Circuit. Sequential logic design. Flip-Flop. Counters

Lecture 9

By Dana Utebayeva

Sequential Circuit. Sequential logic design. Flip-Flop. Counters



LATCHES

FLIP-FLOPS

WHAT IS THE CLOCK?

In Sequential Circuit, the Present Output depends on the Present Input as well as Past output / outputs

Cascaded NOT logic gate

The basic storage element is called latch

LECTURE 11-12

Lecture 10-11

MICROCOMPUTER ARCHITECTURE

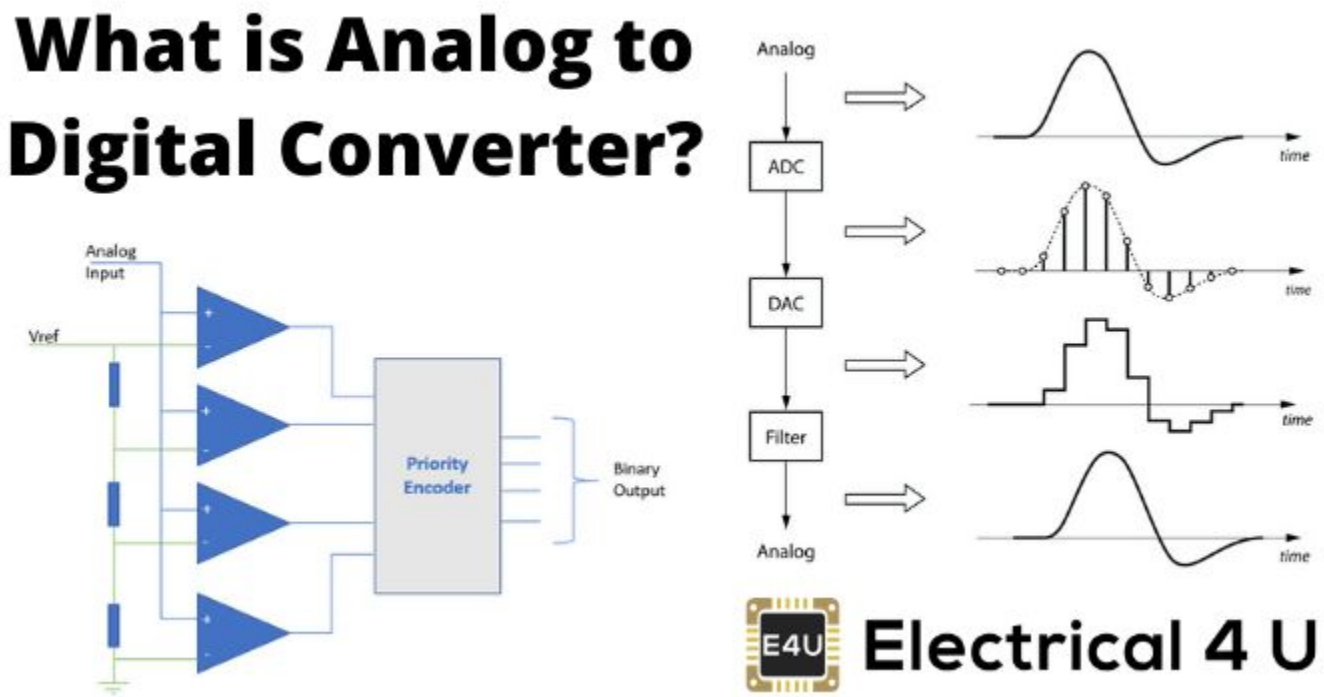
Memory

ADC and DAC

Outline

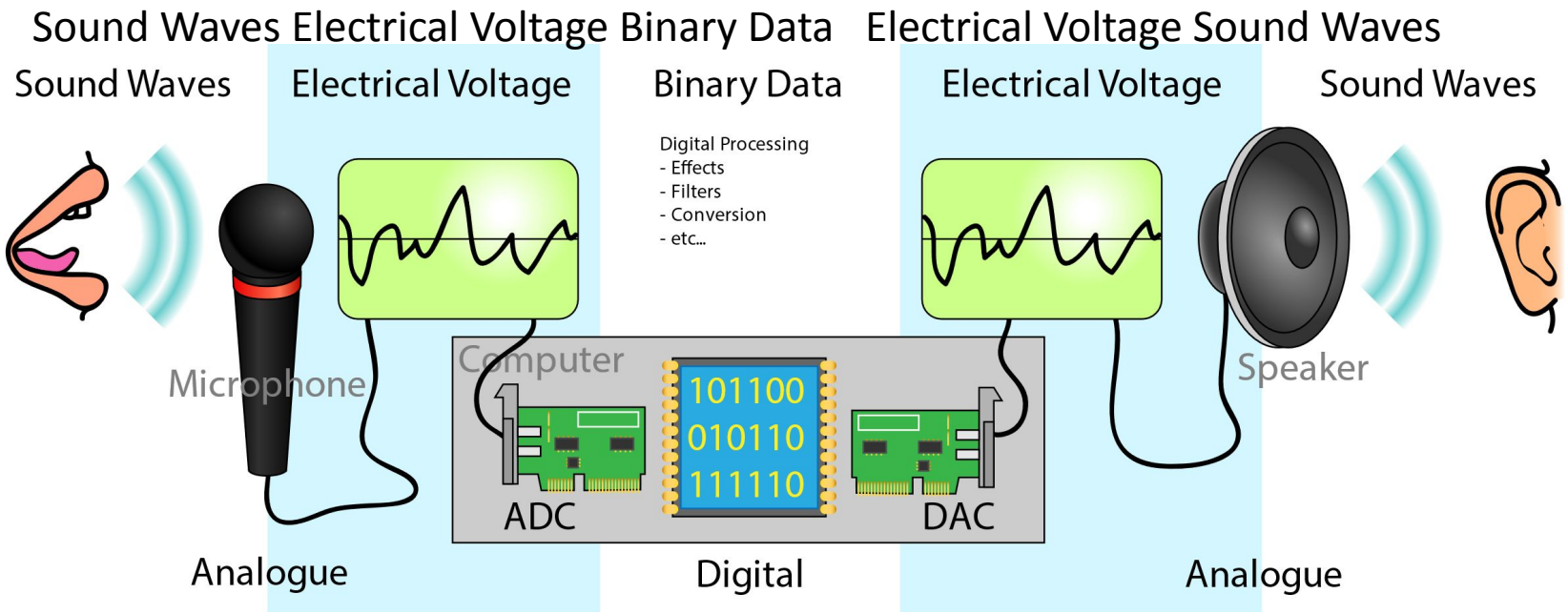
- ADC and DAC
- 1.1 Basic Blocks of a Microcomputer
- 1.2 Typical Microcomputer Architecture
- 1.3 Single-Chip Microprocessor
- 1.4 Program Execution by Conventional Microprocessors
- 1.5 Program Execution by typical 32-bit Microprocessors
- 1.6 Scalar and Superscalar Microprocessors
- 1.7 RISC vs. CISC

ADC

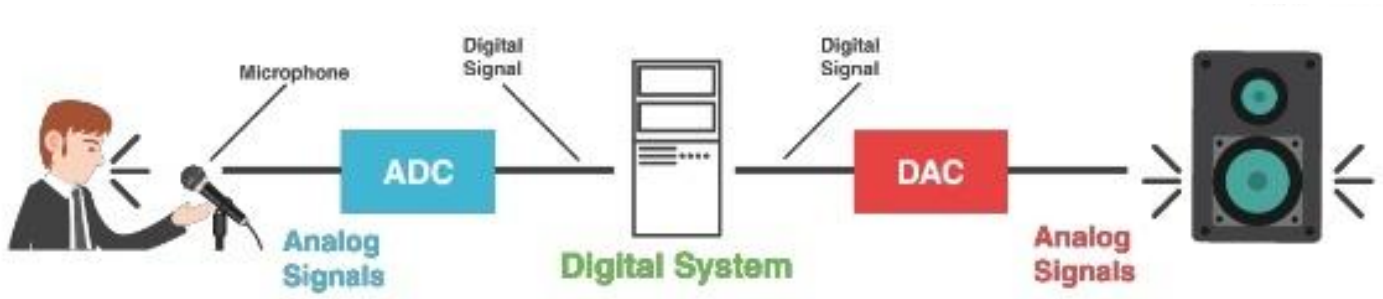

What is Analog to Digital Converter?
Analog Input
Vref
Priority Encoder
Binary Output
Analog
ADC
DAC
Filter
Analog
time
time
time
time
E4U
Electrical 4 U

ADC and DAC

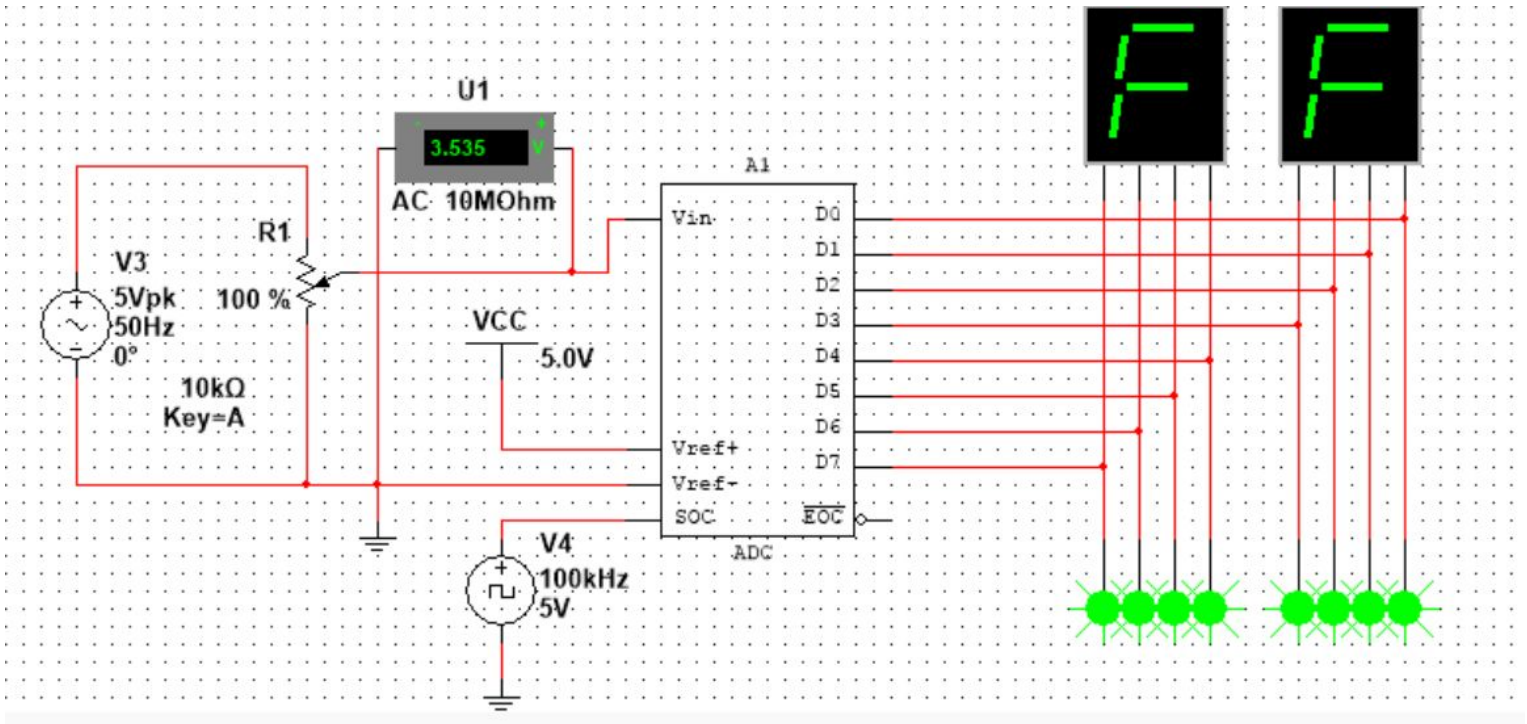
Work use of ADC and DAC



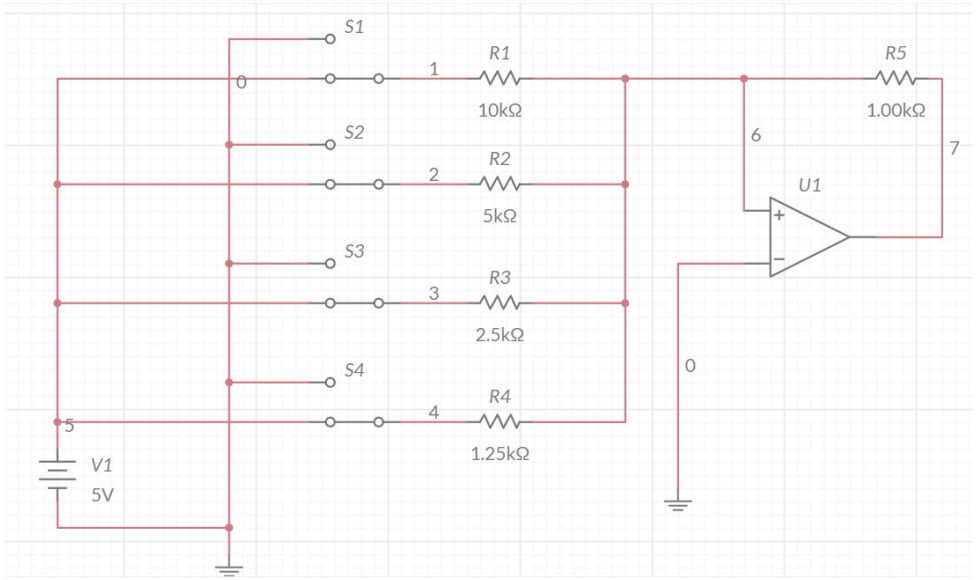
Need Conversation



ADC in Multisim



DAC



DAC scheme DAC circuit

2.1 Basic Blocks of a Microcomputer

- A microcomputer has three basic blocks: a central processing unit (CPU), a memory unit,

  and an input/output (I/O) unit.

- The CPU(microprocessor) executes all the instructions and performs arithmetic and logic operations on data.

- A *memory unit stores both data and instructions. The memory section typically*

  contains ROM and RAM chips.

- A system bus (comprised of several wires) connects these blocks.

2.1 Basic Blocks of a Microcomputer

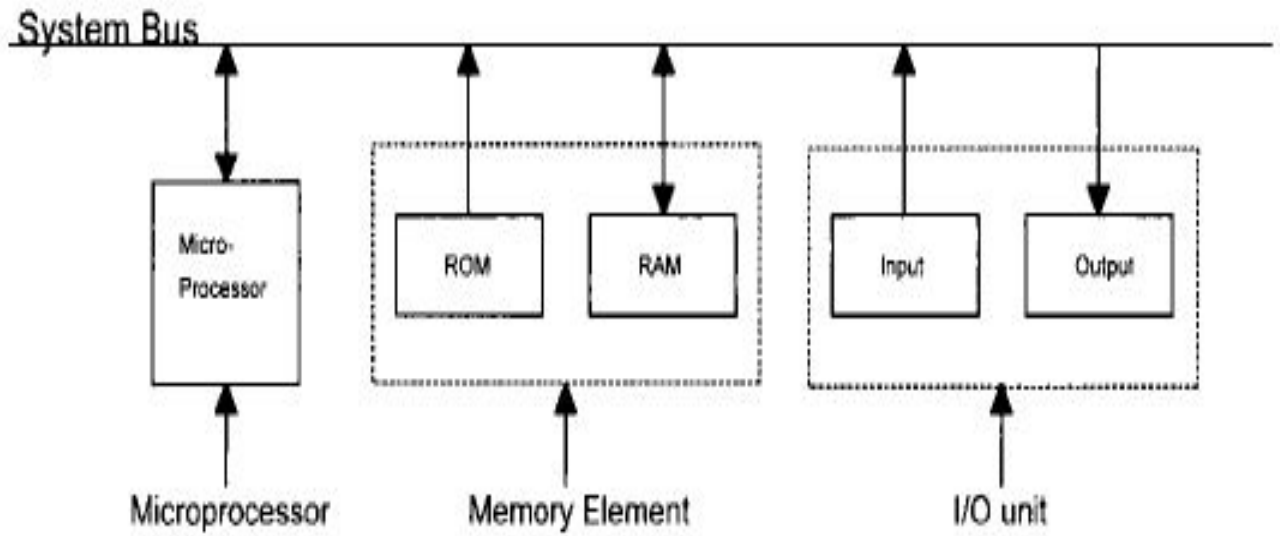


FIGURE 2.1 Basic blocks of a microcomputer.

System bus

2.1 Basic Blocks of a Microcomputer

- In a single-chip microcomputer, these three elements are on one chip, whereas
- in a single-chip microprocessor, separate chips are required for memory and I/O.

2.2 Typical Microcomputer Architecture

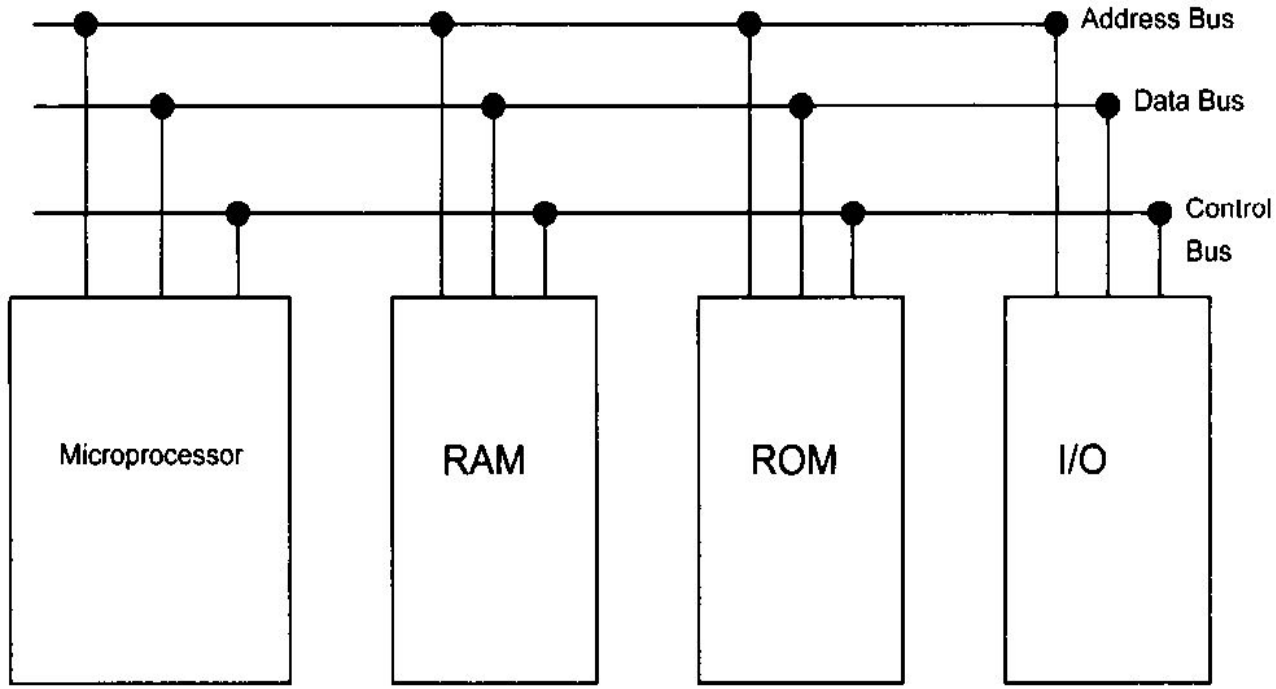


FIGURE 2.2 Simplified version of a typical microcomputer.

Simplified version of typical microprocessor

2.2.1 System Bus

- The microcomputer's system bus contains three buses, address, data, and control bus
- When a memory or an I/O chip receives data from the microprocessor, it is called a ***WRITE operation,*** *and* data is written into a selected memory location or an I/O port (register).
- When a memory or an I/O chip sends data to the microprocessor, it is called a ***READ operation, and data is*** read from a selected memory location or an I/O port.

2.2.1 System Bus

The Address Bus

1. *Unidirectional bus: Information transfer takes place in only one direction, from* the microprocessor to the memory or I/O elements.
2. *Typically 20 to **32 bits long.***
3. The size of the address bus determines the total number of memory addresses available

For example : microprocessor with 32 address pins can generate 2^{32} = 4,294,964,296 bytes

2.2.1 System Bus

- ***The data bus,***

1. bidirectional bus: data *can flow in both directions, that is, to or from the* microprocessor.
2. The size of the data bus varies from one microprocessor to another.

2.2.1 System Bus

- **The *control bus***

1. *consists of a number of signals that are used to synchronize* operation of the individual microcomputer elements.

Is it *Unidirectional or* bidirectional bus ??

2.2.2 Clock Signals

- The system clock signals are contained in the control bus.

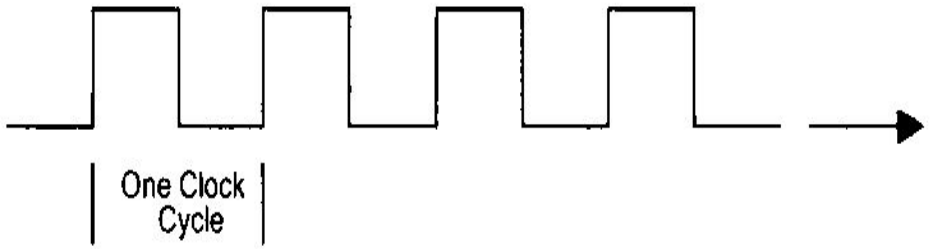


FIGURE 2.3 Typical clock signal.

- The number of cycles per second (hertz, abbreviated Hz) is referred to as the *clock frequency.*
- clock cycle = 1/f where f is the clock frequency.
- clock frequency determines the speed of the microcomputer.

2.3 Single-Chip Microprocessor

- The microprocessor is the CPU of the microcomputer
- The logic inside the microprocessor chip can be divided into three main areas: the

  register section, the control unit, and the arithmetic-logic unit (ALU).

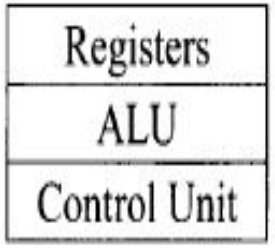


FIGURE 2.4 Microprocessor chip with the main functional elements.

2.3.1 Register Section

- The number, size, and types of registers vary from one microprocessor to another.
- **Basic Microprocessor Registers There are four basic microprocessor registers:** *instruction register, program counter, memory address register, and accumulator.*

2.3.1 Register Section

- ***Instruction register (IR) :***

❑ *The instruction register stores instructions.*

❑ The word size of the microprocessor determines the size of the instruction register. For example, a 32-bit microprocessor has a 32-bit instruction register.

2.3.1 Register Section

- ***Program Counter (PC):***

❑ *The program counter contains the address of the instruction or* operation code (op-code).

❑ The program counter normally contains the address of the next instruction to be executed.

❑ The size of the program counter is determined by the size of the address bus.

2.3.1 Register Section

How Program Counter is Work ?

1. Upon activating the microprocessor's RESET input, the address of the first instruction to be executed is loaded into the program counter.
2. To execute an instruction, the microprocessor typically places the contents of the program counter on the address bus and reads ("fetches") the contents of this address(i.e., instruction) from memory
3. The program counter contents are incremented automatically by the microprocessor's internal logic. Microprocessor executes a program sequentially, unless the program contains an instruction such as a JUMP instruction, which changes the sequence.

2.3.1 Register Section

- ***Memory Address Register (MAR).***

The memory address register contains the address of data. The microprocessor uses the address, which is stored in the memory address register, as a direct pointer to memory. The contents of the address is the actual data that is being transferred.

2.3.1 Register Section

- ***General Purpose Register (GPR). For an 8-bit microprocessor, the general-purpose*** register is called the ***accumulator.***
- It stores the result after most ALU operations.
- These 8-bit microprocessors have instructions to shift or rotate the accumulator one bit to the right or left through the carry flag.
- In16- and 32-bit microprocessors the accumulator is replaced by a GPR.
- any GPR can be used as an accumulator.

2.3.1 Register Section

- ***General Purpose Register (GPR).***

The term ***general-purpose comes from the fact that these registers can hold data, memory***

addresses, or the results of arithmetic or logic operations.

- Most registers are general-purpose, but some, such as the program counter (PC),are provided for dedicated functions.

2.3.1 Register Section

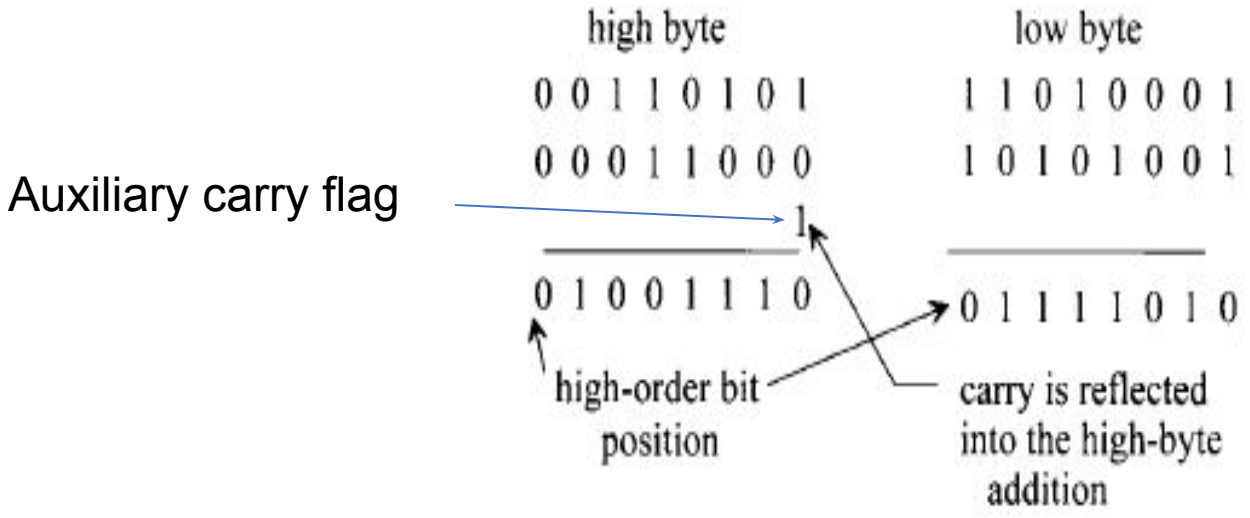
- **Other Microprocessor Registers** such as general-purpose registers, index register, status register and stack pointer register.

- **general-purpose registers** speeds up the execution of a program because the microprocessor does not have to read data from external memory via the data bus if data is stored in one of its general-purpose registers.
  - **Index Register is** typically used as a counter in address modification for an instruction or for general storage functions. Used to access tables or arrays of data.
  - **Status Register(a processor status word register or condition code register**, contains individual bits, with each bit having special significance. The bits in the status register are called flags.

2.3.1 Register Section

- **Flags Type**

❖ A **carry flag** is used to reflect whether or not the result generated by an arithmetic operation is greater than the microprocessor's word size.



2.3.1 Register Section

- **Flags Type**

❖ ***A zero flag*** is used to show whether the result of an operation is zero. It is set to1 if the result is zero, and it is reset to 0 if the result is nonzero.

❖ ***A parity flag*** is set to 1 to indicate whether the result of the last operation contains either an even number of 1's (even parity) or an odd number of 1's (odd parity), depending on the microprocessor.

2.3.1 Register Section

- **Flags Type**

❖ ***A sign flag (***sometimes called a negative flag) is used to indicate whether the result of the last operation is positive(set to 0) or negative(set to 1)

❖ ***Overflow flag*** arises from representation of the sign flag by the most significant bit of a word in signed binary operation. The overflow flag is set to1 if the result of an arithmetic operation is too big for the microprocessor's maximum word size, otherwise it is reset to 0

2.3.1 Register Section

- EXAMPLE :
- Find the sign,carry,zero,overflow,and parity even flag for the following arithmetic sign number:

(11110000)+(10100001) =10010001

SF =1 ,CF=1 ,ZF=0 ,OF=0 ,PF=0

2.3.1 Register Section

- **Stack Pointer Register** *A stack consists of a number of RAM locations set aside for* reading data from or writing data into these locations and is typically used by subroutines
- Two instructions, PUSH and POP, are usually available with a stack. The *PUSH* ***operation***

is defined as writing to the top or bottom of the stack, whereas the *POP* ***operation means*** reading from the top or bottom of the stack.

2.3.1 Register Section

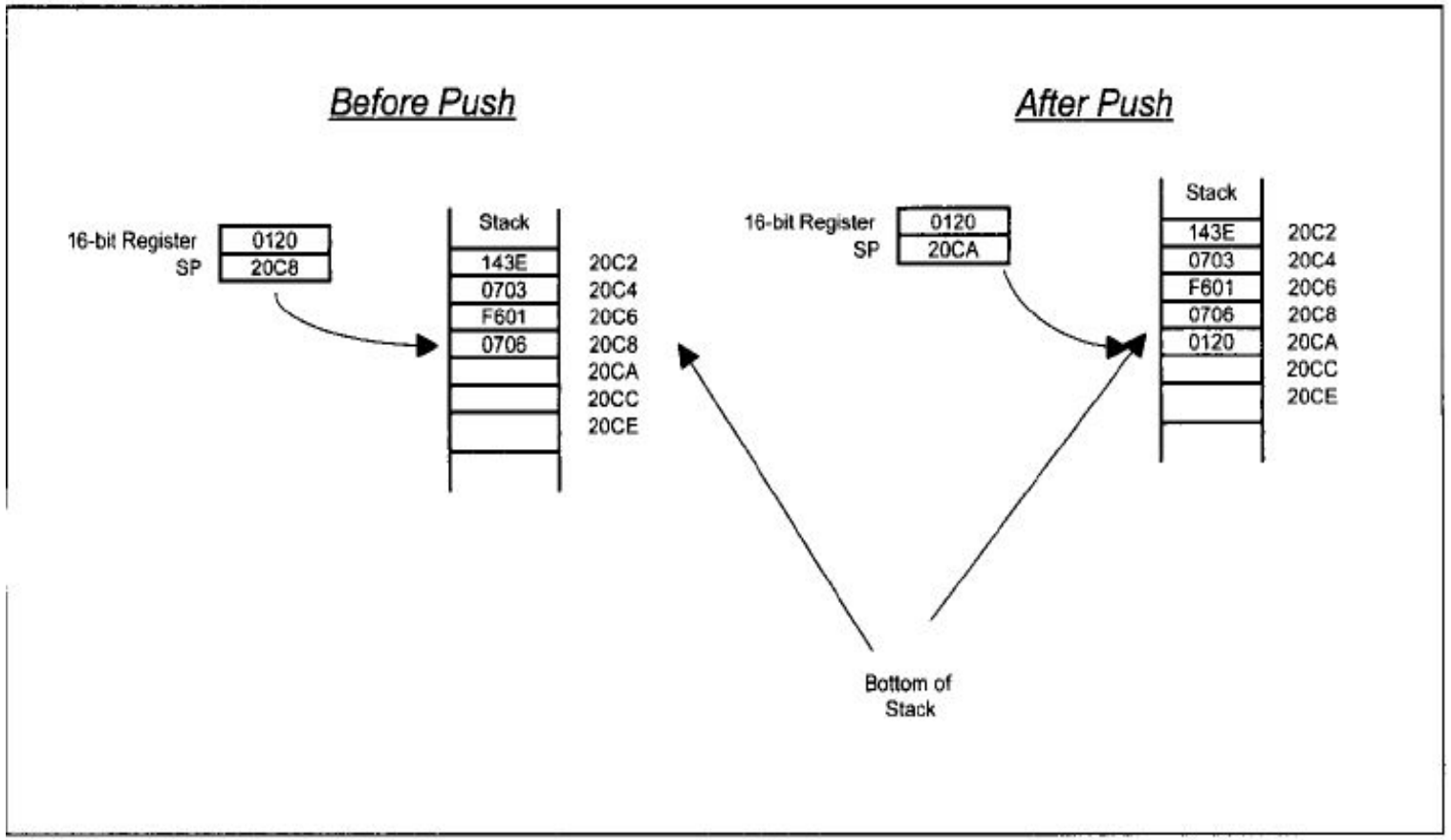


FIGURE 2.5 PUSH operation when accessing a stack from the bottom.

Push operation when accessing a stack from the bottom

2.3.1 Register Section

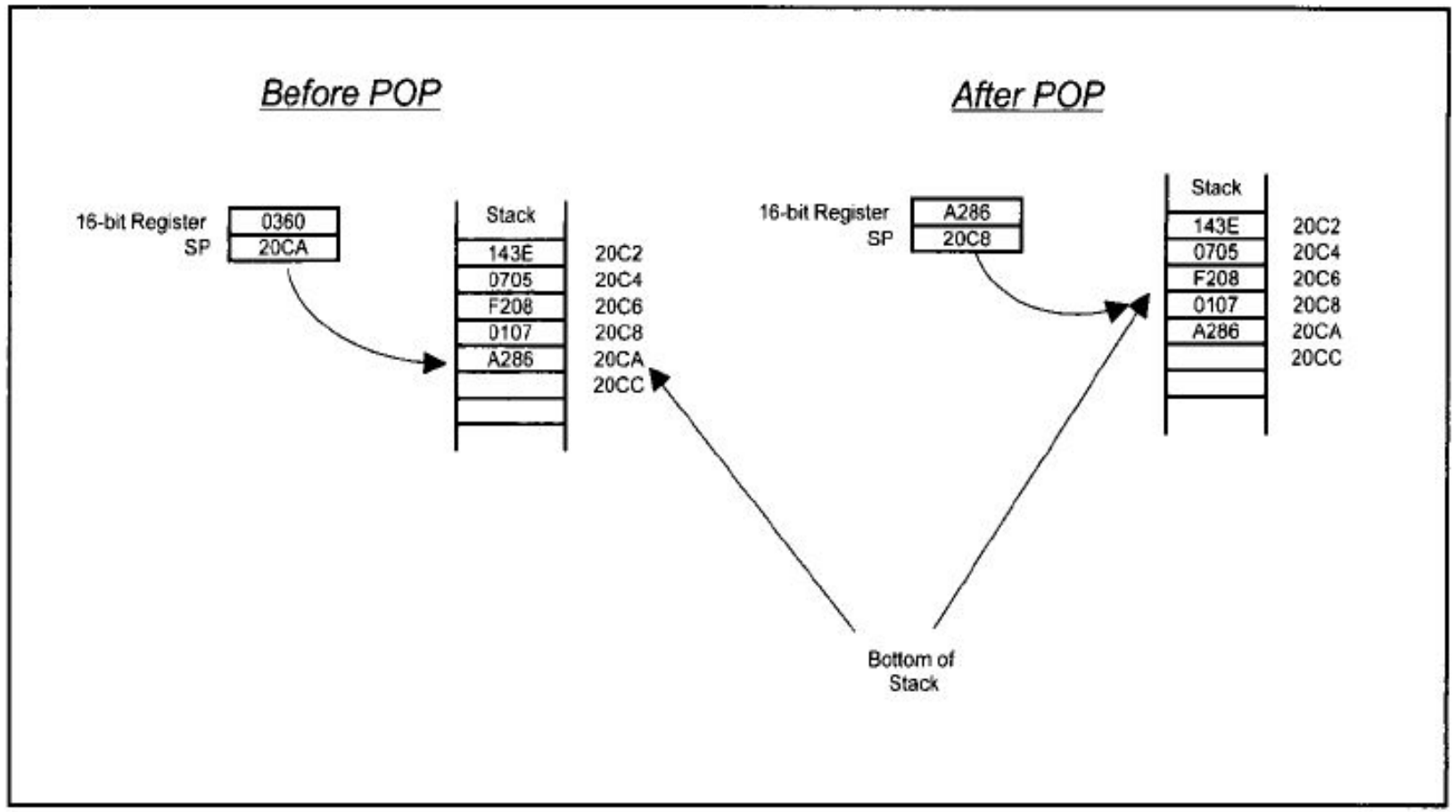


FIGURE 2.6 POP operation when accessing a stack from the bottom.

2.3.1 Register Section

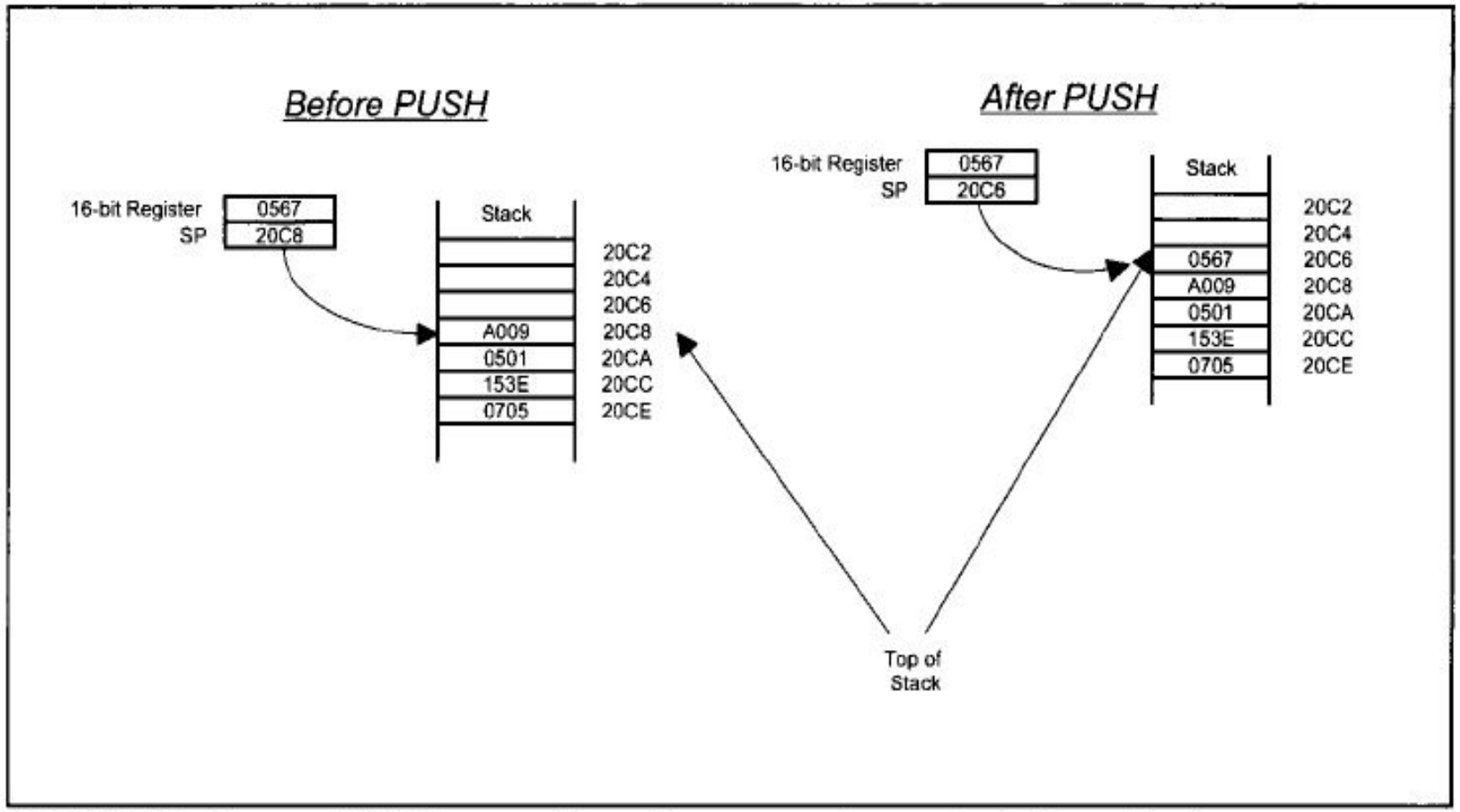


FIGURE 2.7 PUSH operation when accessing a stack from the top.

2.3.1 Register Section

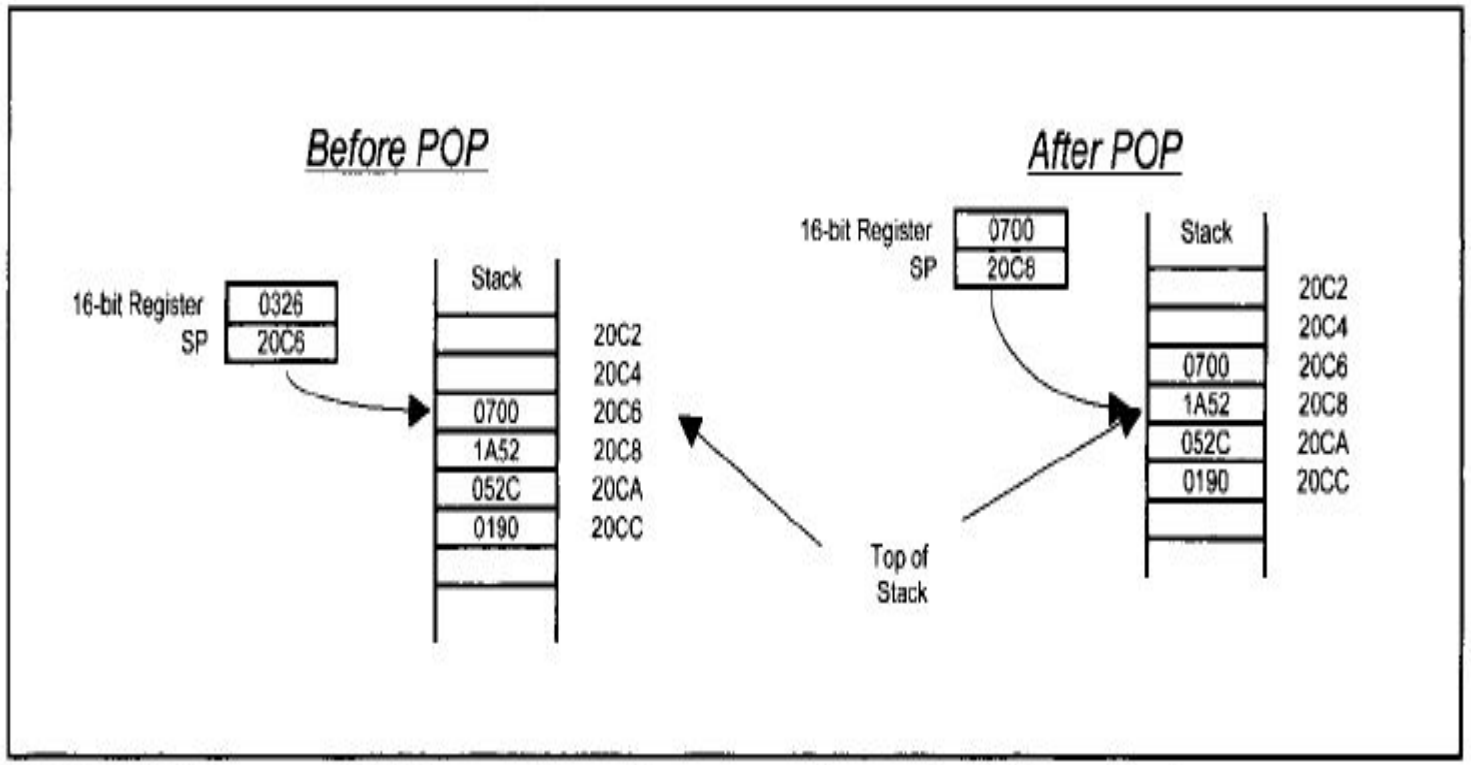


FIGURE 2.8 POP operation when accessing a stack from the top.

2.3.2 Control Unit

- The main purpose of the control unit is to read and decode instructions from the program memory.
- To execute an instruction, the control unit steps through the appropriate blocks of the ALU based on the op-codes contained in the instruction register.

2.3.2 Control Unit

Control Signal Actions

- **RESET.** This input is common to all microprocessors. When this input pin is driven HIGH or LOW (depending on the microprocessor), the program counter is loaded with a predefined address specified by the manufacturer.

2.3.2 Control Unit

Control Signal Actions

- **READ/WRITE *(R/W)*** This output line is common to all microprocessors. The status of this line tells the other microcomputer elements whether the microprocessor is performing a READ or a WRITE operation. A HIGH signal on this line indicates a READ operation, and a LOW indicates a WRITE operation.

2.3.2 Control Unit

Control Signal Actions

- **READY,** This is an input to a microprocessor. Slow devices (memory and I/O) use this signal to gain extra time to transfer data to or receive data from a microprocessor. The READY signal is usually an active low signal; that is, LOW indicates that the microprocessor is ready. Therefore, when the microprocessor selects a slow device, the device places a LOW on the READY pin. The microprocessor responds by suspending all its internal operations and enters a WAIT state. When the device is ready to send or receive data, it removes the READY signal. The microprocessor comes out of the WAIT state and performs the appropriate operation.

2.3.2 Control Unit

Control Signal Actions

- **Interrupt Request (INT or IRQ).** The external I/O devices can interrupt the microprocessor via this input pin on the microprocessor chip. When this signal is activated by the external devices, the microprocessor jumps to a special program called the *interrupt service routine. This program is normally written by the user for performing tasks that the* interrupting device wants the microprocessor to carry out. After completing this program, the microprocessor returns to the main program it was executing when the interrupt occurred.

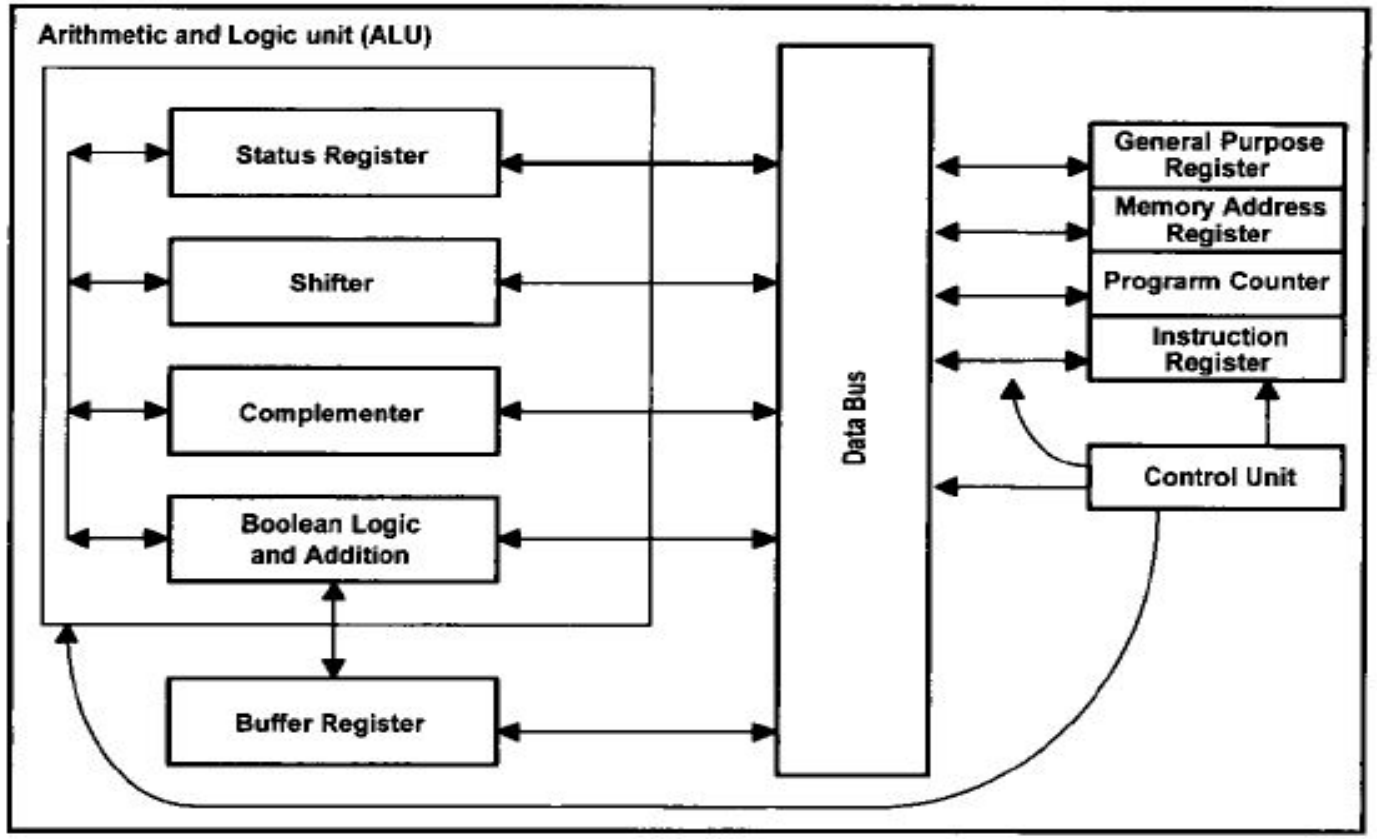
2.3.3 Arithmetic-Logic Unit

- The ALU performs all the data manipulations, such as arithmetic and logic operations, inside a microprocessor. The size of the ALU conforms to the word length of the microcomputer.
- ALU Functions:

1. Binary addition and logic operations
2. Finding the one's complement of data
3. Shifting or rotating the contents of a general-purpose register 1 bit to the left or right through a carry

2.3.4 Functional Representations of Simple and Typical Microprocessors

- Simple Micro



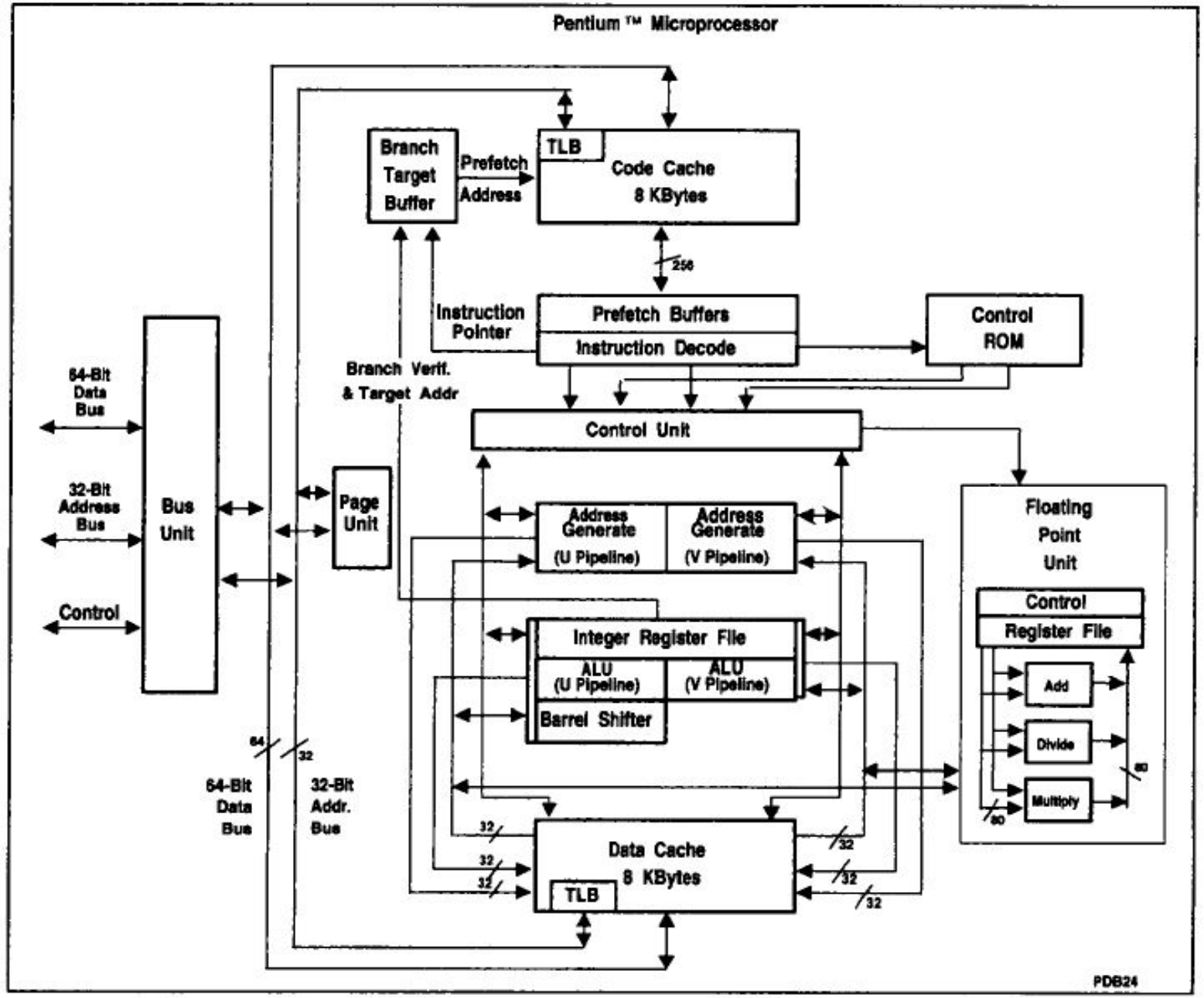
(a) Simple microprocessor

2.3.4 Functional Representations of Simple and Typical Microprocessors

- Buffer Register : Stores any data read from memory for further processing by the ALU.

2.3.4 Functional Representations of Simple and Typical Microprocessors

- Typical Microprocessor



(b) Pentium Microprocessor

- The Pentium contains two instruction pipelines: the U-pipe and the V-pipe. The U-pipe can execute all integer and floating-point instructions. The V-pipe can execute simple integer instructions
- The Pentium contains two separate cache memories: code cache and data cache.

2.3.5 Simplified Explanation of Control Unit design

- The control unit performs two basic operations:

1. instruction interpretation
2. and instruction sequencing.

2.3.5 Simplified Explanation of Control Unit design

- There are two methods for designing a control unit:

hardwired control	Microprogrammed control(firmware)
clocked sequential circuit.	ROM inside the control unit (*control memory)* more expensive flexibility

2.3.5 Simplified Explanation of Control Unit design

- How incrementing the contents of the register by 1 is done in microprogramming

control ??

(see figures in next slides)

2.3.5 Simplified Explanation of Control Unit design

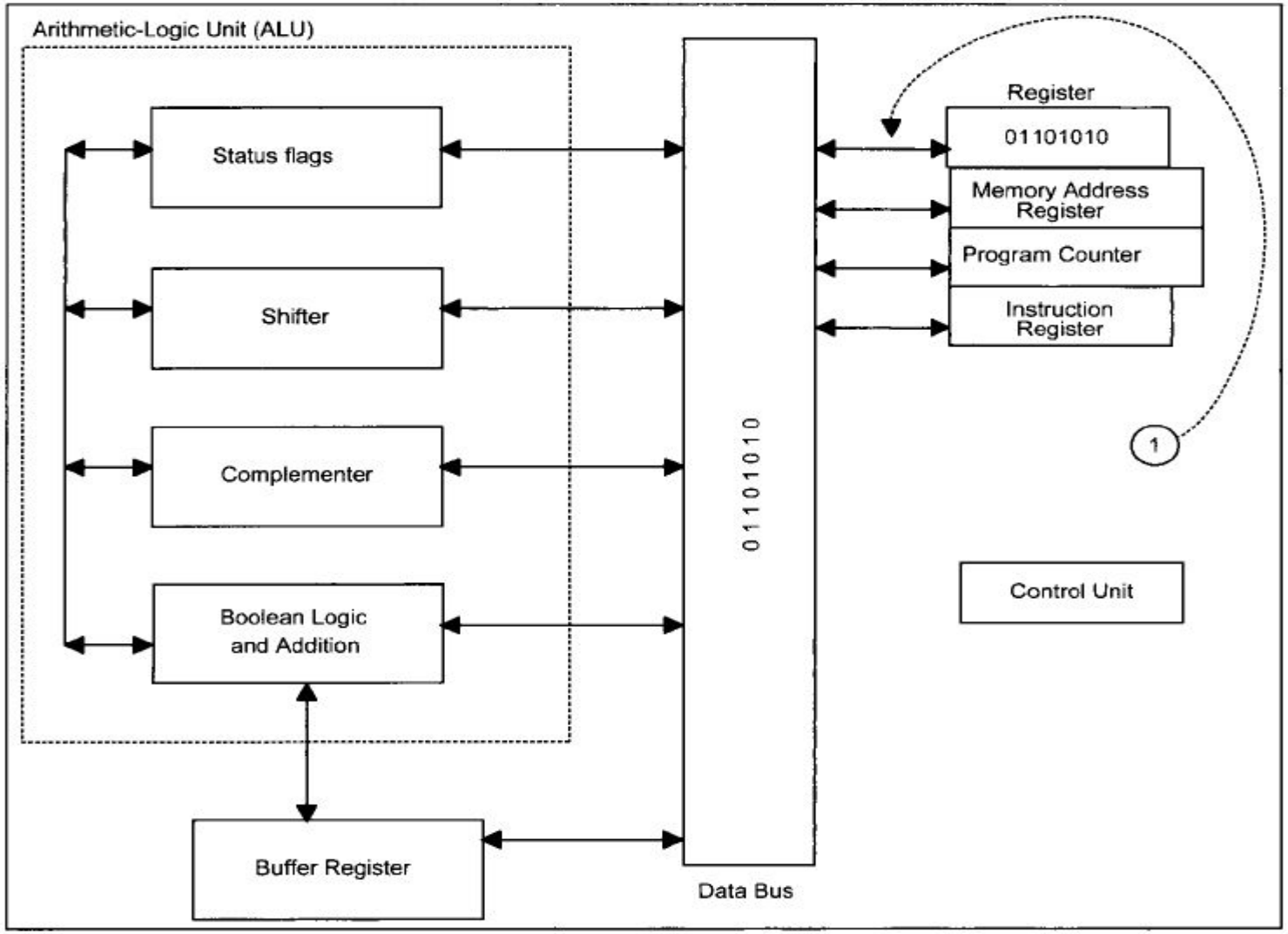


FIGURE 2.10 Transferring register contents to a data bus.

Transferring register contents to a data bus

2.3.5 Simplified Explanation of Control Unit design

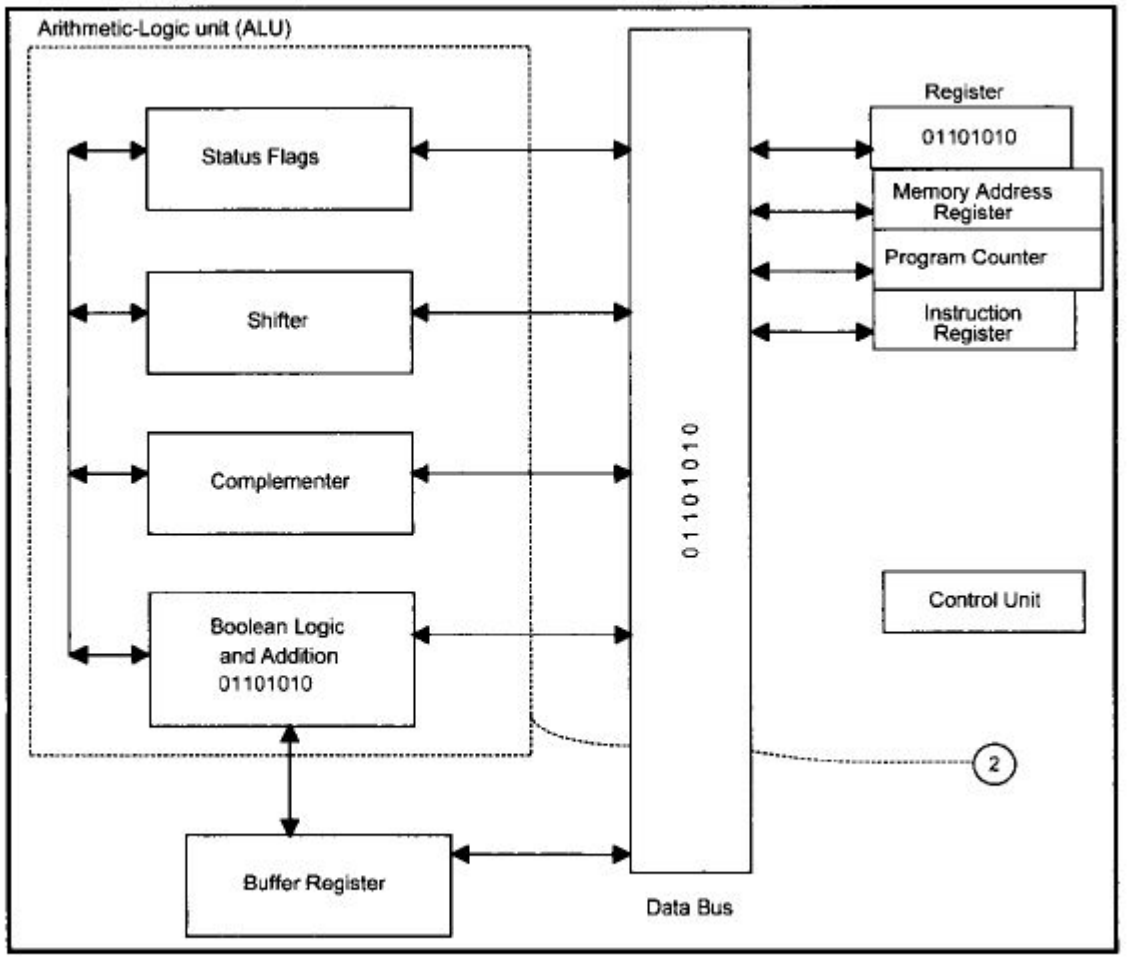


FIGURE 2.11 Transferring data bus contents to an ALU.

2.3.5 Simplified Explanation of Control Unit design

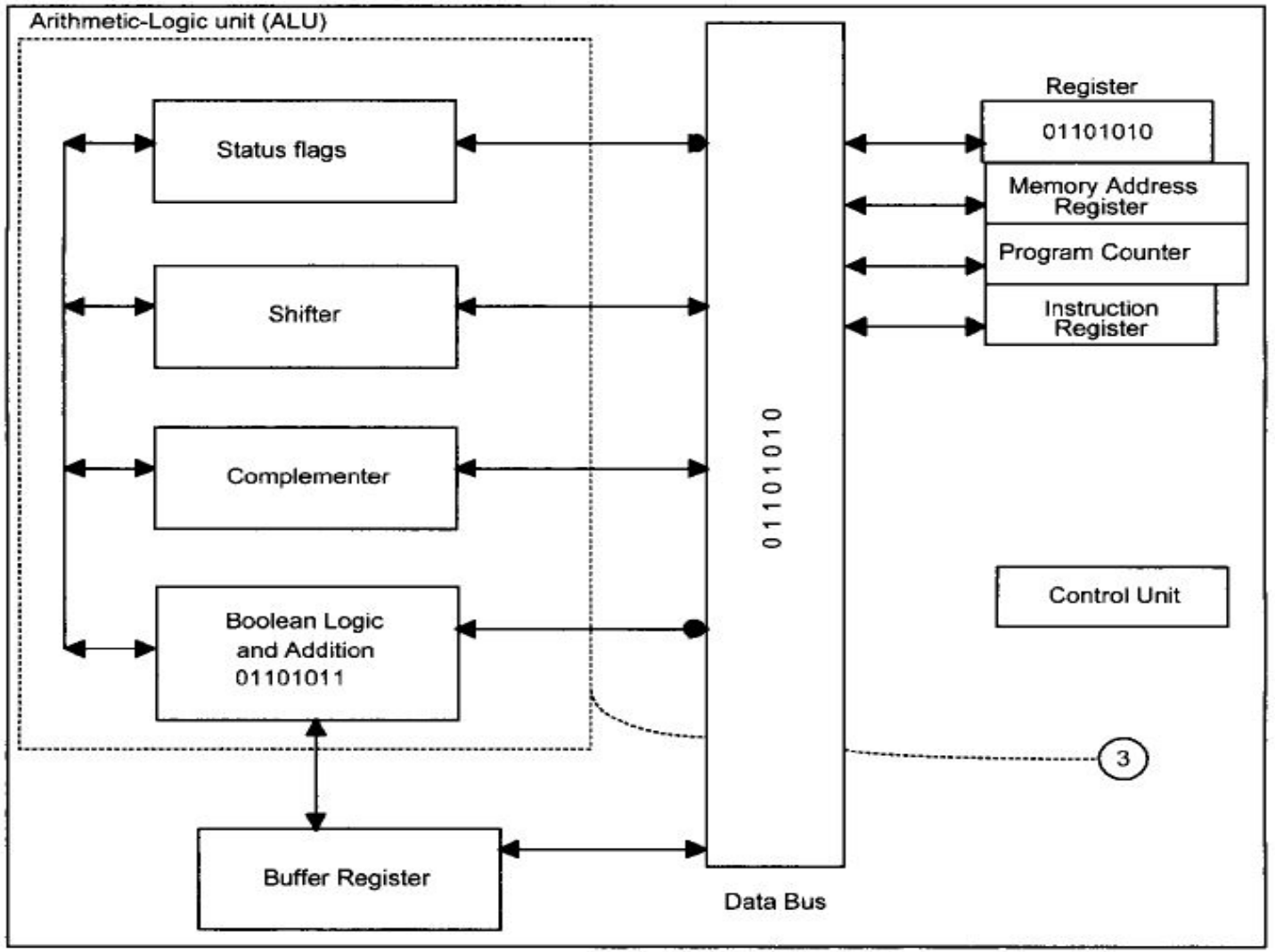


FIGURE 2.12 Activating the ALU logic.

2.3.5 Simplified Explanation of Control Unit design

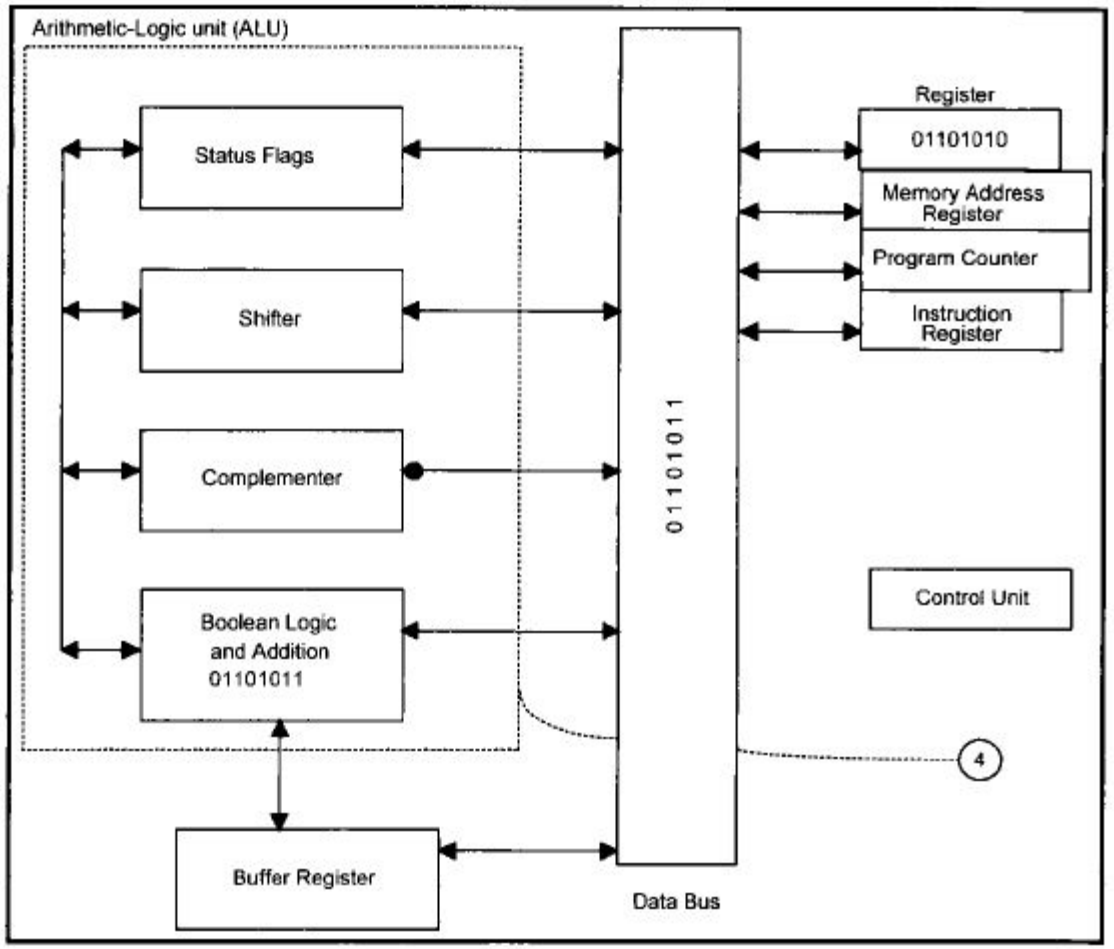


FIGURE 2.13 Transferring an ALU result to a data bus.

2.3.5 Simplified Explanation of Control Unit design

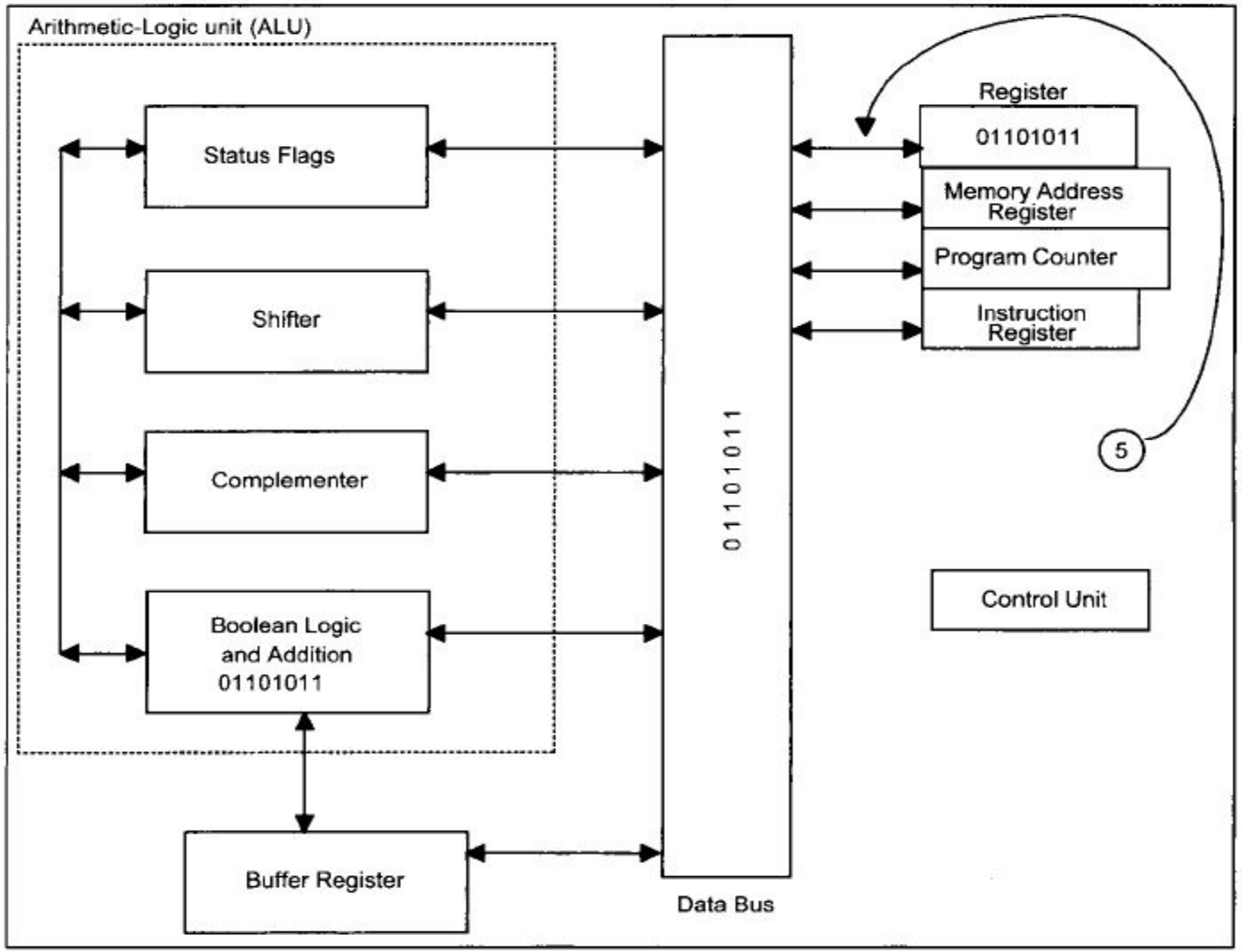


FIGURE 2.14 Transferring a data bus.

2.4 Program Execution by Conventional Microprocessors

- The following three steps for completing the instruction:

1. *Fetch. The microprocessor fetches (instruction read) the instruction from the main* memory (external to the microprocessor) into the instruction register.

2. *Decode. The microprocessor decodes or translates the instruction using the control unit.* The control unit inputs the contents of the instruction register, and then decodes (translates) the instruction to determine the instruction type.

3. *Execute. The microprocessor executes the instruction using the control unit. To* accomplish the task, the control unit generates a number of enable signals required by the instruction.

2.4 Program Execution by Conventional Microprocessors

- For example, suppose that it is desired to add the contents of two registers, **X** and *Y, and store the result in register Z. To accomplish this, a conventional microprocessor* performs the following steps:

1. The microprocessor fetches the instruction into the instruction register.
2. The control unit (CU) decodes the contents of the instruction register.
3. The CU executes the instruction by generating enable signals for the register and **ALU** sections to perform the following:

   a. The CU transfers the contents of registers **X and *Y from the Register section into the*** ALU.

   b. The CU commands the **ALU to ADD.**

   c. The CU transfers the result from the ALU into register Z of the register section.

2.5 Program Execution by typical 32-bit Microprocessors

- Enhancement in 32-bit microprocessors (like Pentium) include : *cache memory*, *memory management*, *pipelining*, *floating-point arithmetic*, and *branch prediction*.
- Cache memory is a high-speed read/write memory implemented as on-chip hardware in typical 32-bit microprocessors in order to increase processing rates. This topic is covered in more detail in Chapter 3.

2.5 Program Execution by typical 32-bit Microprocessors

- **Memory management** allows programmers to write programs much larger than those that could fit in the main memory space available to the microprocessors; the programs are simply stored on a secondary device, such as a hard disk. This topic is covered in more detail in Chapter 3.

2.5.1 Pipelining

- Basic Concept

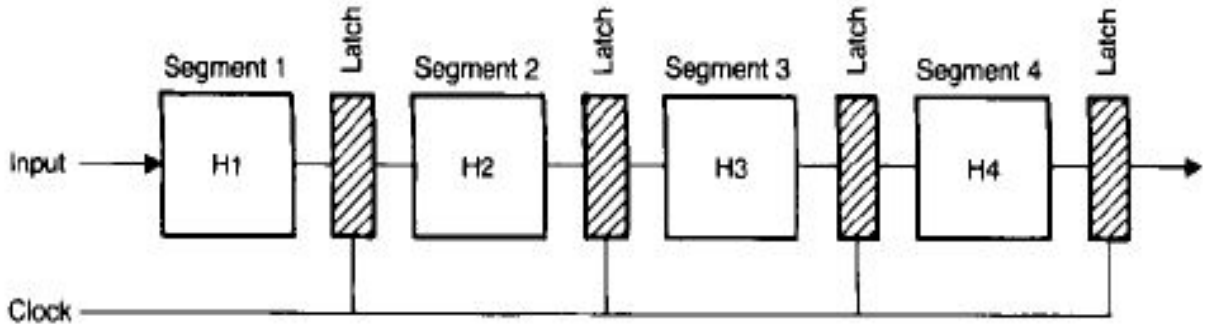


FIGURE 2.15 Four-segment pipeline.

Hi is Hardware designed to perform activity Ai

2.5.1 Pipelining

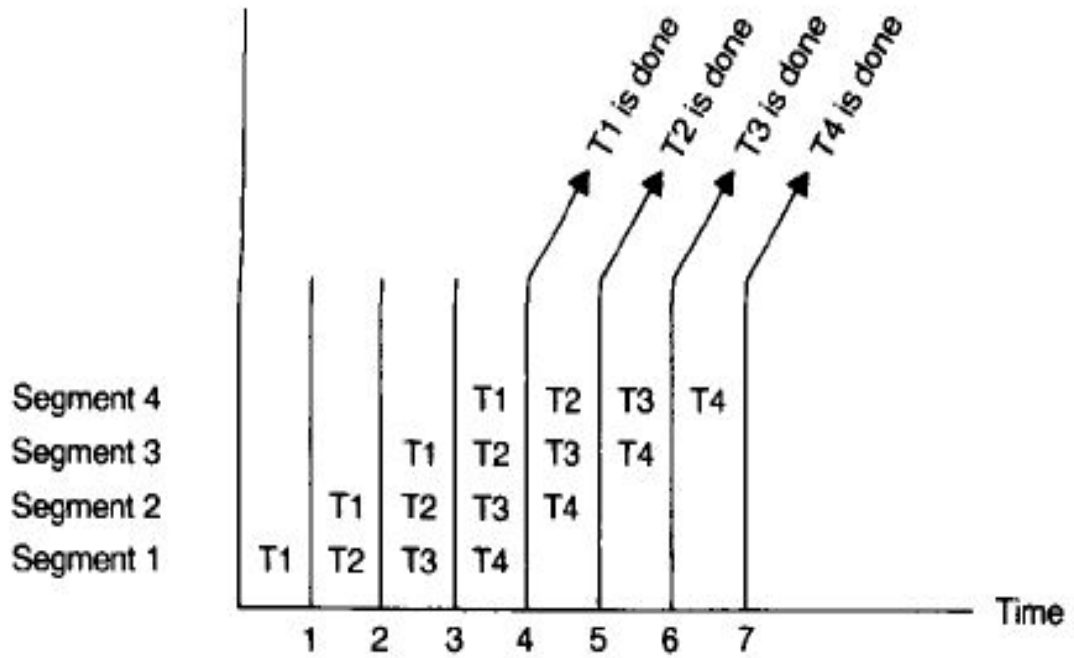


FIGURE 2.16 **Overlapped execution of four tasks using a pipeline.**

2.5.1 Pipelining

- Two Kind of Pipelining:

Arithmetic operations and instruction execution.

2.5.1 Pipelining

- **Arithmetic Pipelines**
- Consider the process of adding two floating-point numbers **x =**0.9234 * **10^4 and *y = 0.48 * 10 ^{2}.***

First: exponents of **x and *y are unequal.***

Second: exponent alignment.

Third: Perform the addition

Fourth: Normalize the final answer

segment

- Pipelined floating-point add/subtract unit

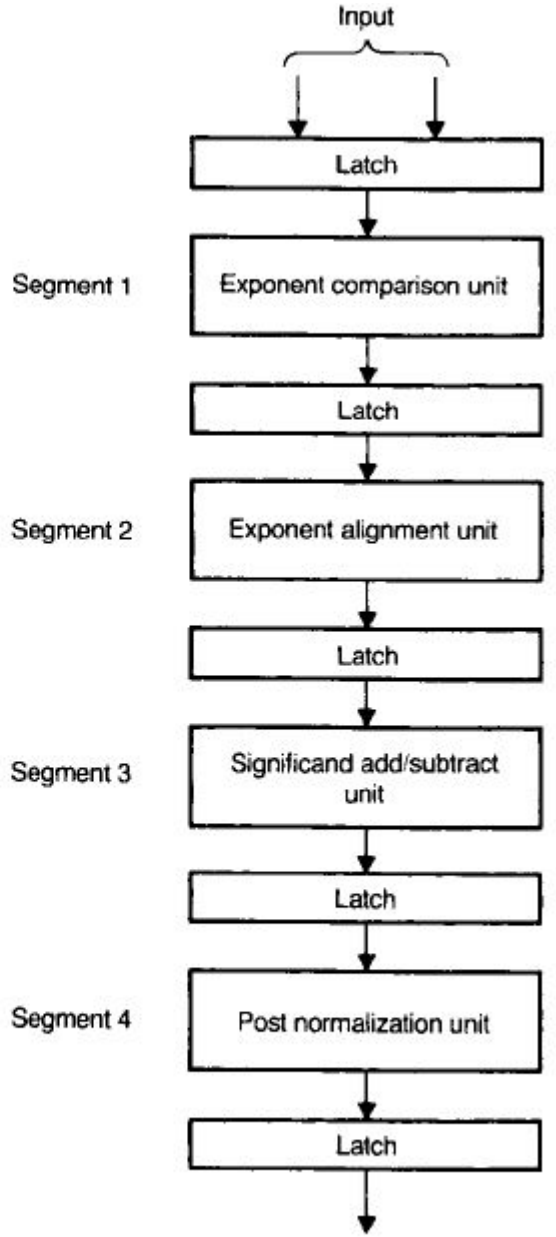


FIGURE 2.17 **Pipelined floating-point add/subtract unit.**

2.5.1 Pipelining

- **Instruction Pipelines**

Instruction cycle typically involves the

following activities:

- 1. Instruction fetch - needs five clocks to complete
- 2. Instruction decode
- 3. Operand fetch (Data Read)
- 4. Operation execution
- 5. Result routing.

Five-segment instruction pipeline

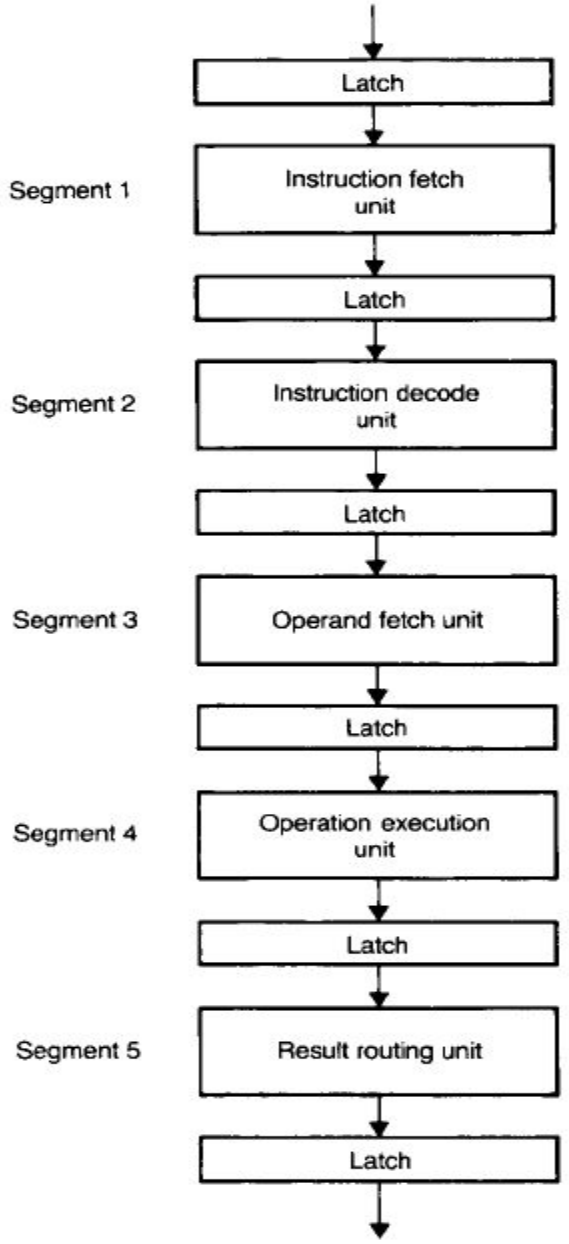


FIGURE 2.18 **Five-segment instruction pipeline.**

2.5.1 Pipelining

- Example of the execution of a stream of five instructions: 11,12,13,14, and 15, in which I3 is a conditional branch instruction.

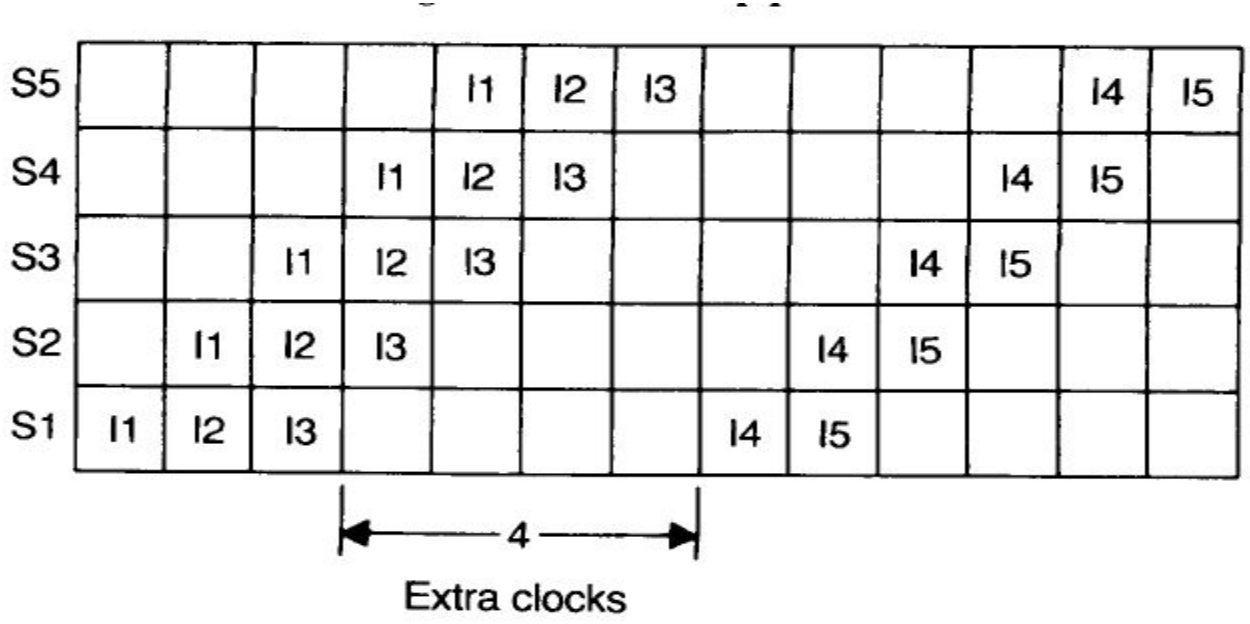


FIGURE 2.19 **Pipelined execution of a stream of five instructions that includes a branch instruction.**

2.5.2 Branch Prediction Feature

- This allows these microprocessors to anticipate jumps of the instruction flow ahead of time.

2.5.2 Branch Prediction Feature

- To accomplish this, the Pentium includes on-chip hardware called the *Branch Unit* (BU). The BU contains the branch execution unit (BEU) and the branch prediction unit (BPU). Whenever the Pentium encounters a conditional branch instruction, it sends it to the BU for execution. The BU evaluates the instruction's branch condition using the BEU and determines whether the branch should or should not be taken. Once the BU determines the branch condition, it calculates the starting address (Branch target) of the next block of code to be executed. The Pentium then starts fetching code at the new address.

2.6 Scalar and Superscalar Microprocessors

- Scalar processors such as the 80486 can execute one instruction per cycle.

  The 80486 contains only one pipeline.

- Superscalar microprocessors, can execute

  more than one instruction per cycle. These microprocessors contain more than one pipeline.

- The Pentium, a superscalar microprocessor, contains two independent pipelines. This

allows the Pentium to execute two instructions per cycle.

2.7 RISC vs. CISC

- There are two types of microprocessor architectures: RISC and CISC.
- RISC stand for (reduced instruction set computer) and CISC for (complex instruction set computer)

2.7 RISC vs. CISC

CISC	RISC
large number of instructions and many addressing modes	a simple instruction set with a few addressing modes
slower clock rate	fast clock rate
complex control unit, thus requiring microprogrammed implementation.	hardwired control Unit
more difficult to pipeline;	more efficient pipelining.
complex programs require fewer instructions in CISC	RISC requires a large number of instructions to accomplish the same task

2.7 RISC vs. CISC

- Intel’s original Pentium is a CISC microprocessor. Intel Pentium Pro and other succeeding members of the Pentium family and Motorola 68060 use a combination of RISC and CISC architectures for providing high performance. The Pentium Pro and other succeeding

LECTURE 11-12

Lecture 10-11

MICROCOMPUTER ARCHITECTURE

Memory

ADC and DAC

Outline

- ADC and DAC
- 1.1 Basic Blocks of a Microcomputer
- 1.2 Typical Microcomputer Architecture
- 1.3 Single-Chip Microprocessor
- 1.4 Program Execution by Conventional Microprocessors
- 1.5 Program Execution by typical 32-bit Microprocessors
- 1.6 Scalar and Superscalar Microprocessors
- 1.7 RISC vs. CISC

2.3.1 Register Section

- **Flags Type**

❖ ***A zero flag*** is used to show whether the result of an operation is zero. It is set to1 if the result is zero, and it is reset to 0 if the result is nonzero.

❖ ***A parity flag*** is set to 1 to indicate whether the result of the last operation contains either an even number of 1's (even parity) or an odd number of 1's (odd parity), depending on the microprocessor.

2.3.1 Register Section

- **Flags Type**

❖ ***A sign flag (***sometimes called a negative flag) is used to indicate whether the result of the last operation is positive(set to 0) or negative(set to 1)

❖ ***Overflow flag*** arises from representation of the sign flag by the most significant bit of a word in signed binary operation. The overflow flag is set to1 if the result of an arithmetic operation is too big for the microprocessor's maximum word size, otherwise it is reset to 0

2.3.1 Register Section

- EXAMPLE :
- Find the sign,carry,zero,overflow,and parity even flag for the following arithmetic sign number:

(11110000)+(10100001) =10010001

SF =1 ,CF=1 ,ZF=0 ,OF=0 ,PF=0

2.3.1 Register Section

- **Stack Pointer Register** *A stack consists of a number of RAM locations set aside for* reading data from or writing data into these locations and is typically used by subroutines
- Two instructions, PUSH and POP, are usually available with a stack. The *PUSH* ***operation***

is defined as writing to the top or bottom of the stack, whereas the *POP* ***operation means*** reading from the top or bottom of the stack.

2.3.1 Register Section

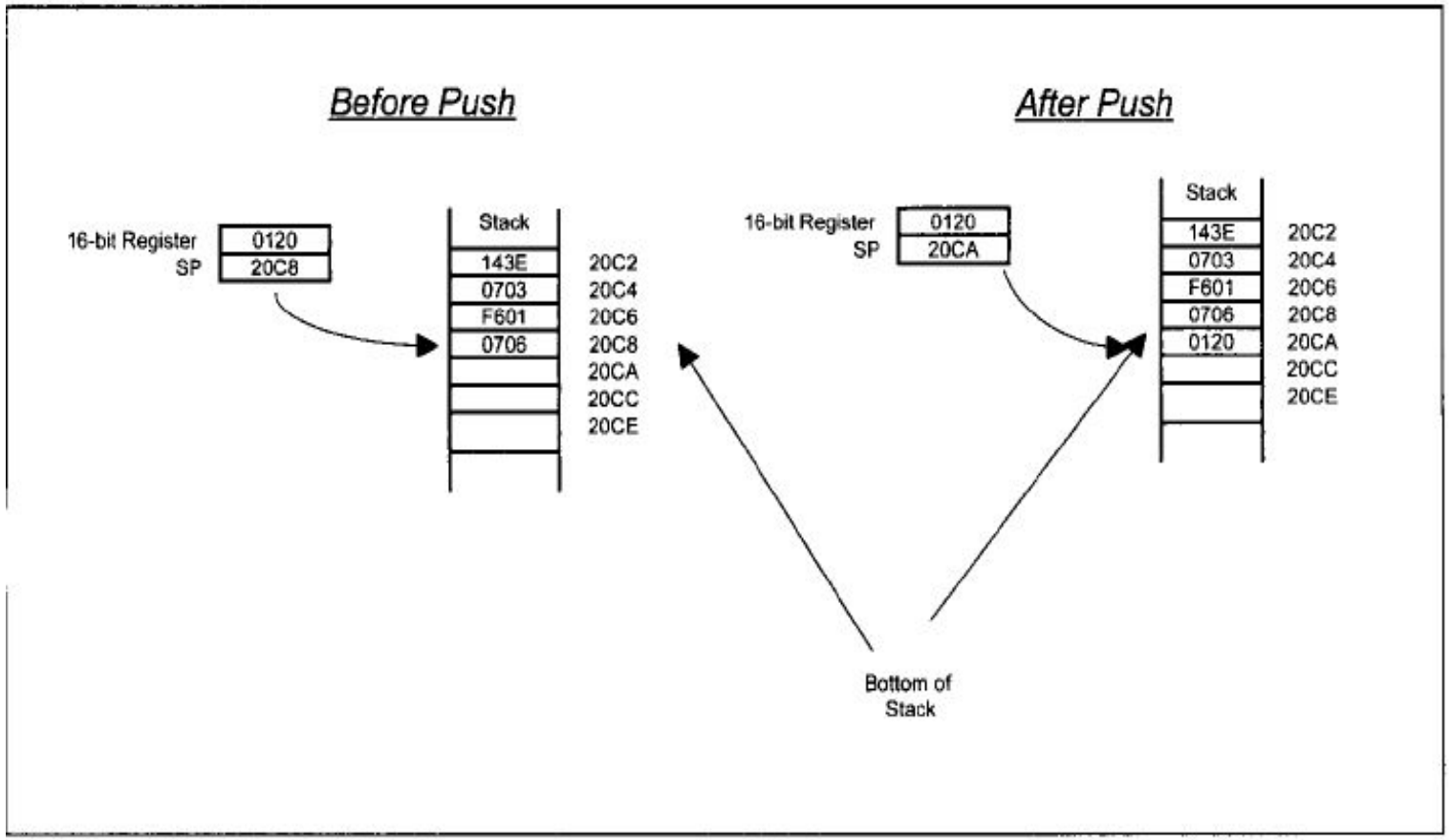


FIGURE 2.5 PUSH operation when accessing a stack from the bottom.

2.3.1 Register Section

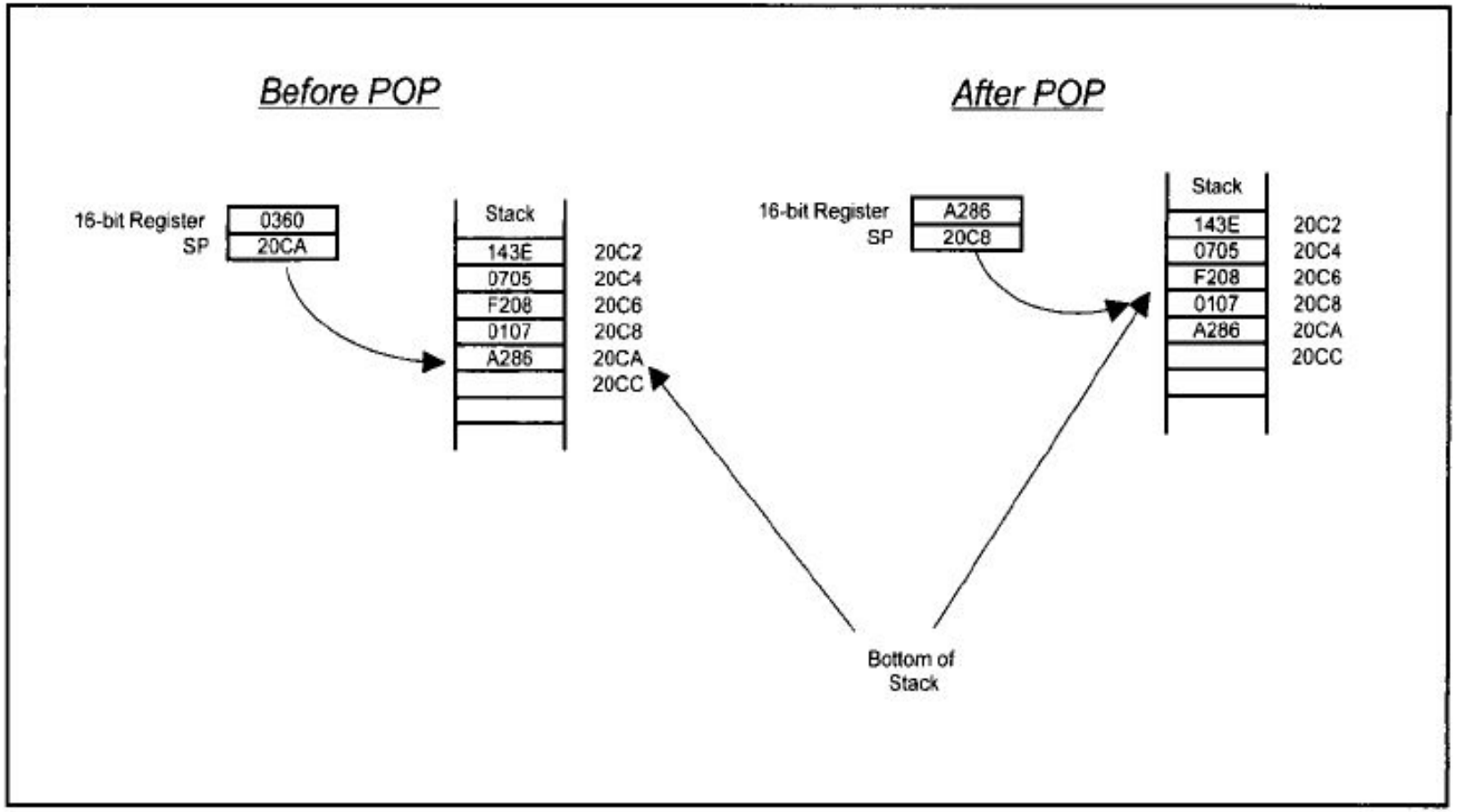


FIGURE 2.6 POP operation when accessing a stack from the bottom.

2.3.1 Register Section

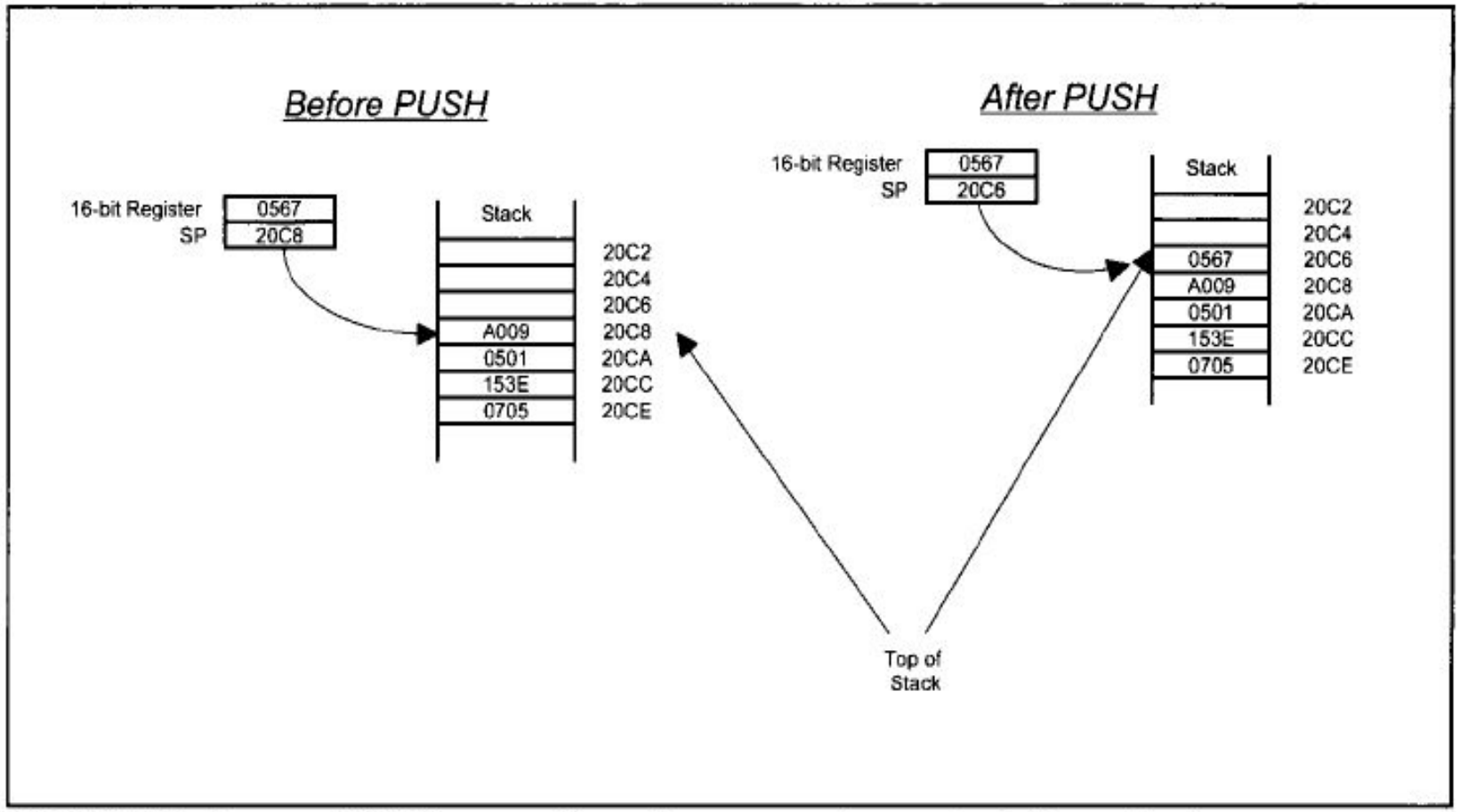


FIGURE 2.7 PUSH operation when accessing a stack from the top.

2.3.1 Register Section

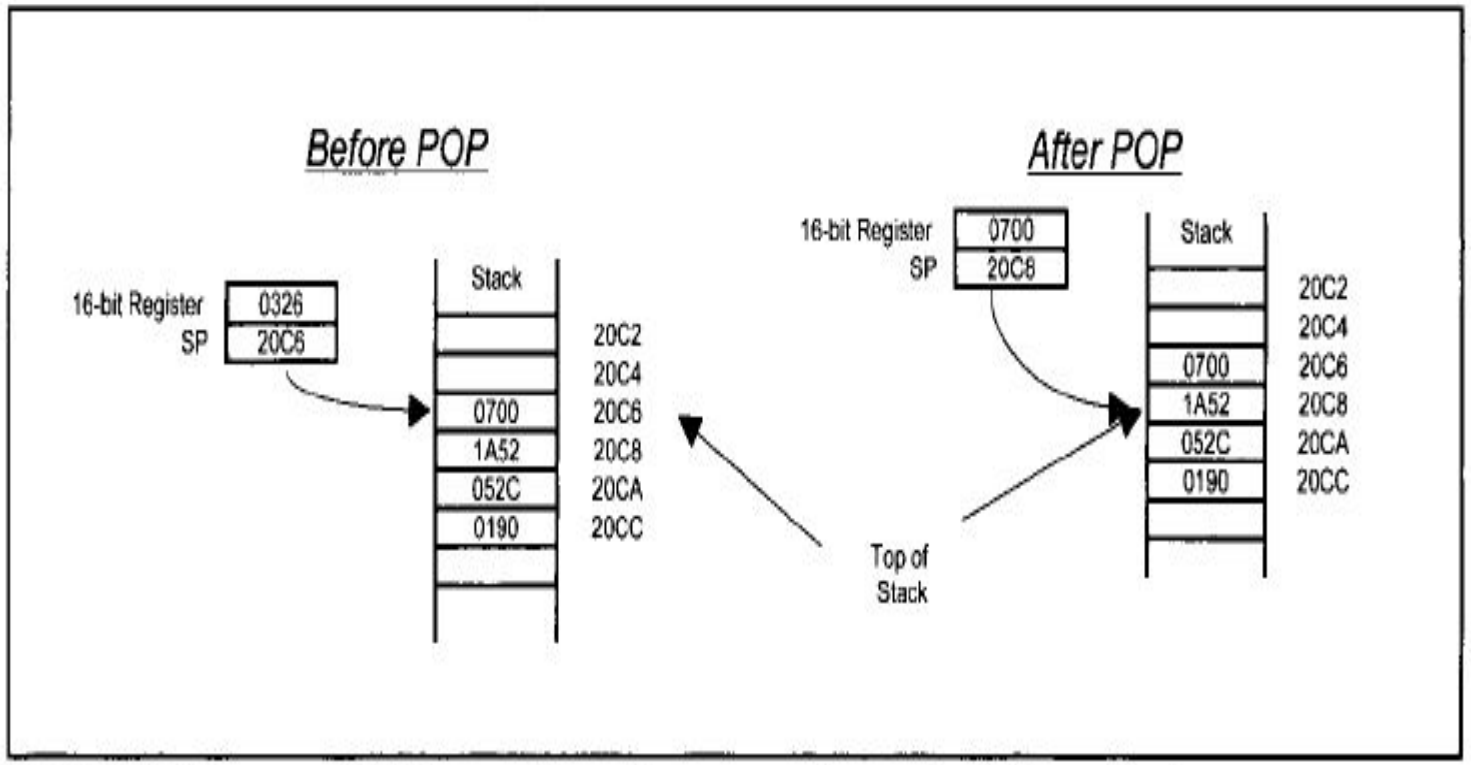


FIGURE 2.8 POP operation when accessing a stack from the top.

2.3.2 Control Unit

- The main purpose of the control unit is to read and decode instructions from the program memory.
- To execute an instruction, the control unit steps through the appropriate blocks of the ALU based on the op-codes contained in the instruction register.

2.3.2 Control Unit

Control Signal Actions

- **RESET.** This input is common to all microprocessors. When this input pin is driven HIGH or LOW (depending on the microprocessor), the program counter is loaded with a predefined address specified by the manufacturer.

2.3.2 Control Unit

Control Signal Actions

- **READ/WRITE *(R/W)*** This output line is common to all microprocessors. The status of this line tells the other microcomputer elements whether the microprocessor is performing a READ or a WRITE operation. A HIGH signal on this line indicates a READ operation, and a LOW indicates a WRITE operation.

2.3.2 Control Unit

Control Signal Actions

- **READY,** This is an input to a microprocessor. Slow devices (memory and I/O) use this signal to gain extra time to transfer data to or receive data from a microprocessor. The READY signal is usually an active low signal; that is, LOW indicates that the microprocessor is ready. Therefore, when the microprocessor selects a slow device, the device places a LOW on the READY pin. The microprocessor responds by suspending all its internal operations and enters a WAIT state. When the device is ready to send or receive data, it removes the READY signal. The microprocessor comes out of the WAIT state and performs the appropriate operation.

2.3.2 Control Unit

Control Signal Actions

- **Interrupt Request (INT or IRQ).** The external I/O devices can interrupt the microprocessor via this input pin on the microprocessor chip. When this signal is activated by the external devices, the microprocessor jumps to a special program called the *interrupt service routine. This program is normally written by the user for performing tasks that the* interrupting device wants the microprocessor to carry out. After completing this program, the microprocessor returns to the main program it was executing when the interrupt occurred.

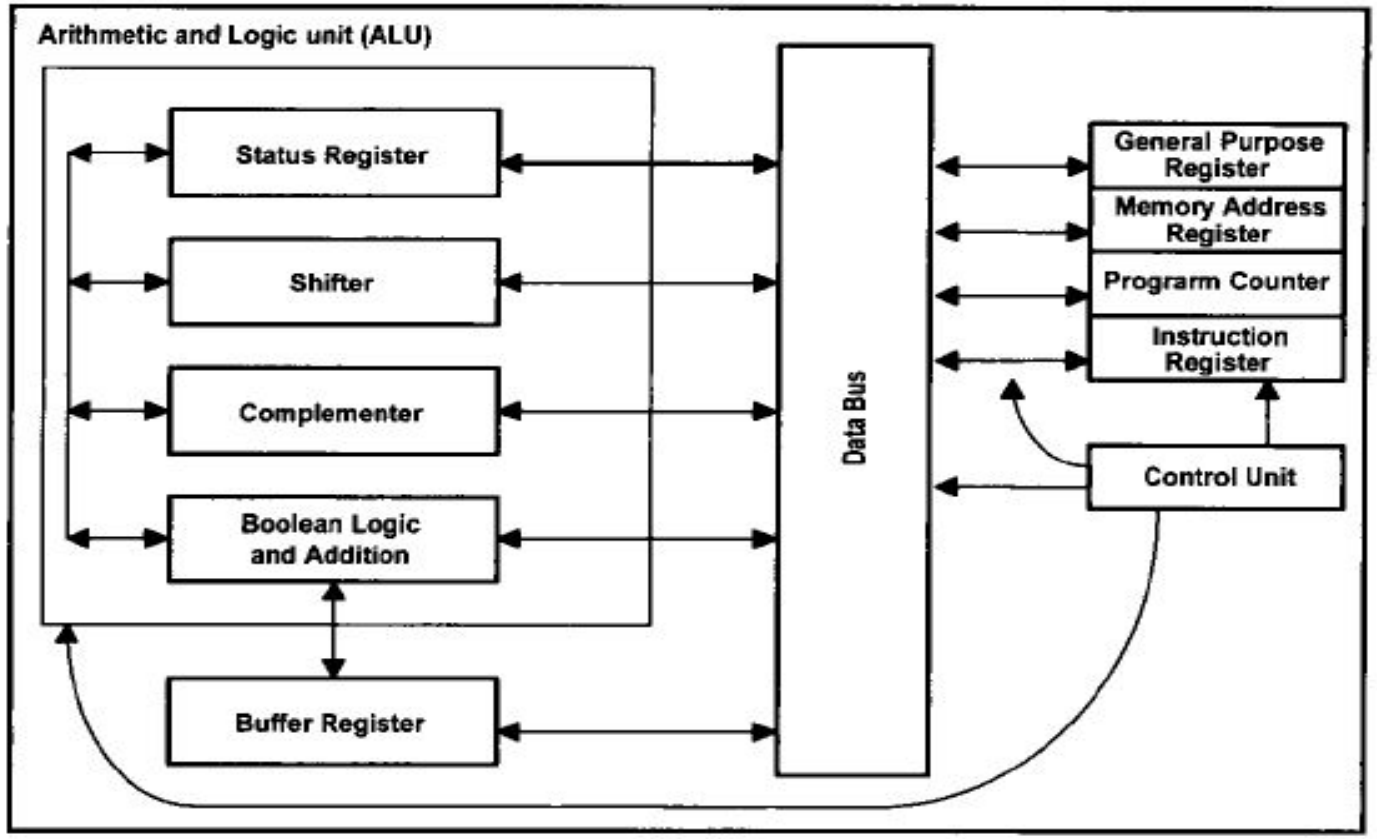
2.3.3 Arithmetic-Logic Unit

- The ALU performs all the data manipulations, such as arithmetic and logic operations, inside a microprocessor. The size of the ALU conforms to the word length of the microcomputer.
- ALU Functions:

1. Binary addition and logic operations
2. Finding the one's complement of data
3. Shifting or rotating the contents of a general-purpose register 1 bit to the left or right through a carry

2.3.4 Functional Representations of Simple and Typical Microprocessors

- Simple Micro



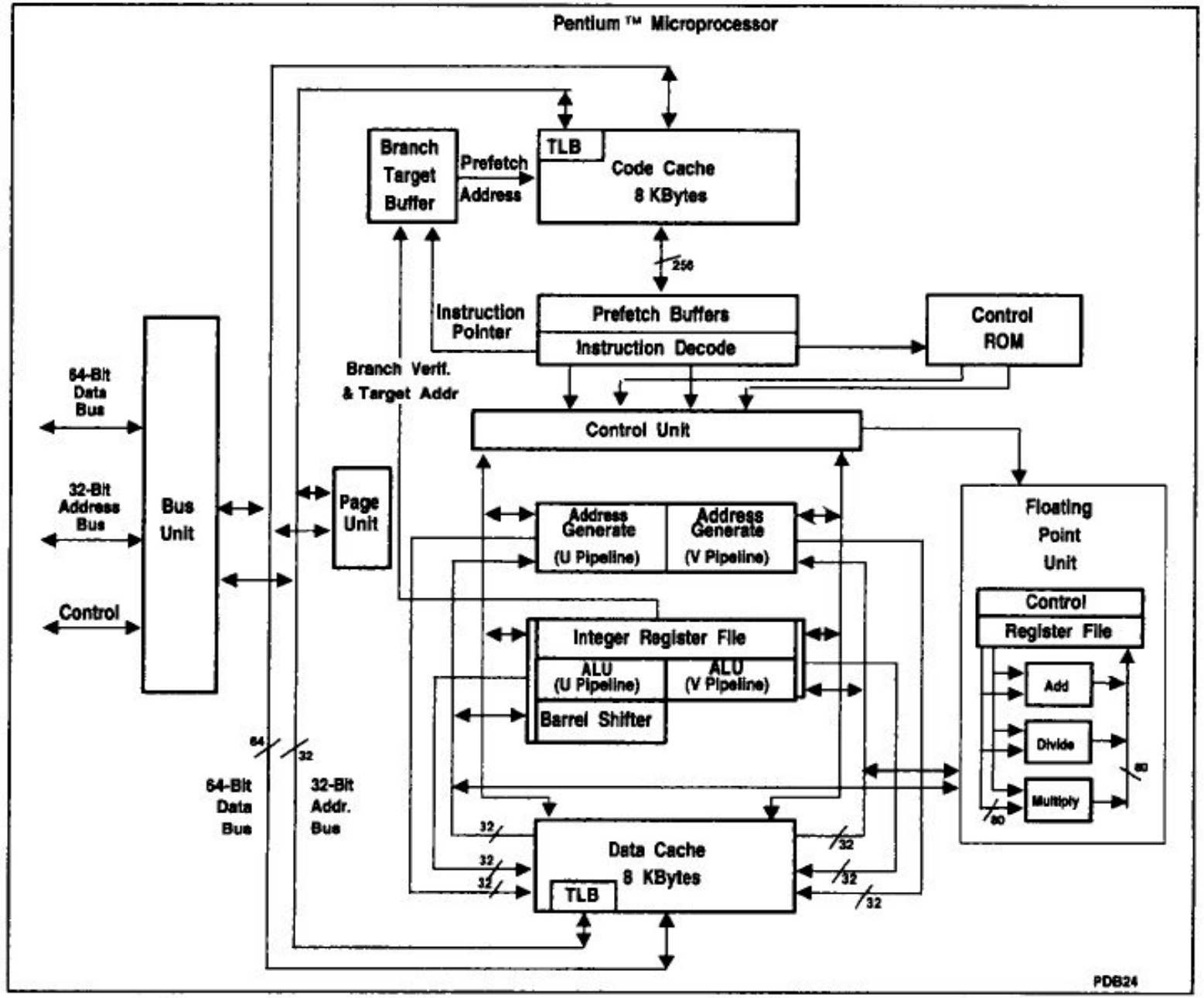
(a) Simple microprocessor

2.3.4 Functional Representations of Simple and Typical Microprocessors

- Buffer Register : Stores any data read from memory for further processing by the ALU.

2.3.4 Functional Representations of Simple and Typical Microprocessors

- Typical Microprocessor



(b) Pentium Microprocessor

- The Pentium contains two instruction pipelines: the U-pipe and the V-pipe. The U-pipe can execute all integer and floating-point instructions. The V-pipe can execute simple integer instructions
- The Pentium contains two separate cache memories: code cache and data cache.

2.3.5 Simplified Explanation of Control Unit design

- The control unit performs two basic operations:

1. instruction interpretation
2. and instruction sequencing.

2.3.5 Simplified Explanation of Control Unit design

- There are two methods for designing a control unit:

hardwired control	Microprogrammed control(firmware)
clocked sequential circuit.	ROM inside the control unit (*control memory)* more expensive flexibility

2.3.5 Simplified Explanation of Control Unit design

- How incrementing the contents of the register by 1 is done in microprogramming

control ??

(see figures in next slides)

2.3.5 Simplified Explanation of Control Unit design

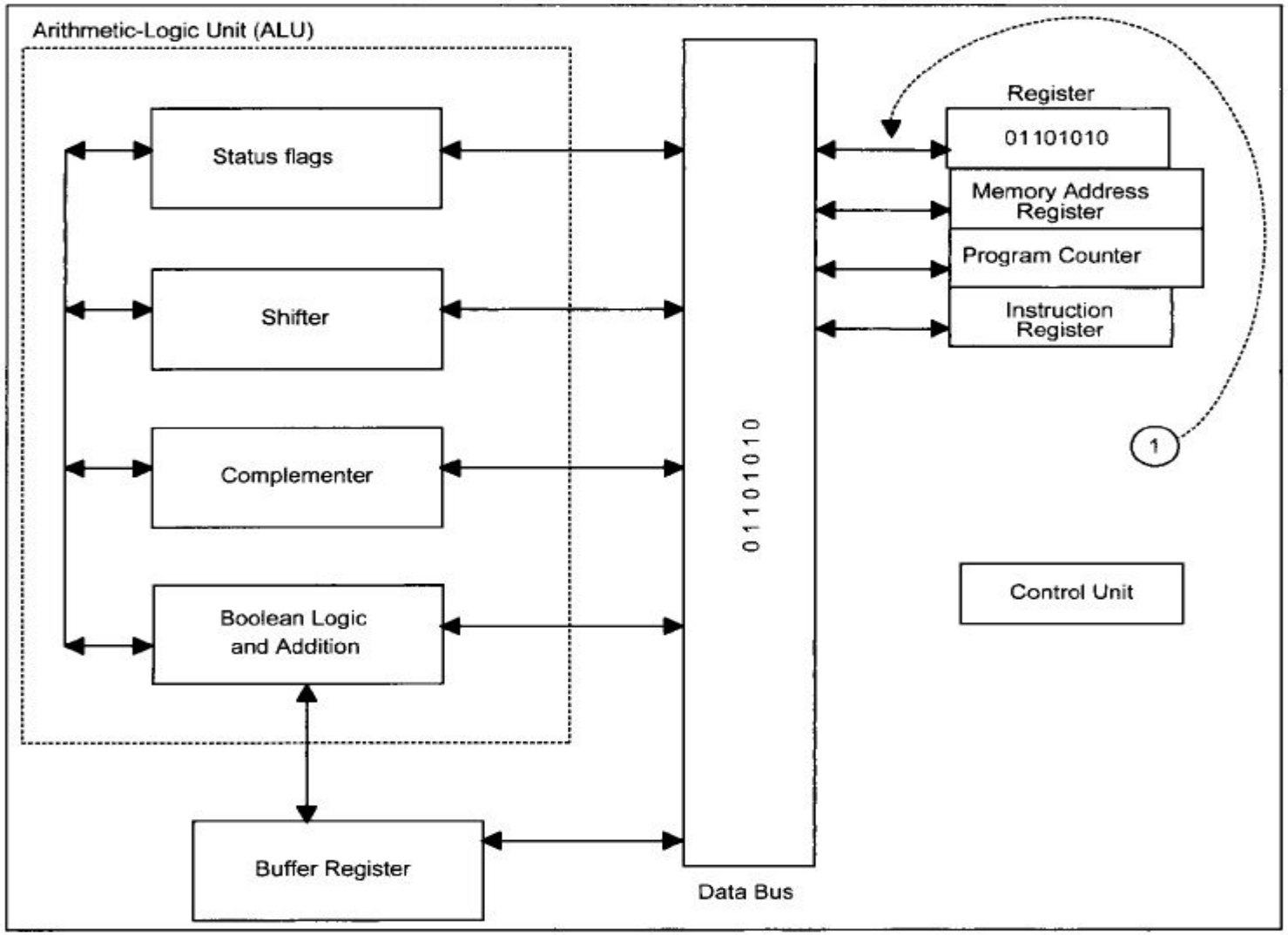


FIGURE 2.10 **Transferring register contents to a data bus.**

2.3.5 Simplified Explanation of Control Unit design

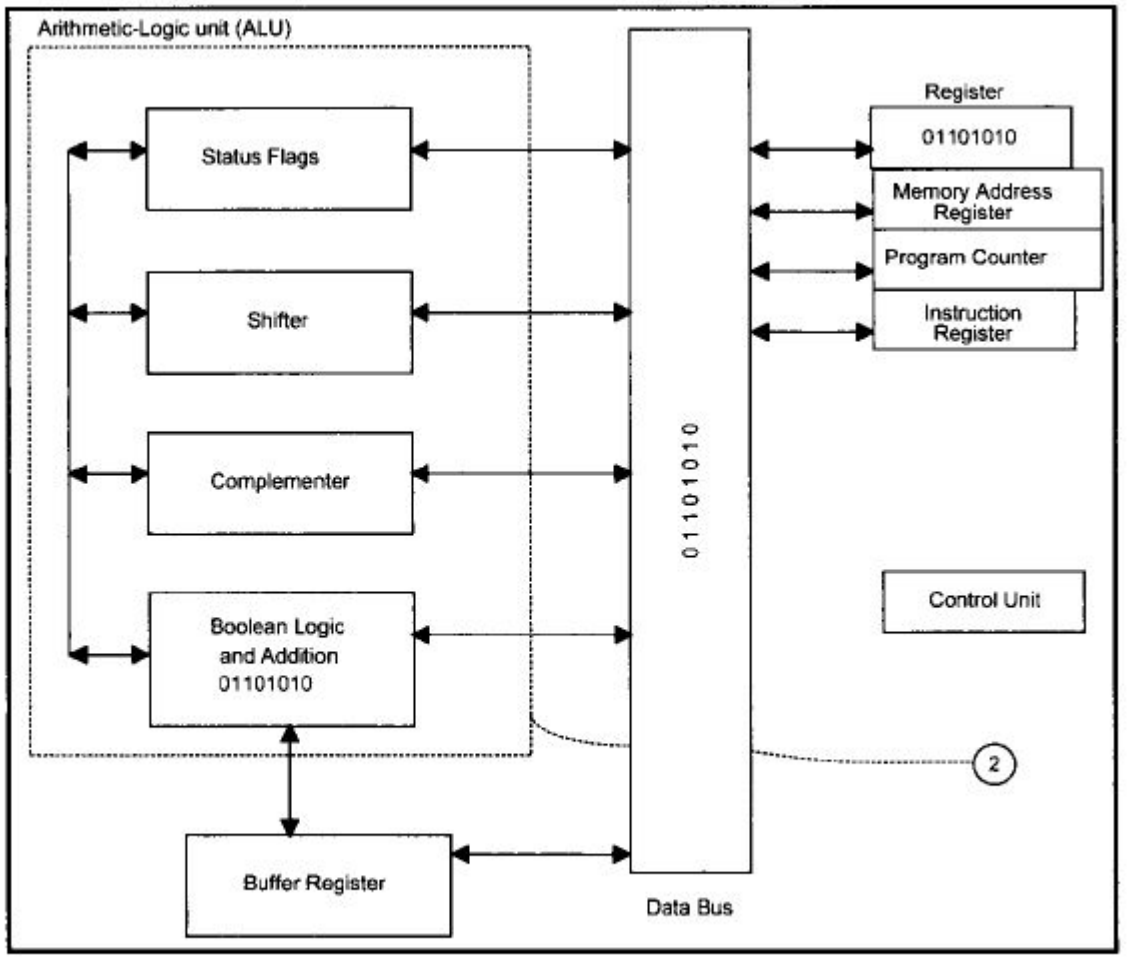


FIGURE 2.11 Transferring data bus contents to an ALU.

2.3.5 Simplified Explanation of Control Unit design

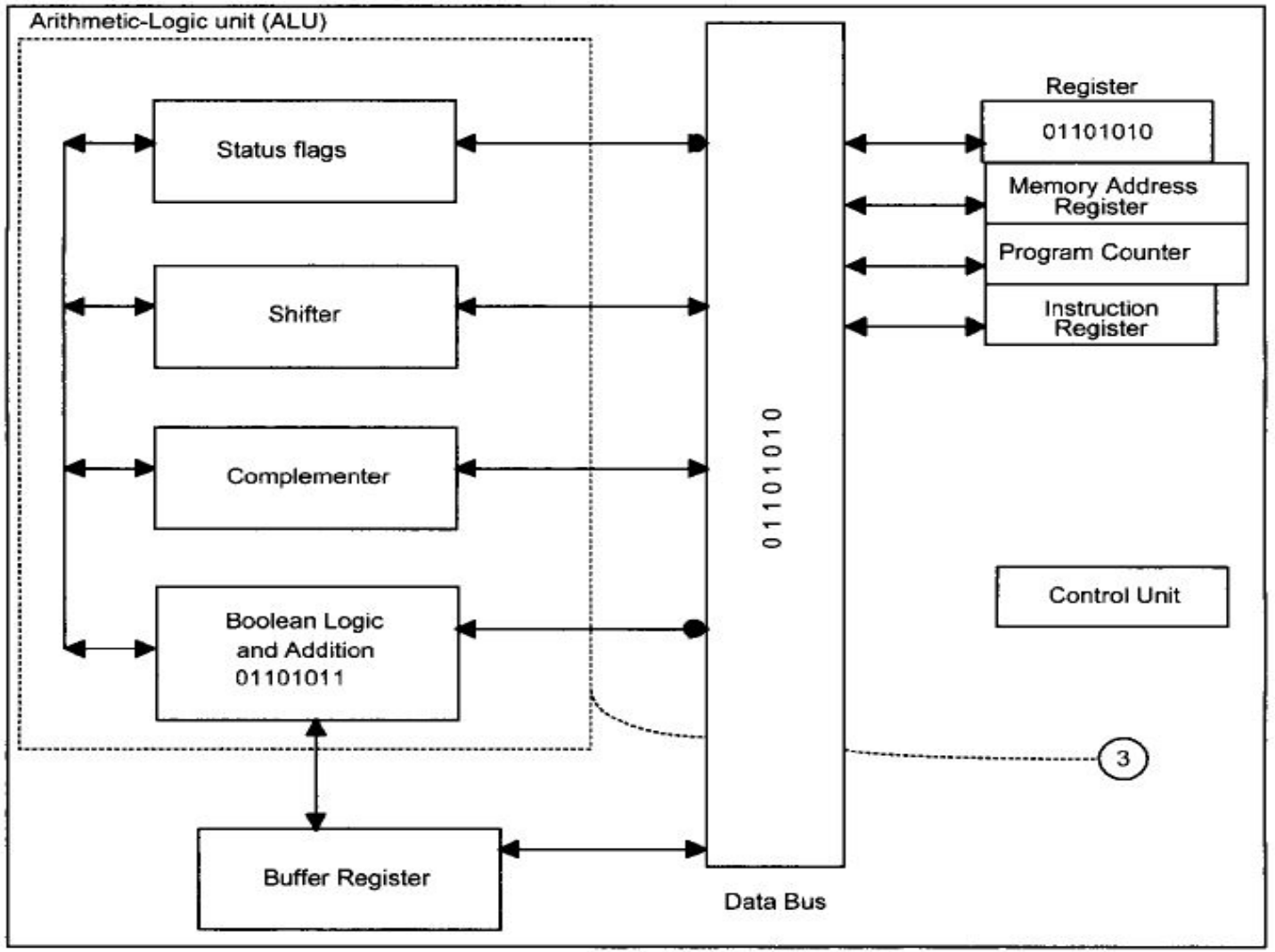


FIGURE 2.12 Activating the ALU logic.

2.3.5 Simplified Explanation of Control Unit design

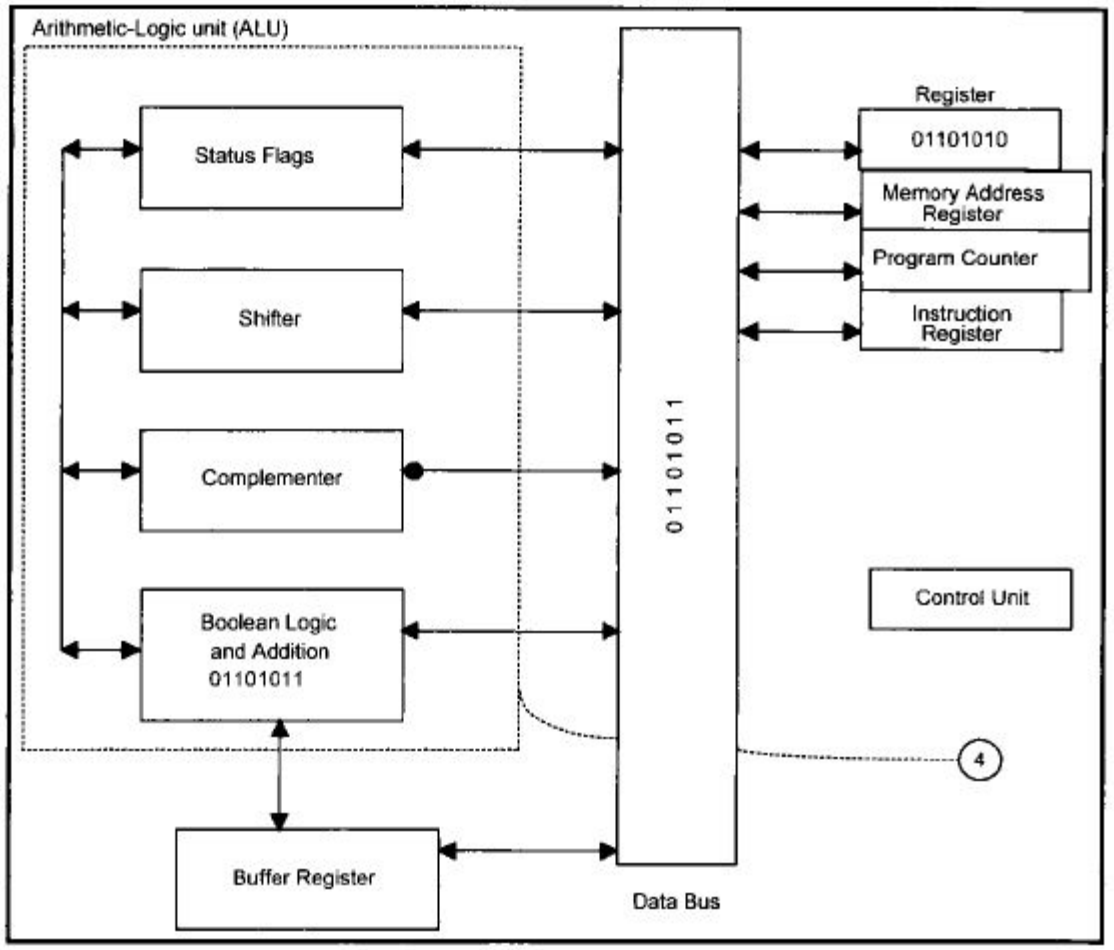


FIGURE 2.13 Transferring an ALU result to a data bus.

2.3.5 Simplified Explanation of Control Unit design

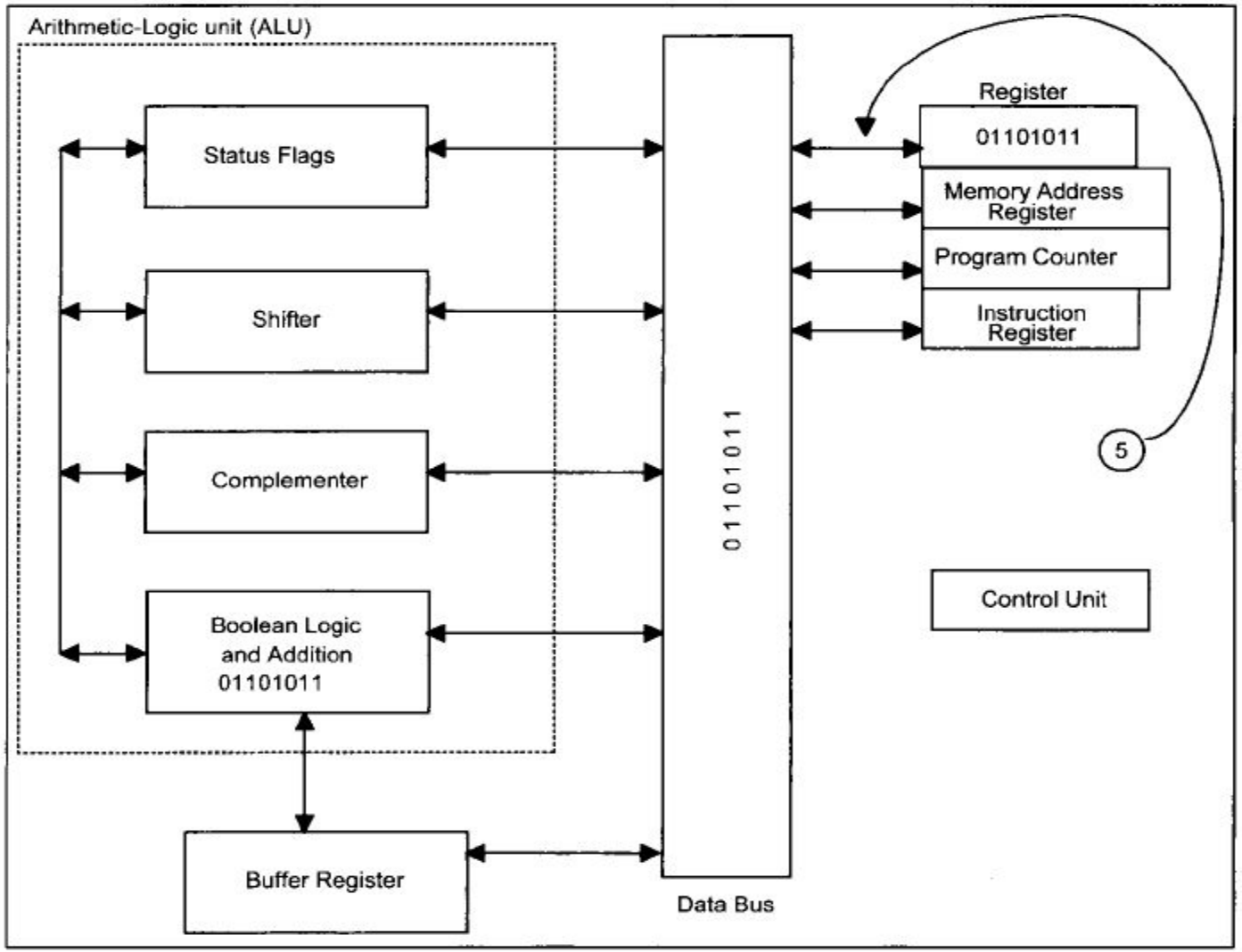


FIGURE 2.14 **Transferring a data bus.**

2.4 Program Execution by Conventional Microprocessors

- The following three steps for completing the instruction:

1. *Fetch. The microprocessor fetches (instruction read) the instruction from the main* memory (external to the microprocessor) into the instruction register.
2. *Decode. The microprocessor decodes or translates the instruction using the control unit.* The control unit inputs the contents of the instruction register, and then decodes (translates) the instruction to determine the instruction type.
3. *Execute. The microprocessor executes the instruction using the control unit. To* accomplish the task, the control unit generates a number of enable signals required by the instruction.

2.4 Program Execution by Conventional Microprocessors

- For example, suppose that it is desired to add the contents of two registers, **X** and *Y, and store the result in register Z. To accomplish this, a conventional microprocessor* performs the following steps:

1. The microprocessor fetches the instruction into the instruction register.
2. The control unit (CU) decodes the contents of the instruction register.
3. The CU executes the instruction by generating enable signals for the register and **ALU** sections to perform the following:

   a. The CU transfers the contents of registers **X and *Y from the Register section into the*** ALU.

   b. The CU commands the **ALU to ADD.**

   c. The CU transfers the result from the ALU into register Z of the register section.

2.5 Program Execution by typical 32-bit Microprocessors

- Enhancement in 32-bit microprocessors (like Pentium) include : *cache memory*, *memory management*, *pipelining*, *floating-point arithmetic*, and *branch prediction*.
- Cache memory is a high-speed read/write memory implemented as on-chip hardware in typical 32-bit microprocessors in order to increase processing rates. This topic is covered in more detail in Chapter 3.

2.5 Program Execution by typical 32-bit Microprocessors

- **Memory management** allows programmers to write programs much larger than those that could fit in the main memory space available to the microprocessors; the programs are simply stored on a secondary device, such as a hard disk. This topic is covered in more detail in Chapter 3.

2.5.1 Pipelining

- Basic Concept

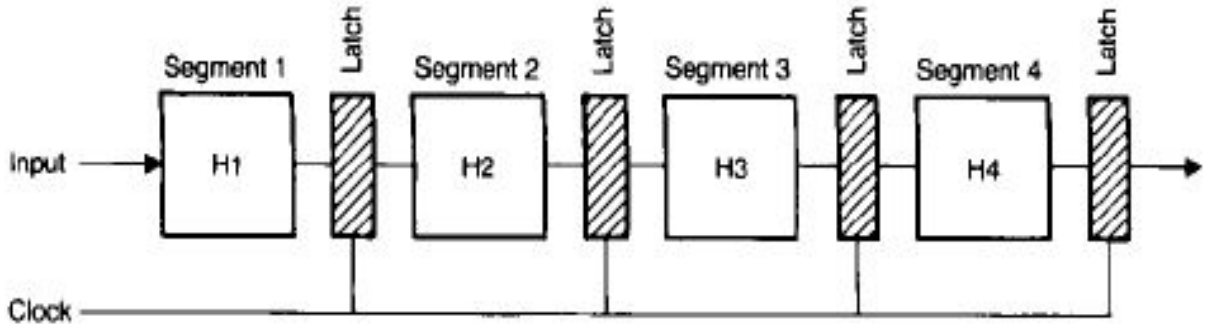


FIGURE 2.15 Four-segment pipeline.

Hi is Hardware designed to perform activity Ai

2.5.1 Pipelining

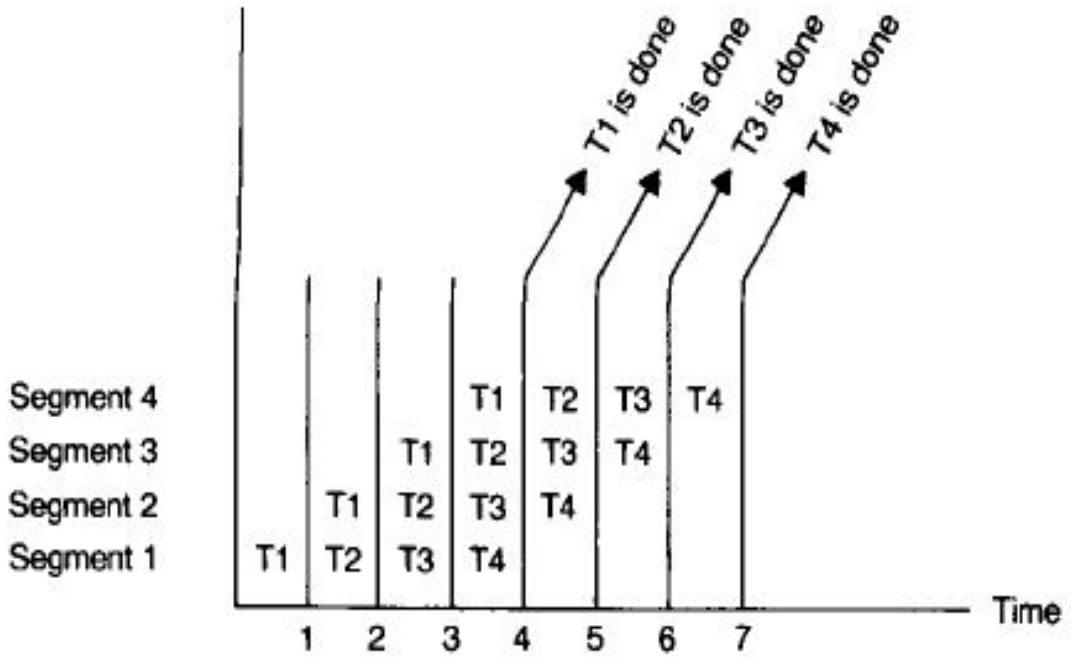


FIGURE 2.16 **Overlapped execution of four tasks using a pipeline.**

2.5.1 Pipelining

- Two Kind of Pipelining:

Arithmetic operations and instruction execution.

2.5.1 Pipelining

- **Arithmetic Pipelines**
- Consider the process of adding two floating-point numbers **x =**0.9234 * **10^4 and *y = 0.48 * 10 ^{2}.***

First: exponents of **x and *y are unequal.***

Second: exponent alignment.

Third: Perform the addition

Fourth: Normalize the final answer

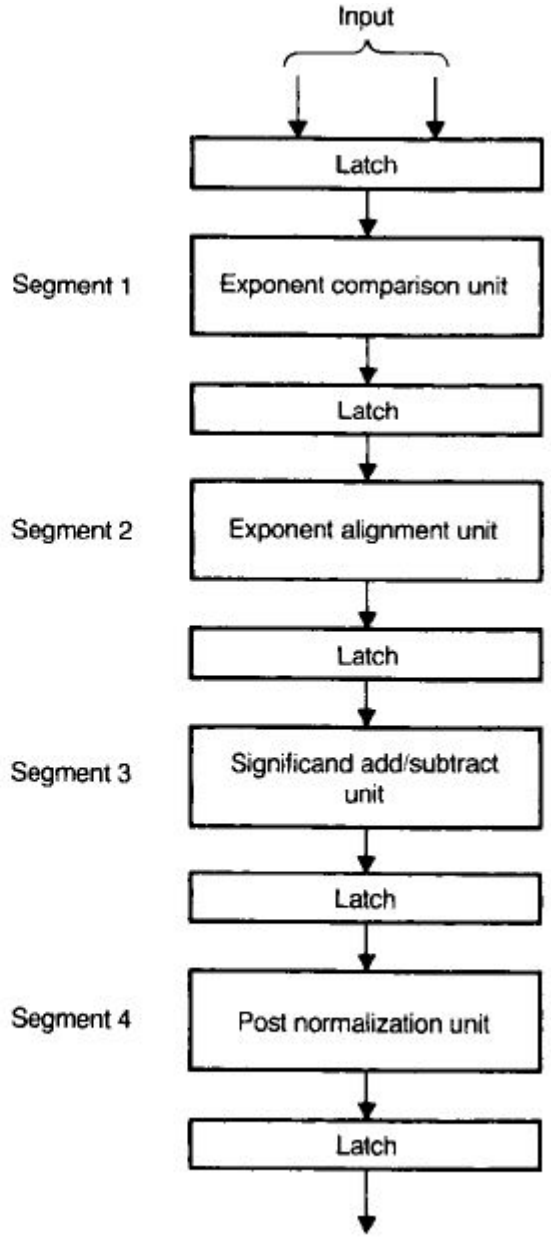


FIGURE 2.17 Pipelined floating-point add/subtract unit.

2.5.1 Pipelining

- **Instruction Pipelines**

Instruction cycle typically involves the

following activities:

- 1. Instruction fetch - needs five clocks to complete
- 2. Instruction decode
- 3. Operand fetch (Data Read)
- 4. Operation execution
- 5. Result routing.

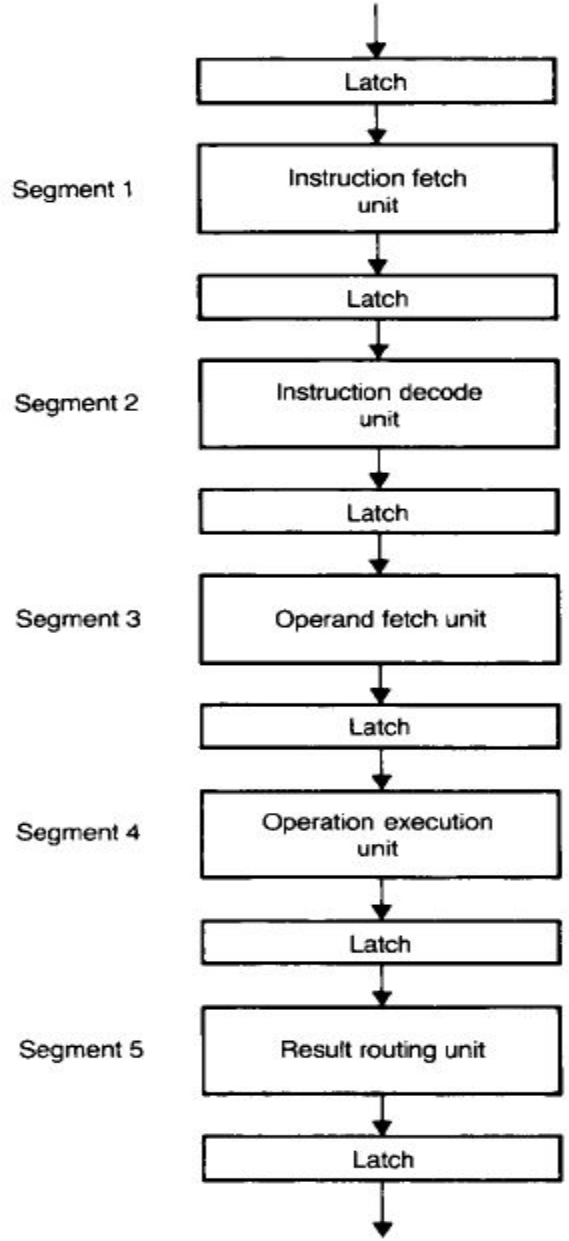


FIGURE 2.18 **Five-segment instruction pipeline.**

2.5.1 Pipelining

- Example of the execution of a stream of five instructions: 11,12,13,14, and 15, in which I3 is a conditional branch instruction.

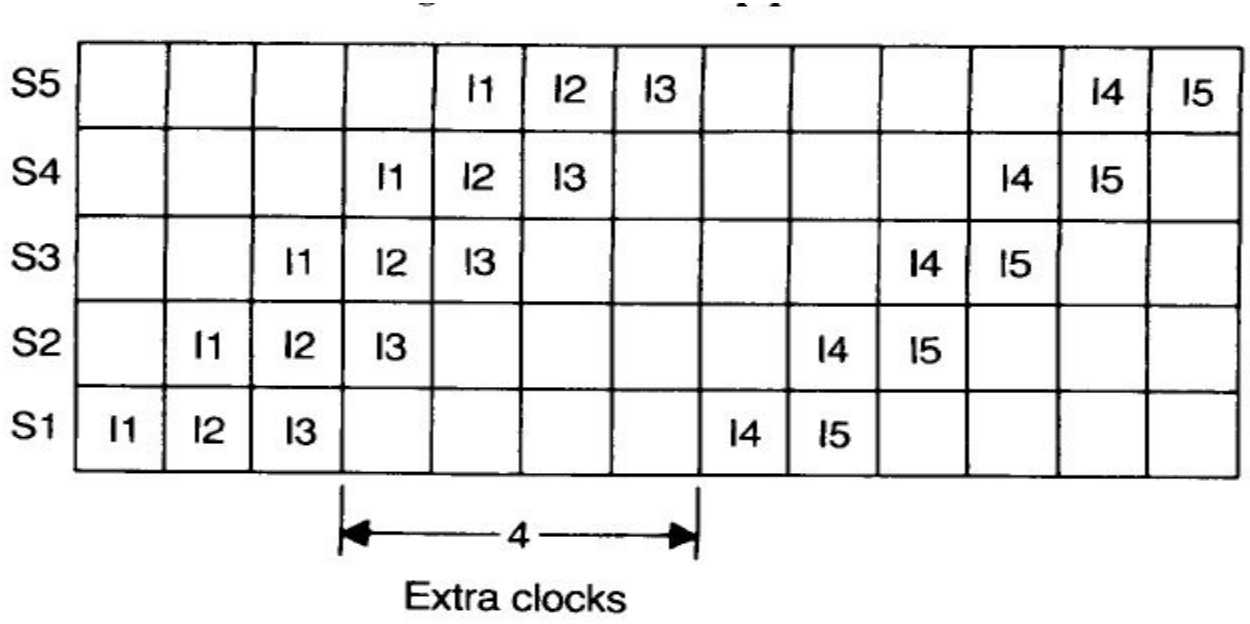


FIGURE 2.19 **Pipelined execution of a stream of five instructions that includes a branch instruction.**

2.5.2 Branch Prediction Feature

- This allows these microprocessors to anticipate jumps of the instruction flow ahead of time.

2.5.2 Branch Prediction Feature

- To accomplish this, the Pentium includes on-chip hardware called the *Branch Unit* (BU). The BU contains the branch execution unit (BEU) and the branch prediction unit (BPU). Whenever the Pentium encounters a conditional branch instruction, it sends it to the BU for execution. The BU evaluates the instruction's branch condition using the BEU and determines whether the branch should or should not be taken. Once the BU determines the branch condition, it calculates the starting address (Branch target) of the next block of code to be executed. The Pentium then starts fetching code at the new address.

2.6 Scalar and Superscalar Microprocessors

- Scalar processors such as the 80486 can execute one instruction per cycle.

The 80486 contains only one pipeline.

- Superscalar microprocessors, can execute

more than one instruction per cycle. These microprocessors contain more than one pipeline.

- The Pentium, a superscalar microprocessor, contains two independent pipelines. This

allows the Pentium to execute two instructions per cycle.

2.7 RISC vs. CISC

- There are two types of microprocessor architectures: RISC and CISC.
- RISC stand for (reduced instruction set computer) and CISC for (complex instruction set computer)

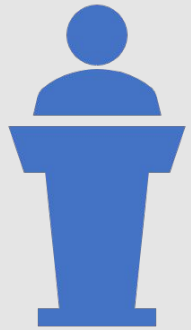
2.7 RISC vs. CISC

CISC	RISC
large number of instructions and many addressing modes	a simple instruction set with a few addressing modes
slower clock rate	fast clock rate
complex control unit, thus requiring microprogrammed implementation.	hardwired control Unit
more difficult to pipeline;	more efficient pipelining.
complex programs require fewer instructions in CISC	RISC requires a large number of instructions to accomplish the same task

2.7 RISC vs. CISC

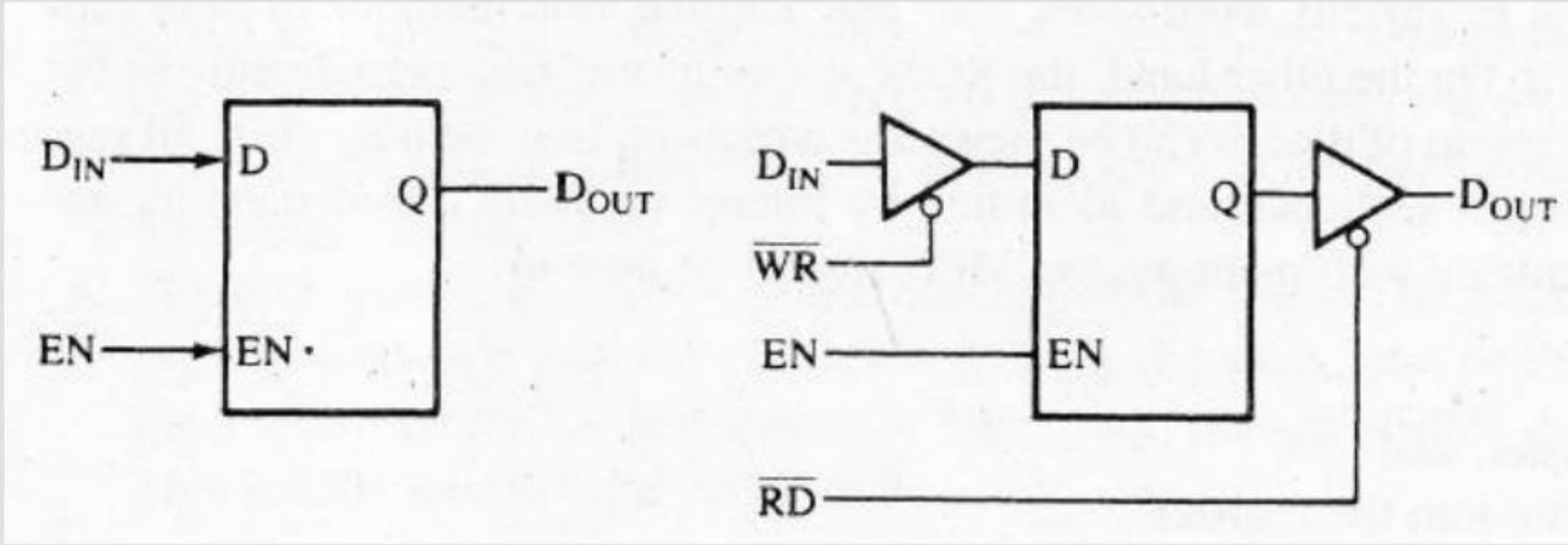
- Intel’s original Pentium is a CISC microprocessor. Intel Pentium Pro and other succeeding members of the Pentium family and Motorola 68060 use a combination of RISC and CISC architectures for providing high performance. The Pentium Pro and other succeeding

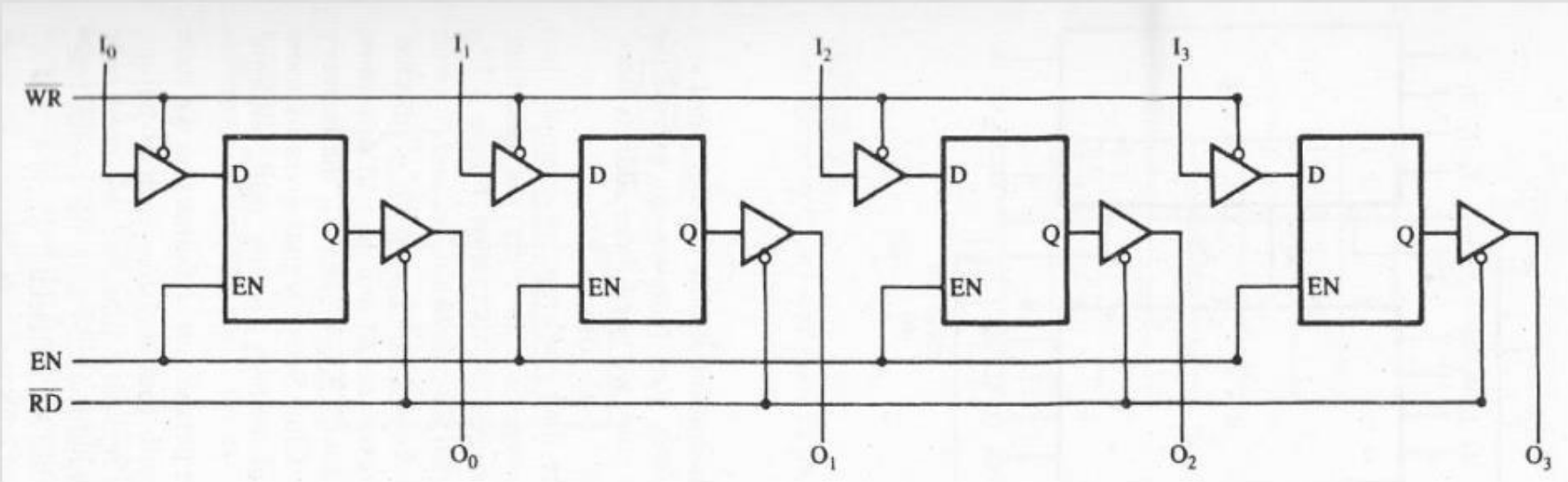
LECTURE 12

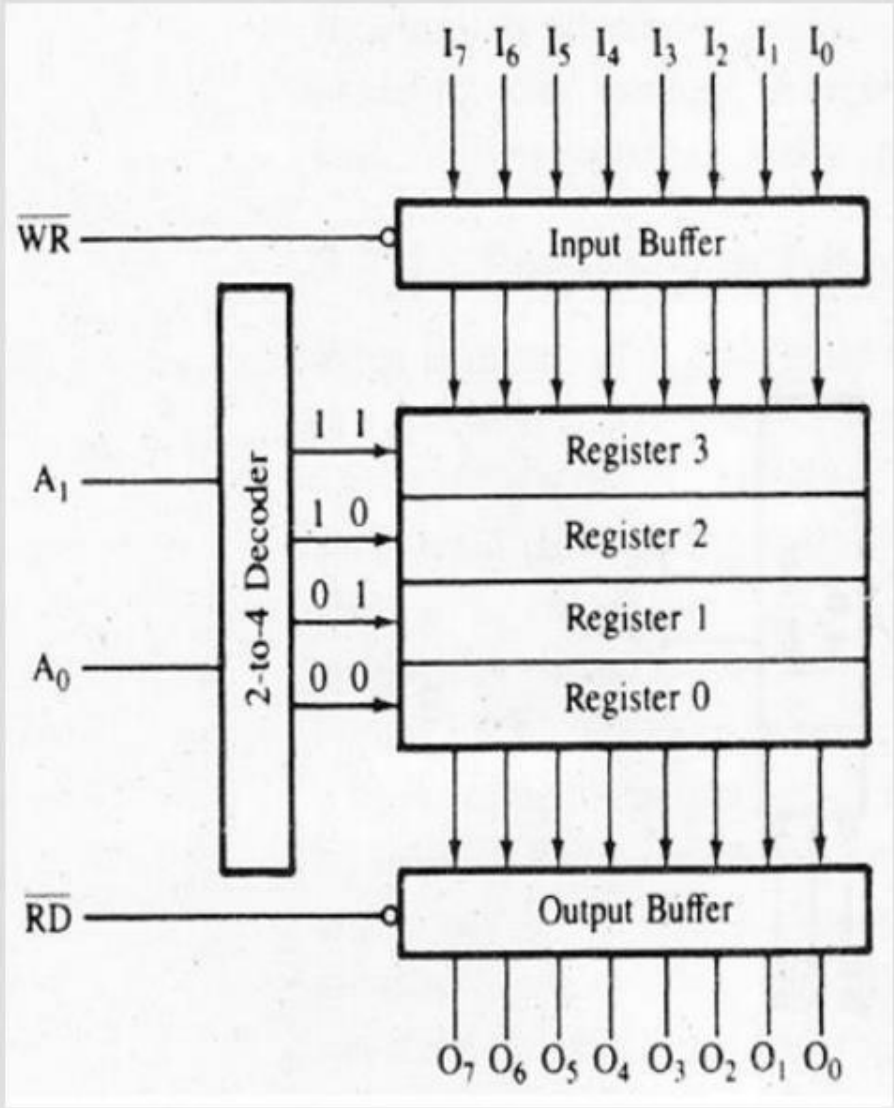


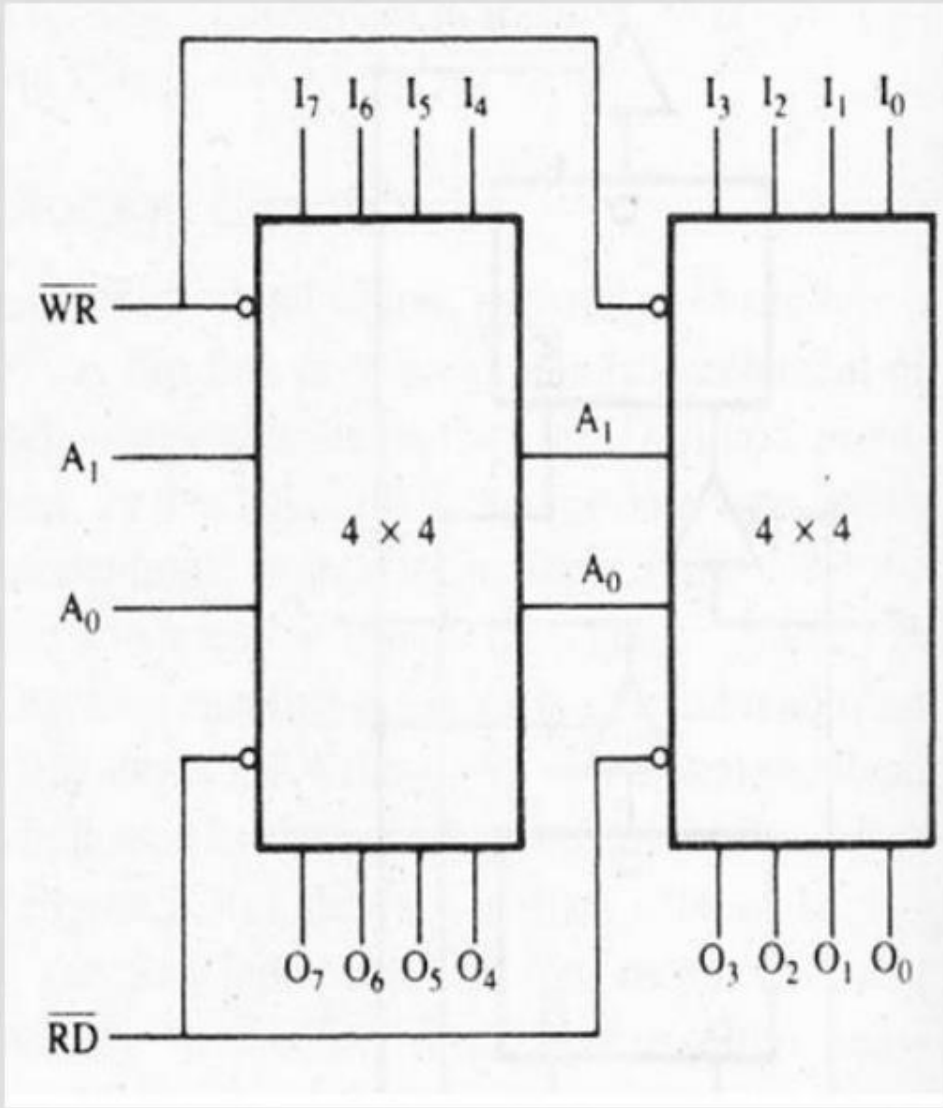
MICROPROCESSOR PIN DIAGRAM & MEMORY INTERFACING

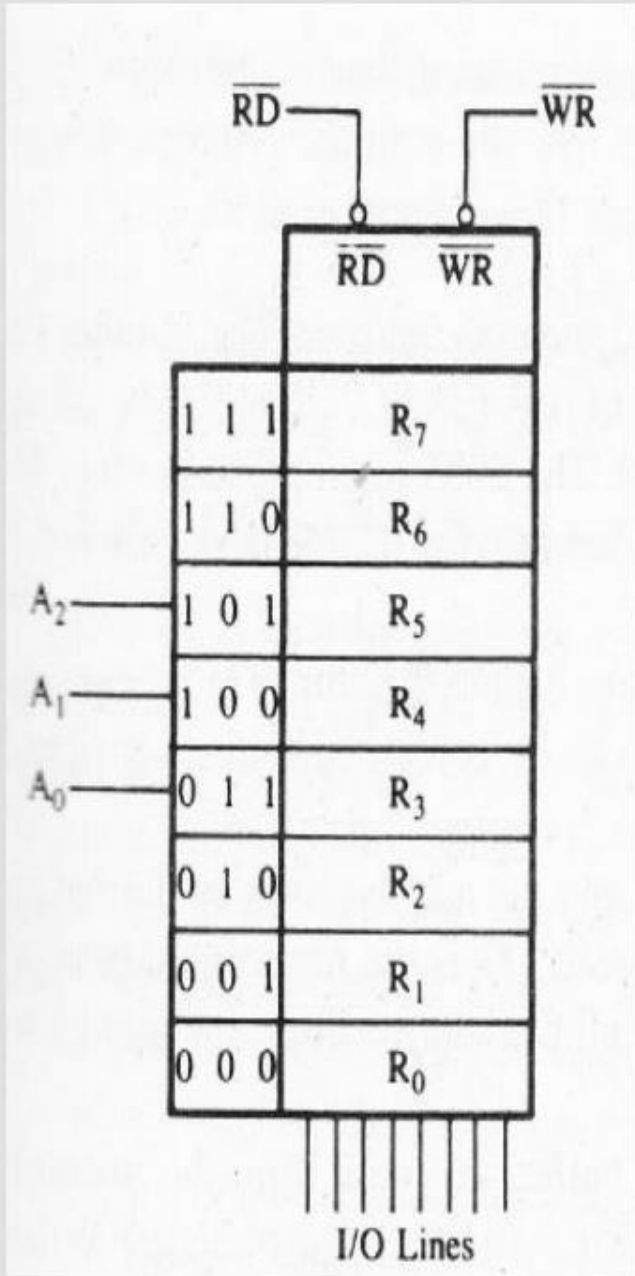
Flip-Flop or Latch as a Storage Element

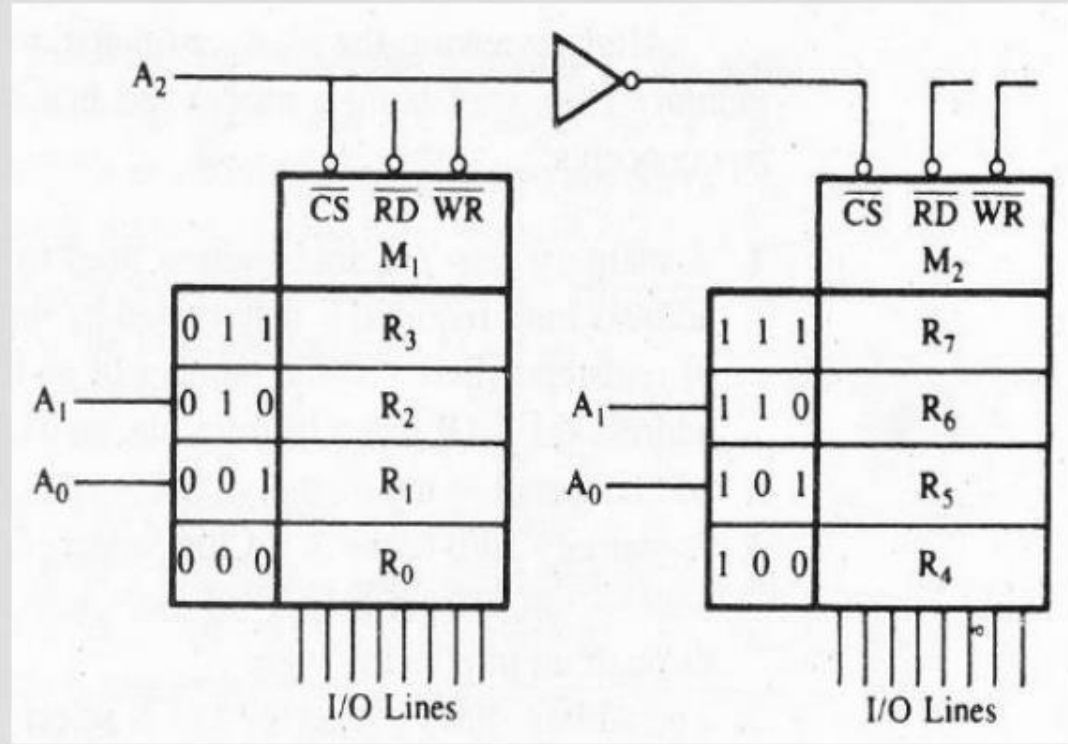


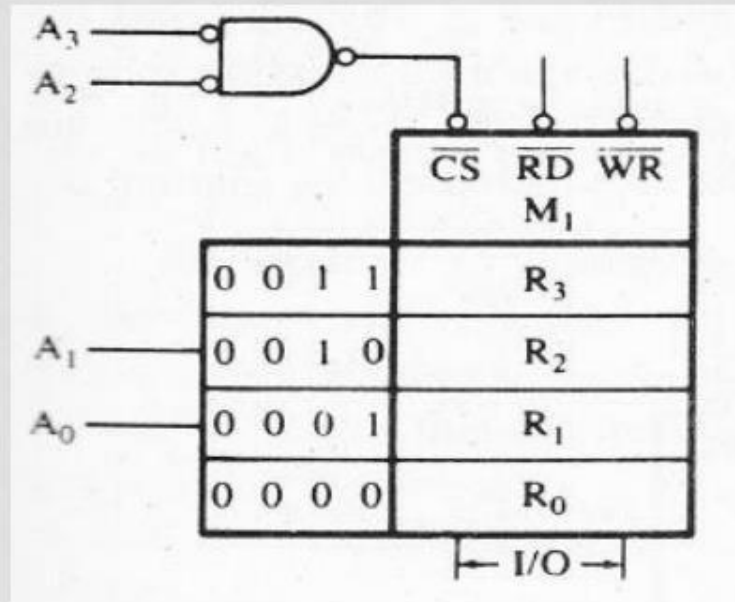



I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0
$\overline{WR}$
Input Buffer
A_1
A_0
2-to-4 Decoder
1 1
1 0
0 1
0 0
Register 3
Register 2
Register 1
Register 0
$\overline{RD}$
Output Buffer
O_7 O_6 O_5 O_4 O_3 O_2 O_1 O_0


I_7 I_6 I_5 I_4
I_3 I_2 I_1 I_0
$\overline{WR}$
A_1
A_0
4 × 4
A_1
A_0
4 × 4
O_7 O_6 O_5 O_4
O_3 O_2 O_1 O_0
$\overline{RD}$


RD
WR
RD WR
1 1 1 R7
1 1 0 R6
A2 1 0 1 R5
A1 1 0 0 R4
A0 0 1 1 R3
0 1 0 R2
0 0 1 R1
0 0 0 R0
I/O Lines


A2
CS RD WR
M1
0 1 1 R3
A1 0 1 0 R2
A0 0 0 1 R1
0 0 0 R0
I/O Lines
CS RD WR
M2
1 1 1 R7
A1 1 1 0 R6
A0 1 0 1 R5
1 0 0 R4
I/O Lines


A3
A2
CS RD WR
M1
0 0 1 1 R3
A1 0 0 1 0 R2
A0 0 0 0 1 R1
0 0 0 0 R0
I/O

- **The Requirements of a memory chip**

✓ A memory chip requires address lines to identity a memory register.

✓ The number of address lines required is determined by the number of registers in a chip

✓ (2^n = number of registers where n is the number of address lines).

- A memory chip requires a Chip Select (CS) signal to enable the chip. The remaining address lines of the microprocessor can be connected to the CS signal through an interfacing logic.
- The address lines connected to CS select the chip, and the address lines connected to the address lines of the memory chip select the register.
- The control signal Read (RD) enables the output buffer, and data from the selected register are made available on the output lines.
- The control signal (WR) enables the input buffer, and data on the input lines are written into memory cells.

• The Requirements of a memory chip

A memory chip requires address lines to identity a memory register.

The number of address lines required is determined by thenumber of registers in a chip(2n = number of registers where n is the number of addresslines).A memory chip requires a Chip Select (CS) signal to enablethe chip. The remaining address lines of themicroprocessor can be connected to the CS signalthrough an interfacing logic.The address lines connected to CS select the chip, and theaddress lines connected to the address lines of the memorychip select the register.° The control signal Read (RD) enables the output buffer,and data from the selected register are made available onthe output lines.° The control signal (WR) enables the input buffer, and dataon the input lines are written into memory cells.

- **INPUT AND OUTPUT (I/O) DEVICES :**
- Input/output devices are the means through which the MPU communicates with "the outside world.“
- There are two different methods by which I/O devices can be identified.

1. I/Os with 8-Bit Addresses (Peripheral-Mapped I/O)

- The steps in communicating with an I/O device

✓ The MPU places an 8-bit address on the address bus. which is decoded by external decode logic.

✓ The MPU sends a control signal (I/O Read or I/O Write) and enables the I/O device.

✓ Data are transferred using the data bus.

2. I/Os with I6-Bit Addresses (Memory-Mapped I/O)

✓ the MPU uses l6 address lines to identify an I/O device. This is known as memory-mapped I/O.

• INPUT AND OUTPUT (I/O) DEVICES :

• Input/output devices are the means through which theMPU communicates with "the outside world.“

• There are two different methods by which I/O devices canbe identified.1. I/Os with 8-Bit Addresses (Peripheral-Mapped I/O)

• The steps in communicating with an I/O deviceThe MPU places an 8-bit address on the address bus.which is decoded by external decode logic.The MPU sends a control signal (I/O Read or I/O Write)and enables the I/O device.Data are transferred using the data bus.2. I/Os with l6-Bit Addresses (Memory-Mapped I/O)the MPU uses 16 address lines to identify an I/O device.This is known as memory-mapped I/O.

- **THE 8085 MPU**

  - The term microprocessing unit (MPU) is similar to the term central processing unit (CPU) used in traditional computers.

- **MicroProcessing Unit (MPU)**

  A device or a group of devices (as a unit) that can communicate with peripherals, provide timing signals, direct data flow, and perform computing tasks as specified by the instructions in memory.

- **The 8085 microprocessor can almost qualify as an MPU with the following two limitations.**

1. The low-order address bus of the 8085 microprocessor is multiplexed (time-shared) with the data bus. The buses need to be demultiplexed.
2. Appropriate control signals need to be generated to interface memory and I/O with the 8085.

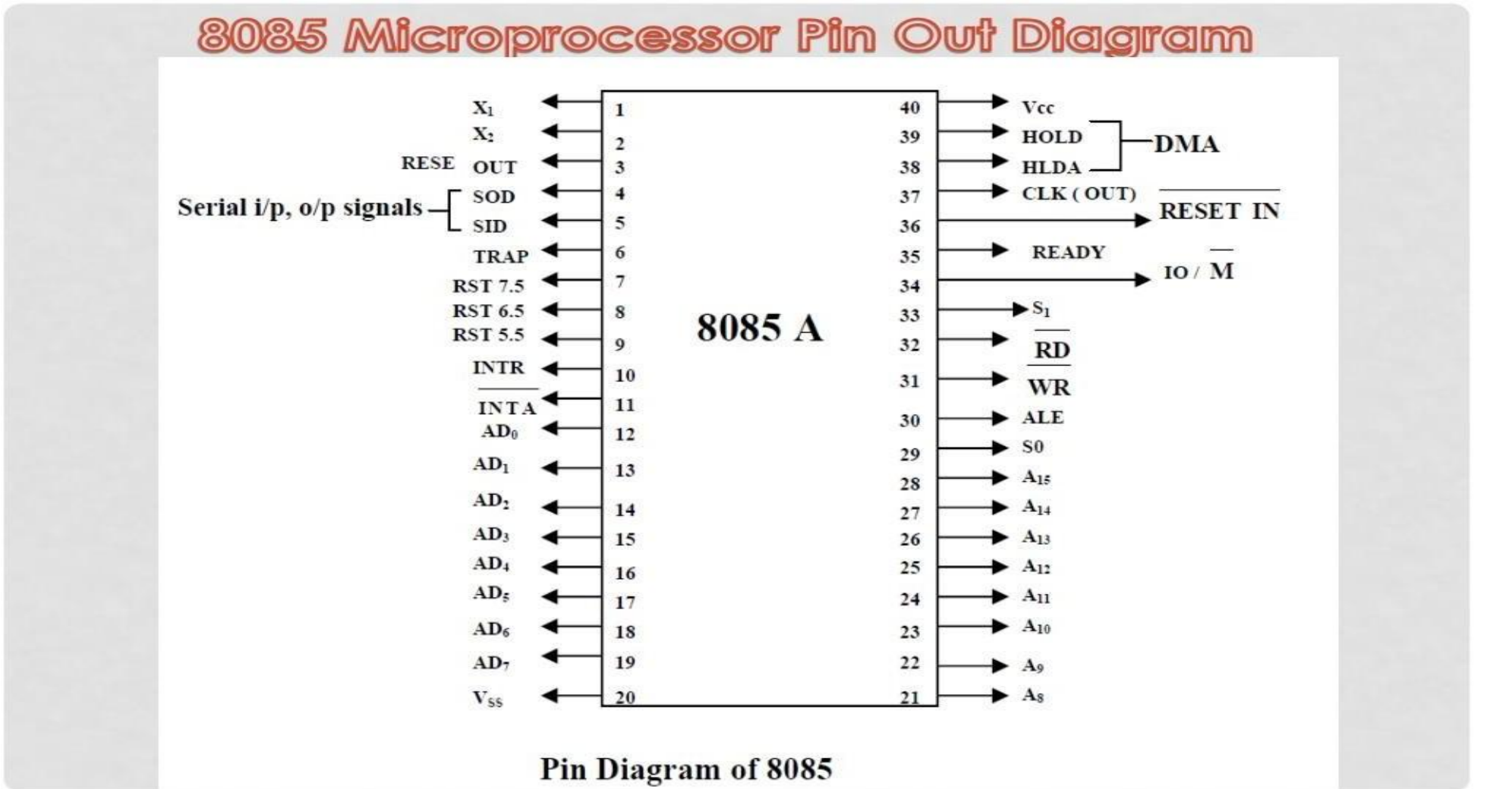
- • THE 8085 MPUoThe term microprocessing unit (MPU) is similar to theterm central processing unit (CPU) used in traditionalcomputers.MicroProcessing Unit (MPU)A device or a group of devices (as a unit) that cancommunicate with peripherals, provide timing signals,direct data flow, and perform computing tasks asspecified by the instructions in memory.The 8085 microprocessor can almost qualify as an MPUwith the following two limitations.1. The low-order address bus of the 8085 microprocessor ismultiplexed (time-shared) with the data bus. The busesneed to be demultiplexed.2. Appropriate control signals need to be generated tointerface memory and I/O with the 8085.

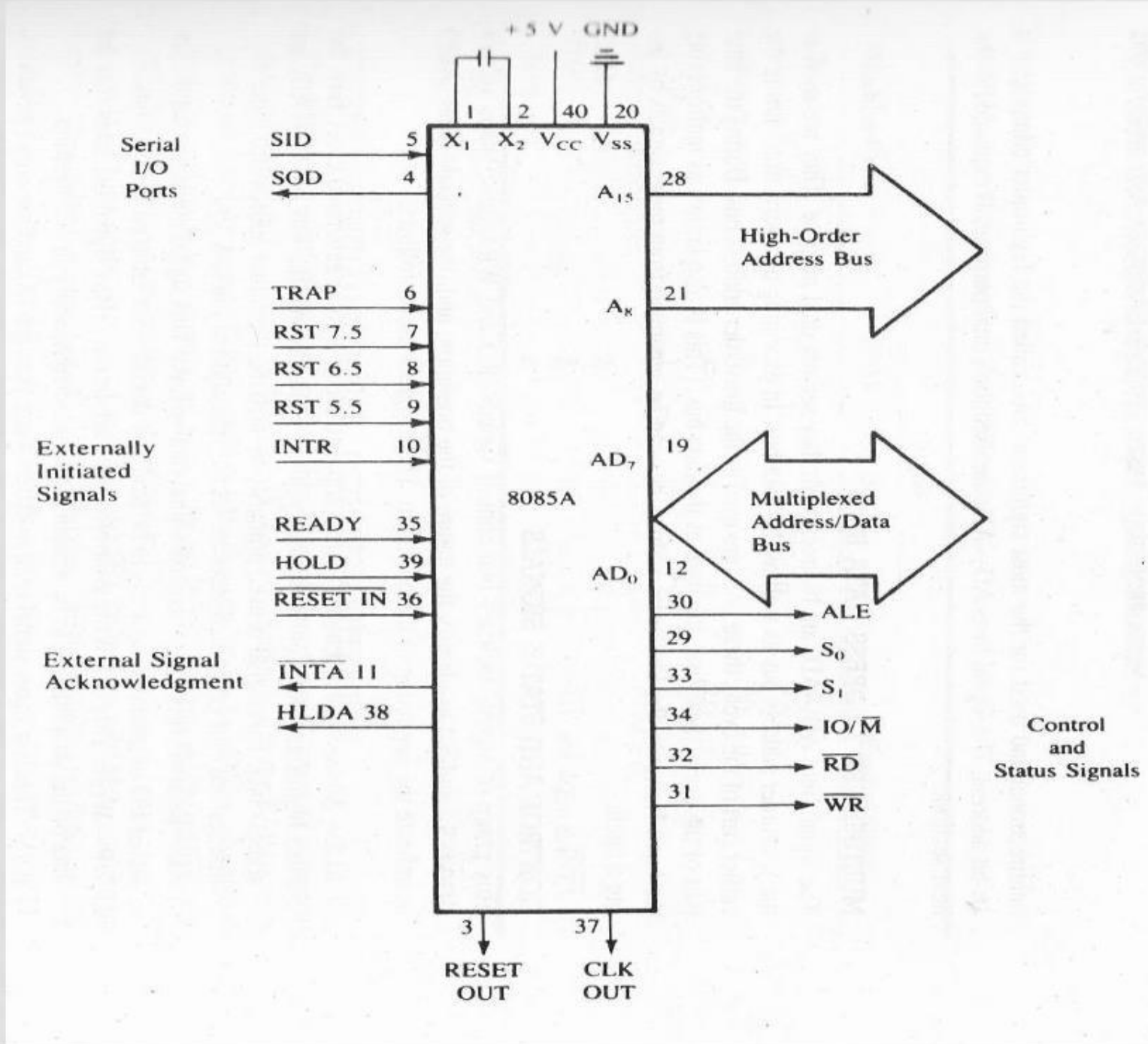
• THE 8085 AND ITS PIN DESCRIPTION

- The 8085 is an 8-bit general purpose microprocessor that can address 64K Byte of memory.
- It has 40 pins and uses +5V for power. It can run at a maximum frequency of 3 MHz.
- The pins on the chip can be grouped into 6 groups:
  - ✓ Address Bus and Multiplexed Data Bus. (16 Pins)
  - ✓ Control and Status Signals. (6 Pins)
  - ✓ Power supply and frequency. (4 Pins)
  - ✓ Externally Initiated Signals. (7 Pins)
  - ✓ Interupt Signals. (5 pins)
  - ✓ Serial I/O ports. (2 Pins)

- • THE 8085 AND ITS PIN DESCRIPTION• The 8085 is an 8-bit general purpose microprocessorthat can address 64K Byte of memory.• It has 40 pins and uses +5V for power. It can run at amaximum frequency of 3 MHz.• The pins on the chip can be grouped into 6 groups:Address Bus and Multiplexed Data Bus. (16 Pins)Control and Status Signals. (6 Pins)Power supply and frequency. (4 Pins)Externally Initiated Signals. (7 Pins)Interupt Signals. (5 pins)Serial I/O ports. (2 Pins)

8085 Microprocessor Pin Out Diagram




+5 V
GND
1
2
40
20
X_1
X_2
V_{CC}
V_{SS}
Serial I/O Ports
SID 5
SOD 4
TRAP 6
RST 7.5 7
RST 6.5 8
RST 5.5 9
Externally Initiated Signals
INTR 10
READY 35
HOLD 39
RESET IN 36
External Signal Acknowledgment
INTA 11
HLDA 38
8085A
A_{15} 28
A_8 21
High-Order Address Bus
AD_7 19
AD_0 12
Multiplexed Address/Data Bus
ALE 30
S_0 29
S_1 33
IO/$\overline{M}$ 34
$\overline{RD}$ 32
$\overline{WR}$ 31
Control and Status Signals
3
RESET OUT
37
CLK OUT

- **Control and Status Signals :-**

  - **ALE-Address Latch Enable:**

✓ This is a positive going pulse generated every time the 8085 begins an operation (machine cycle): it indicates that the bits on AD_7-AD_0 are address bits.

✓ This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines. A_7-A_0.

  - **RD-READ :**

✓ This is a Read control signal (active low).

✓ This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.

✓ **WR-WRITE :**

✓ This is a Write control signal (active low).

✓ This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.

- • Control and Status Signals :-o ALE-Address Latch Enable:This is a positive going pulse generated every time the 8085begins an operation (machine cycle): it indicates that thebits on AD7-AD, are address bits.This signal is used primarily to latch the low-order addressfrom the multiplexed bus and generate a separate set ofeight address lines. A7-A0-o RD-READ :This is a Read control signal (active low).This signal indicates that the selected I/O or memorydevice is to be read and data are available on the data bus.WR-WRITE :This is a Write control signal (active low).This signal indicates that the data on the data bus are tobe written into a selected memory or I/O location.

- **IO/M :**

✓ This is a status signal used to differentiate between I/O and memory operations.

✓ When it is high it indicates an I/O operation; when it is low, it indicates a memory operation.

✓ This signal is combined with RD (Read) and WR (Write) to generate I/O and memory control signals.

- **S_1 & S_0 :**

✓ These status signals, similar to IO/M.

✓ They can identify various operations, but they are rarely used in small systems

- IO/M :This is a status signal used to differentiate between I/Oand memory operations.When it is high it indicates an I/O operation; when it islow, it indicates a memory operation.This signal is combined with RD (Read) and WR (Write) togenerate I/O and memory control signals.o S, & So :These status signals, similar to IO/M.They can identify various operations, but they are rarelyused in small systems

Machine Cycle	Status			Control Signals
	$IO/\overline{M}$	S_1	S_0	
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	
Hold	Z	X	X	$\overline{RD}$, $\overline{WR}$ = Z and $\overline{INTA} = 1$
Reset	Z	X	X	

- **Interrupts :**

Processor has 5 interrupts. They are presented below in the order of their priority (from lowest to highest):

INTR is maskable interrupt. When the interrupt occurs the processor fetches instruction from the bus.

RST 5.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 2CH (hexadecimal) address.

RST 6.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 34H (hexadecimal) address.

- • Interrupts :Processor has 5 interrupts. They are presented below inthe order of their priority (from lowest to highest):INTR is maskable interrupt. When the interrupt occursthe processor fetches instruction from the bus.RST 5.5 is a maskable interrupt. When this interrupt is… received the processor saves the contents of the PC cregister into stack and branches to 2CH (hexadecimal)address.RST 6.5 is a maskable interrupt. When this interrupt isreceived the processor saves the contents of the PCregister into stack and branches to 34H (hexadecimal)address.

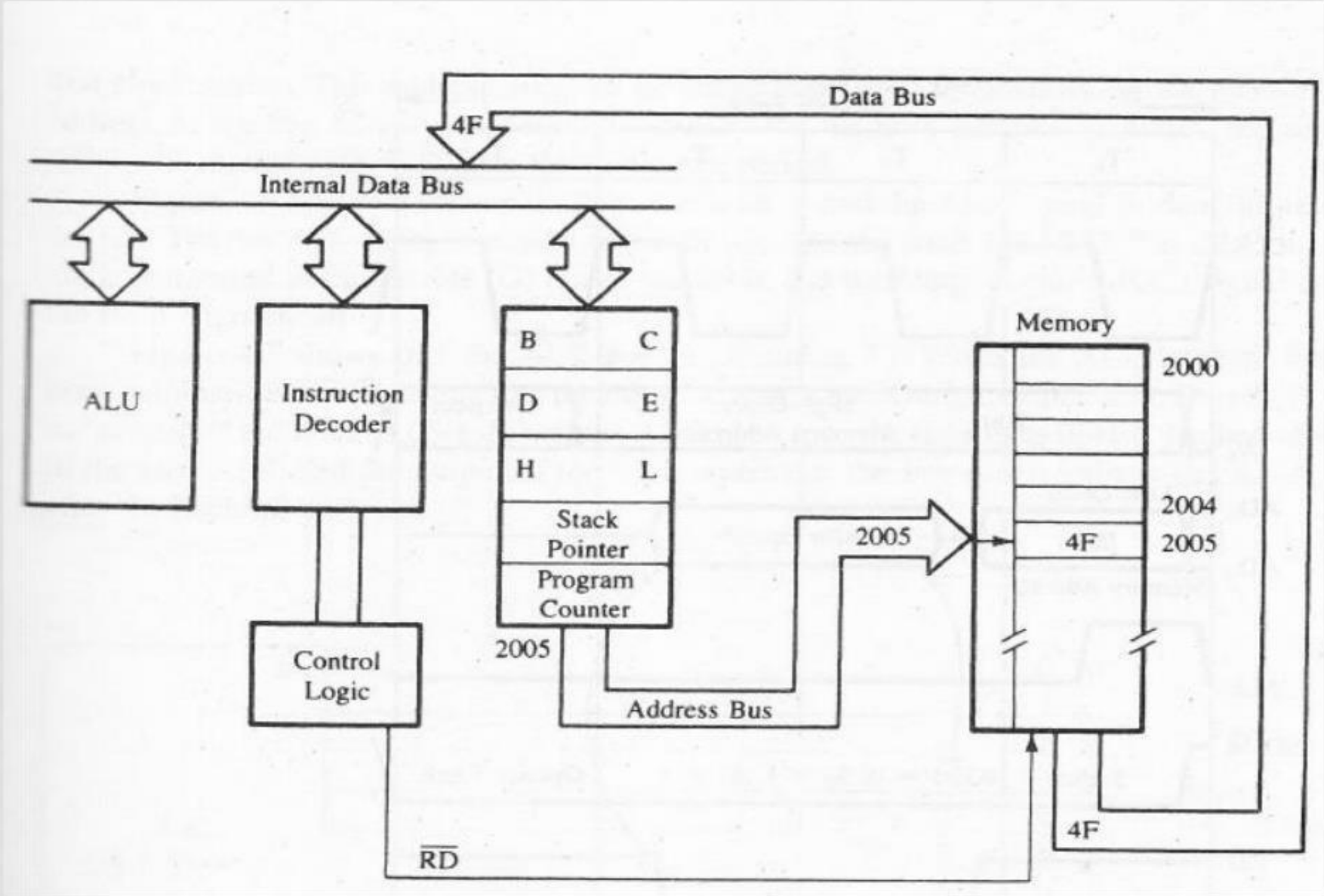
RST 7.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 3CH (hexadecimal) address.

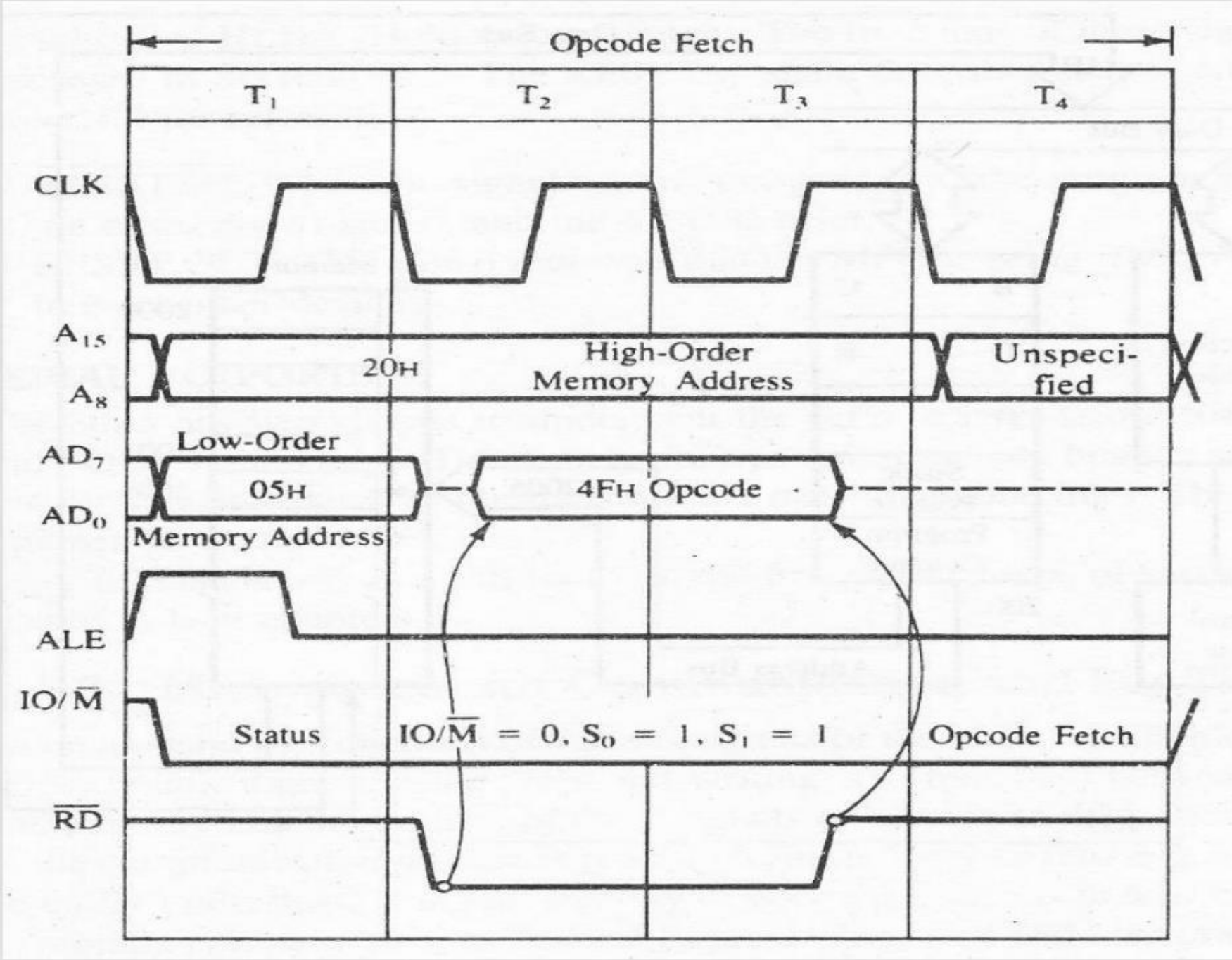
TRAP is a non-maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 24H (hexadecimal) address.

All maskable interrupts can be enabled or disabled using EI and DI instructions.

- RST 7.5 is a maskable interrupt. When this interrupt isreceived the processor saves the contents of the PCregister into stack and branches to 3CH (hexadecimal)address.TRAP is a non-maskable interrupt. When this interruptis received the processor saves the contents of the PCregister into stack and branches to 24H (hexadecimal)address.All maskable interrupts can be enabled or disabled usingEI and DI instructions.

- **Microprocessor Communication and Bus Timings**




Opcode Fetch
T1
T2
T3
T4
CLK
A15
A8
20H
High-Order
Memory Address
Unspeci-
fied
AD7
AD0
Low-Order
05H
4FH Opcode
Memory Address
ALE
IO/M
Status
IO/M = 0, S0 = 1, S1 = 1
Opcode Fetch
RD

- **TIMING SIGNALS FOR FETCHING AN INSTRUCTION**
  - At T1 , the high order 8 address bits (20H) are placed on the address lines A8 – A15 and the low order bits are placed on AD7–AD0.
  - The ALE signal goes high to indicate that AD0 – AD8 are carrying an address.
  - At exactly the same time, the IO/M signal goes low to indicate a memory operation.
  - At the beginning of the T2 cycle, the low order 8 address bits are removed from AD7– AD0 and the controller sends the Read (RD) signal to the memory.
  - The signal remains low (active) for two clock periods to allow for slow devices.
  - During T2 , memory places the data from the memory location on the lines AD7– AD0 .

- TIMING SIGNALS FOR FETCHING AN INSTRUCTIONAt T1 , the high order 8 address bits (20H) are placed on>the address lines A8 - A15 and the low order bits areplaced on AD7-ADO.The ALE signal goes high to indicate that AD0 - AD8 arecarrying an address.At exactly the same time, the IO/M signal goes low toindicate a memory operation.At the beginning of the T2 cycle, the low order 8 addressbits are removed from AD7- ADO and the controllersends the Read (RD) signal to the memory.The signal remains low (active) for two clock periods toallow for slow devices.During T2 , memory places the data from the memory>location on the lines AD7- AD0 .

- During T3 the RD signal is Disabled (goes high). This turns off the output Tri-state buffers in the memory. That makes the AD7– AD0 lines go to high impedence mode.

- The machine code or the byte (4FH) is decoded by the instruction decoder, and the contents of the accumulator are copied into register C. This task is performed during the period T4

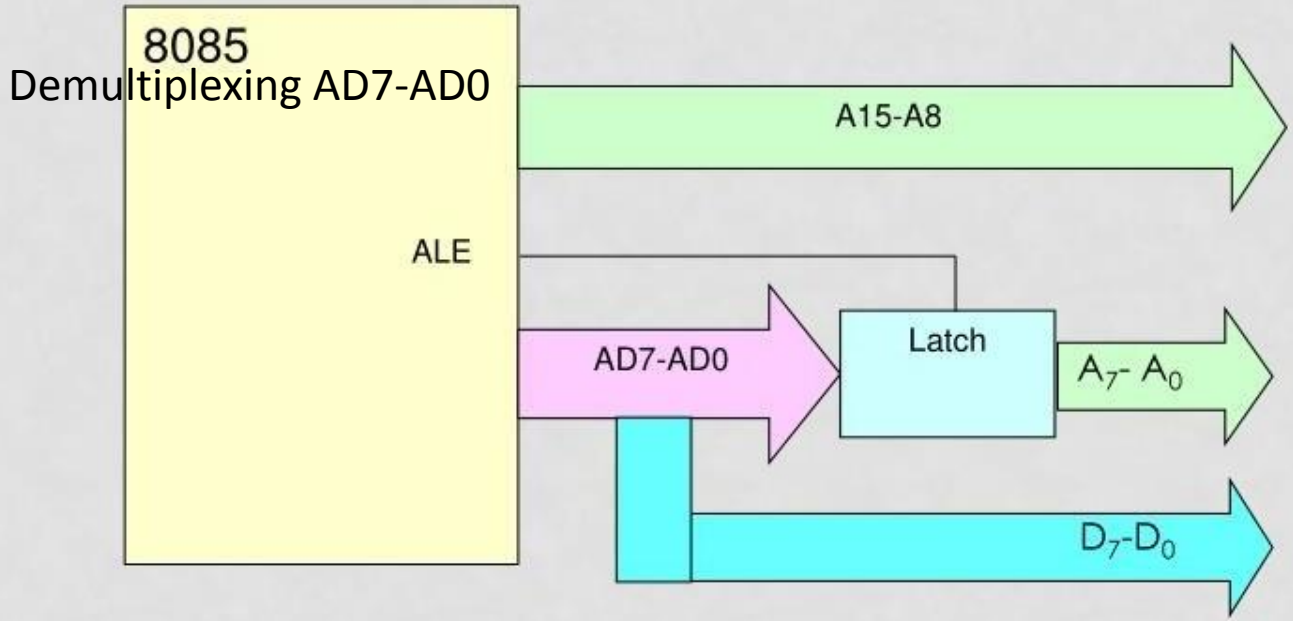
- During T3 the RD signal is Disabled (goes high). This turnsoff the output Tri-state buffers in the memory. That makesthe AD7- ADO lines go to high impedence mode.The machine code or the byte (4FH) is decoded by theinstruction decoder, and the contents of theaccumulator are copied into register C. This task isperformed during the period T4

• DEMULTIPLEXING AD7-AD0

- From the above description, it becomes obvious that the AD7– AD0 lines are serving a dual purpose and that they need to be demultiplexed to get all the information.
- The high order bits of the address remain on the bus for three clock periods. However, the low order bits remain for only one clock period and they would be lost if they are not saved externally.
- Also, notice that the low order bits of the address disappear when they are needed most.
- To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7– AD0 when it is carrying the address bits. We use the ALE signal to enable this latch.

- • DEMULTIPLEXING AD7-ADO• From the above description, it becomes obvious thatthe AD7- ADO lines are serving a dual purpose andthat they need to be demultiplexed to get all theinformation.• The high order bits of the address remain on the busfor three clock periods. However, the low order bitsremain for only one clock period and they would belost if they are not saved externally.• Also, notice that the low order bits of the addressdisappear when they are needed most.• To make sure we have the entire address for the fullthree clock cycles, we will use an external latch tosave the value of AD7- ADO when it is carrying theaddress bits. We use the ALE signal to enable thislatch.

• DEMULTIPLEXING AD7-AD0



- Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7– AD0 lines can be used for their purpose as the bi-directional data lines.

- Given that ALE operates as a pulse during T1, wewill be able to latch the address. Then when ALEgoes low, the address is saved and the AD7- ADOlines can be used for their purpose as the bi-directional data lines.

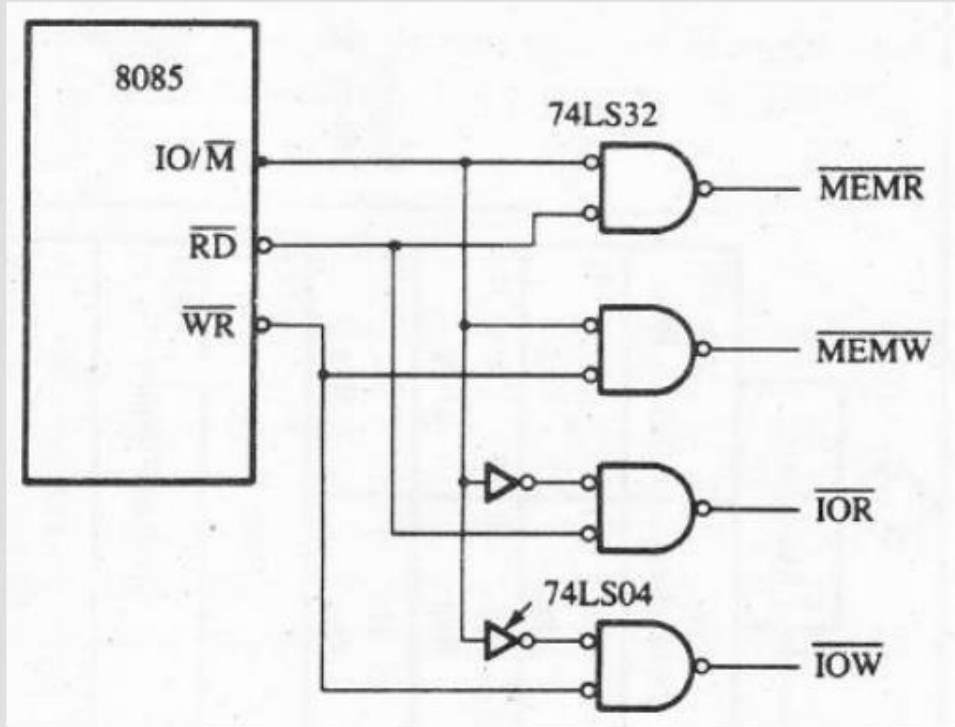
• CYCLES AND STATES

- From the above discussion, we can define terms that will become handy later on:
  - T- State: One subdivision of an operation. A T-state lasts for one clock period.
    - An instruction's execution length is usually measured in a number of T-states. (clock cycles).
  - Machine Cycle: The time required to complete one operation of accessing memory, I/O, or acknowledging an external request.
    - This cycle may consist of 3 to 6 T-states.
  - Instruction Cycle: The time required to complete the execution of an instruction.
    - In the 8085, an instruction cycle may consist of 1 to 6 machine cycles.

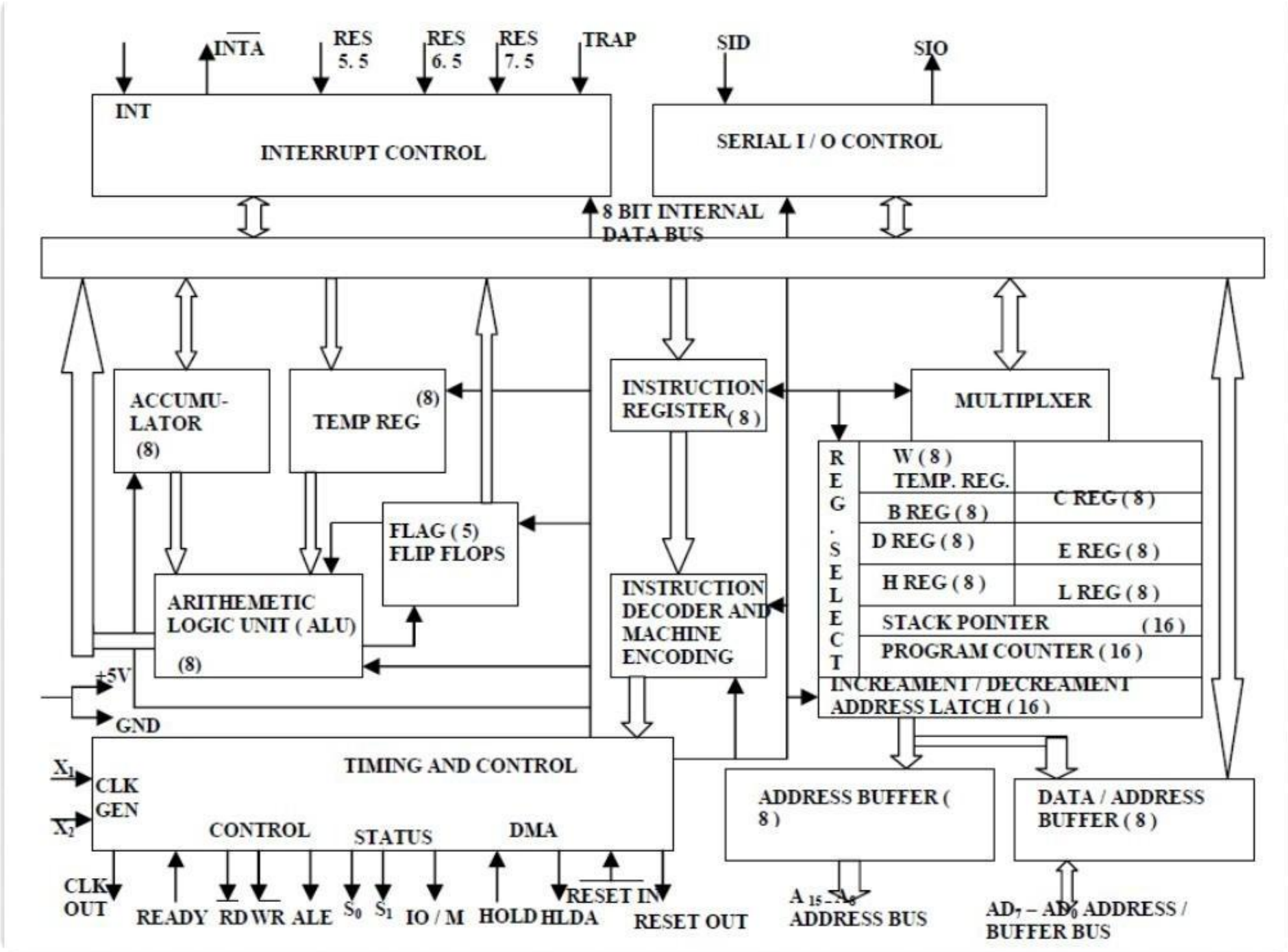
- • CYCLES AND STATES• From the above discussion, we can define terms thatwill become handy later on:• T- State: One subdivision of an operation. A T-statelasts for one clock period.• An instruction's execution length is usuallymeasured in a number of T-states. (clock cycles).• Machine Cycle: The time required to complete oneoperationofaccessingmemory, I/O,oracknowledging an external request.• This cycle may consist of 3 to 6 T-states.• Instruction Cycle: The time required to complete theexecution of an instruction.• In the 8085, an instruction cycle may consist of 1to 6 machine cycles.

• GENERATING CONTROL SIGNALS

- The 8085 generates a single RD signal. However, the signal needs to be used with both memory and I/O. So, it must be combined with the IO/M signal to generate different control signals for the memory and I/O.
  - Keeping in mind the operation of the IO/M signal we can use the following circuitry to generate the right set of signals:



- •GENERATING CONTROL SIGNALS• The 8085 generates a single RD signal. However, thesignal needs to be used with both memory and I/O. So,it must be combined with the IO/M signal to generatedifferent control signals for the memory and I/O.• Keeping in mind the operation of the IO/M signal wecan use the following circuitry to generate the right setof signals:


INTA
RES 5. 5
RES 6. 5
RES 7. 5
TRAP
SID
SIO
INT
INTERRUPT CONTROL
SERIAL I / O CONTROL
8 BIT INTERNAL DATA BUS
ACCUMU-LATOR (8)
TEMP REG (8)
INSTRUCTION REGISTER (8)
MULTIPLXER
FLAG (5) FLIP FLOPS
ARITHEMETIC LOGIC UNIT (ALU) (8)
INSTRUCTION DECODER AND MACHINE ENCODING
REG. SELECT
W (8) TEMP. REG.
C REG (8)
B REG (8)
D REG (8)
E REG (8)
H REG (8)
L REG (8)
STACK POINTER (16)
PROGRAM COUNTER (16)
INCREAMENT / DECREAMENT ADDRESS LATCH (16)
+5V
GND
TIMING AND CONTROL
CLK GEN
CONTROL
STATUS
DMA
ADDRESS BUFFER (8)
DATA / ADDRESS BUFFER (8)
CLK OUT
READY
RD
WR
ALE
S0
S1
IO / M
HOLD
HLDA
RESET IN
RESET OUT
A15 – A8 ADDRESS BUS
AD7 – AD0 ADDRESS / BUFFER BUS

- **THE 8085 MACHINE CYCLES**

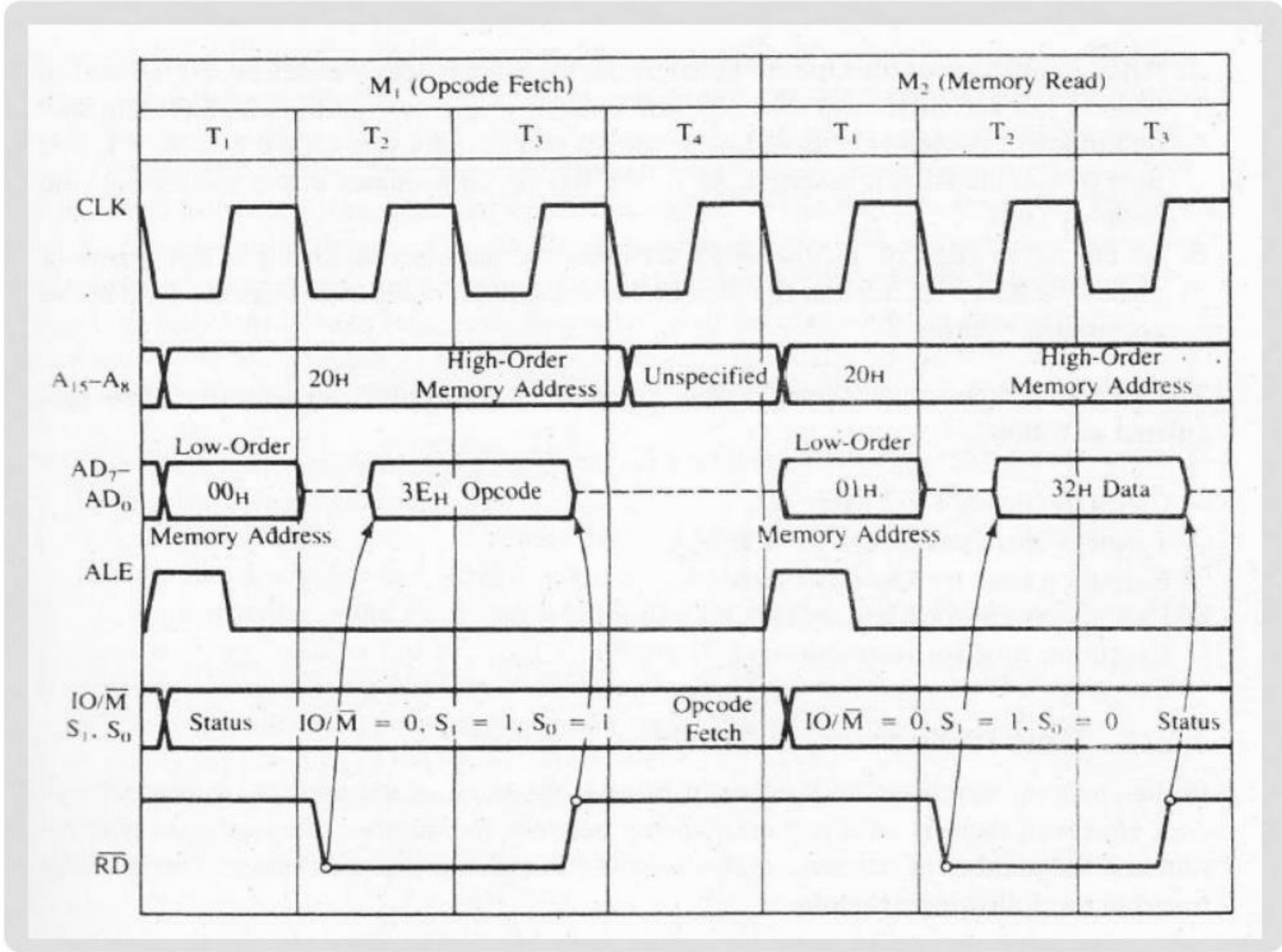
  - The 8085 executes several types of instructions with each requiring a different number of operations of different types. However, the operations can be grouped into a small set.
  - The three main types are:
    - Memory Read and Write.
    - I/O Read and Write.
    - Request Acknowledge.
  - These can be further divided into various operations (machine cycles).

- • THE 8085 MACHINE CYCLES• The 8085 executes several types of instructions witheach requiring a different number of operations ofdifferent types. However, the operations can begrouped into a small set.• The three main types are:• Memory Read and Write.• I/O Read and Write.• Request Acknowledge.• These can be further divided into various operations(machine cycles).

• OPCODE FETCH MACHINE CYCLE

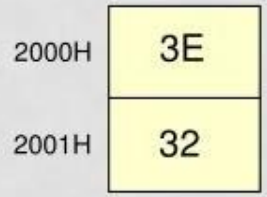
- The first step of executing any instruction is the Opcode fetch cycle.
  - In this cycle, the microprocessor brings in the instruction's Opcode from memory.
    - To differentiate this machine cycle from the very similar "memory read" cycle, the control & status signals are set as follows:
      - IO/M=0, s0 and s1 are both 1.
  - This machine cycle has four T-states.
    - The 8085 uses the first 3 T-states to fetch the opcode.
    - T4 is used to decode and execute it.
  - It is also possible for an instruction to have 6 T-states in an opcode fetch machine cycle.

- • OPCODE FETCH MACHINE CYCLE• The first step of executing any instruction is theOpcode fetch cycle.• In this cycle, the microprocessor brings in theinstruction's Opcode from memory.• To differentiate this machine cycle from the verysimilar "memory read" cycle, the control & statussignals are set as follows:• IO/M=0, s0 and s1 are both 1.• This machine cycle has four T-states.• The 8085 uses the first 3 T-states to fetch theopcode.• T4 is used to decode and execute it.• It is also possible for an instruction to have 6 T-states in an opcode fetch machine cycle.


M1 (Opcode Fetch)
M2 (Memory Read)
T1
T2
T3
T4
T1
T2
T3
CLK
A15–A8
20H
High-Order Memory Address
Unspecified
20H
High-Order Memory Address
Low-Order
AD7–AD0
00H
3EH Opcode
Low-Order
01H
32H Data
Memory Address
Memory Address
ALE
IO/M
S1, S0
Status
IO/M = 0, S1 = 1, S0 = 1
Opcode Fetch
IO/M = 0, S1 = 1, S0 = 0
Status
RD

THE MEMORY READ MACHINE CYCLE

- To understand the memory read machine cycle, let's study the execution of the following instruction:
  - MVI A, 32



- In memory, this instruction looks like:
  - The first byte 3EH represents the opcode for loading a byte into the accumulator (MVI A), the second byte is the data to be loaded.
- The 8085 needs to read these two bytes from memory before it can execute the instruction. Therefore, it will need at least two machine cycles.
  - The first machine cycle is the opcode fetch discussed earlier.
  - The second machine cycle is the Memory Read Cycle.

- the memory read machine cycle• To understand the memory read machine cycle, let'sstudy the execution of the following instruction:• MVI A, 322000H3E• In memory, this instruction looks like:2001H32• The first byte 3EH represents the opcode forloading a byte into the accumulator (MVI A), thesecond byte is the data to be loaded.• The 8085 needs to read these two bytes from memorybefore it can execute the instruction. Therefore, it willneed at least two machine cycles.• The first machine cycle is the opcode fetchdiscussed earlier.• The second machine cycle is the Memory ReadCycle.

• MACHINE CYCLES VS. NUMBER OF BYTES IN THE INSTRUCTION

- Machine cycles and instruction length, do not have a direct relationship.
  - To illustrate lets look at the machine cycles needed to execute the following instruction.
    - STA 2065H
    - This is a 3-byte instruction requiring 4 machine cycles and 13 T-states.
    - The machine code will be stored in memory as shown to the right

32H	2010H
65H	2011H
20H	2012H

    - This instruction requires the following 4 machine cycles:
      - Opcode fetch to fetch the opcode (32H) from location 2010H, decode it and determine that 2 more bytes are needed (4 T-states).
      - Memory read to read the low order byte of the address (65H) (3 T-states).
      - Memory read to read the high order byte of the address (20H) (3 T-states).
      - A memory write to write the contents of the accumulator into the memory location.

- • MACHINE CYCLES VS. NUMBER OF BYTES INTHE INSTRUCTION• Machine cycles and instruction length, do not have a directrelationship.• To illustrate lets look at the machine cycles needed to executethe following instruction.• STA 2065H• This is a 3-byte instruction requiring 4 machine cycles and 13 T-states.32H 2010-• The machine code will be stored65H 2011Hin memory as shown to the right20H2012H• This instruction requires the following 4 machine cycles:• Opcode fetch to fetch the opcode (32H) from location 2010H,decode it and determine that 2 more bytes are needed (4 T-states).• Memory read to read the low order byte of the address (65H) (3T-states).•• Memory read to read the high order byte of the address (20H) (3T-states).• A• A memory write to write the contents of the accumulator intothe memory location.

LECTURE 13
MICROPROCESSORS

Microprocessors

Input / Output

- **Input Devices**
  - Switches , Keyboard , ….

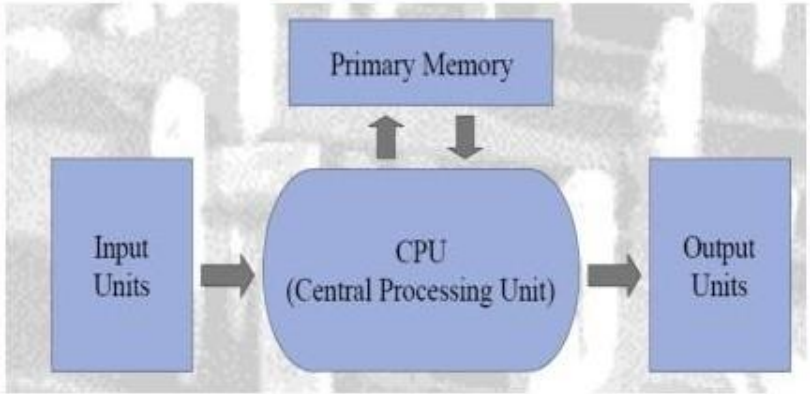
- **Output Devices:**
  - Seven Segments (LEDs) , printer , Monitor ,..

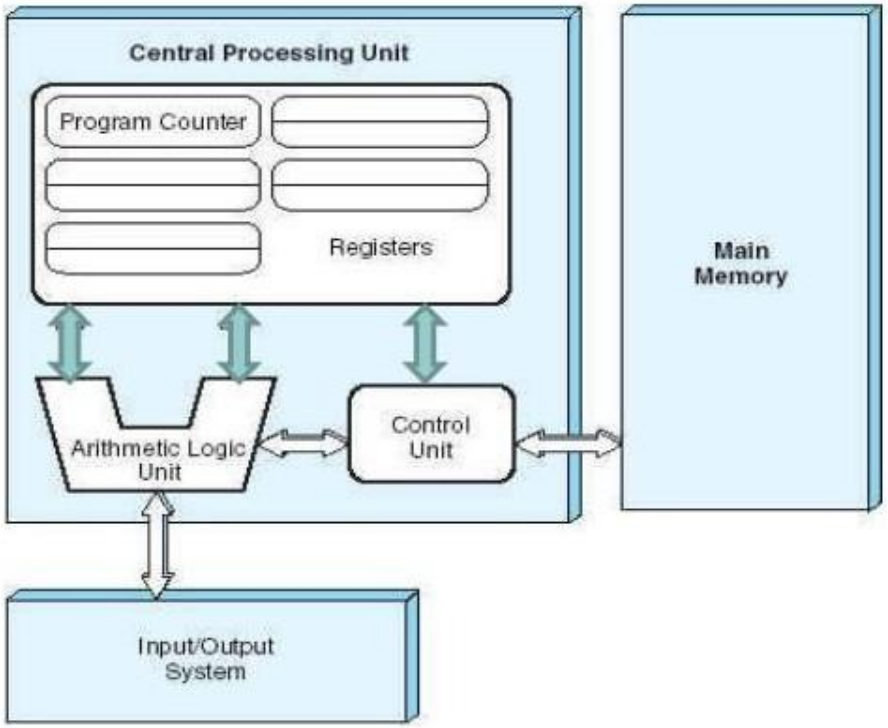
- The processor reads the instructions from the memory , data from the input devices, processes them, produces the output

- Input / OutputInput DevicesSwitches , Keyboard , …Output Devices:Seven Segments (LEDs) , printer , Monitor ,…The processor reads the instructions from thememory , data from the input devices,processes them, produces the output

The CPU includes ALU, control Units, and Various Registers

Microprocessor as CPU





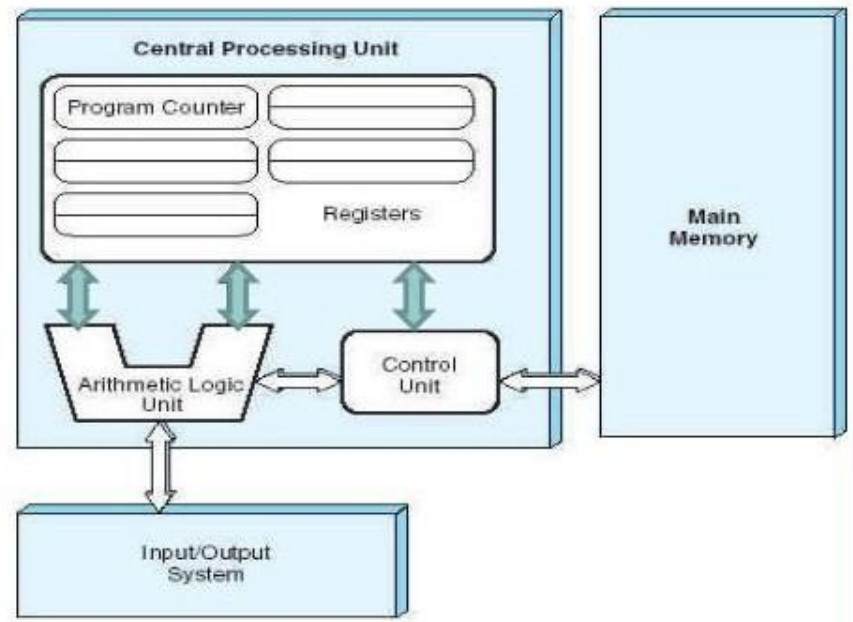
-The CPU includes ALU, control Units , and Various Registers

-Known as Microprocessor

The Von Neumann Model

- It uses *von Neumann execution cycle*

  (also called the *fetch-decode-execute cycle*)

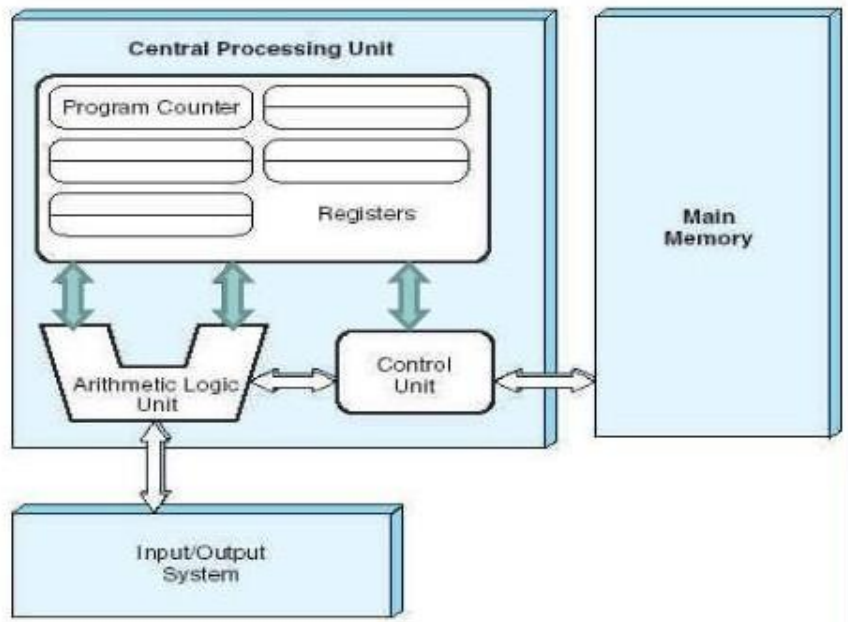


The Von Neumann Model
It uses von Neumann execution cycle
(also called the fetch-decode-execute cycle)

The Von Neumann Model (Cont.)

- **A cycle could be as follows:**

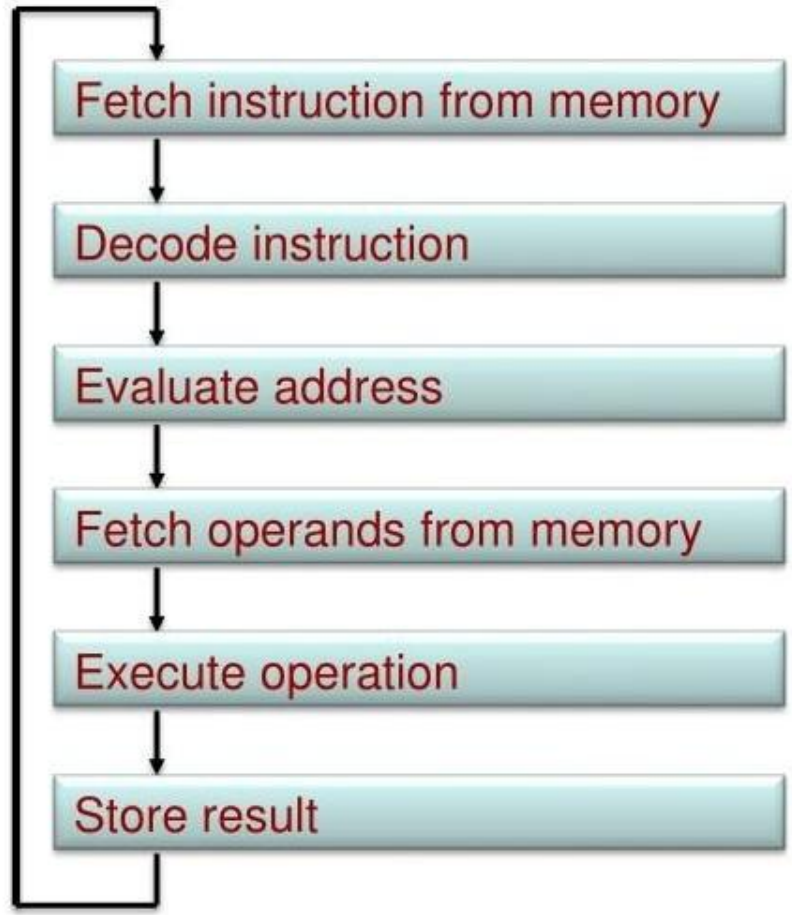
1. The control unit fetches the next program instruction from the memory, using the program counter to determine where the instruction is located.
2. The instruction is decoded into a language the ALU can understand.
3. Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.
4. The ALU executes the instruction and places the results in registers or memory.



- A cycle could be as follows:The control unit fetches the next programinstruction from the memory, using the programcounter to determine where the instruction islocated.The instruction is decoded into a language theALU can understand.Any data operands required to execute theinstruction are fetched from memory and placedinto registers within the CPU.The ALU executes the instruction and places theresults in registers or memory.

Instruction Processing

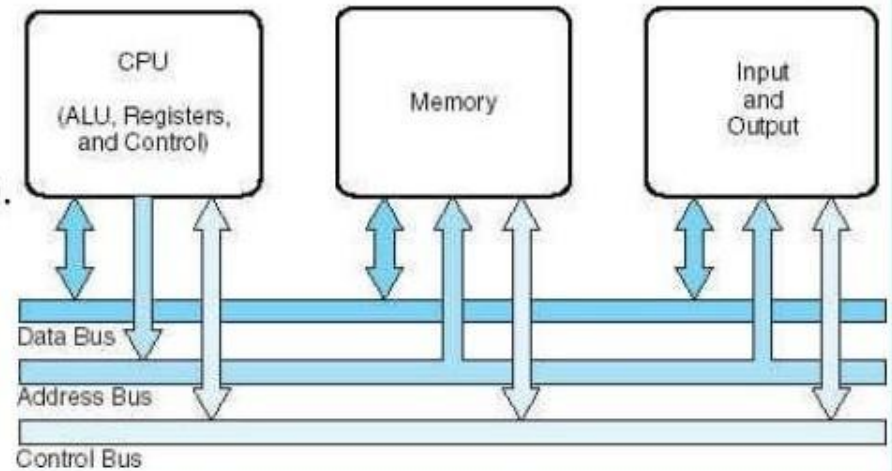
Von Neumann execution cycle



The Modified Von Neumann Model

- **The data bus:**

  Moves data from main memory to the CPU registers (and vice versa).



- **The address bus:**

  Holds the address of the data that the data bus is currently accessing.

- **The control bus:**

  Carries the necessary control signals that specify how the information transfer is to take place.

Advances in Semiconductor Technology

- **IC-** Integrated Circuits → few transistors and diodes on one chip
- **SSI** –small scale Integration→ few gates on one chip
- **MSI-** Medium scale Integration- 100 gates on a chip
- **LSI** – Large Scale Integration – 1000 gates on a chip
- **VLSI** – Very large scale Integration
- **SLSI** – Super Large Scale Integration

- Borders between **VLSI** and **SLSI** are not strict.

- Advances in SemiconductorTechnologyIC- Integrated Circuits -> few transistors and diodes ononе chipSSI -small scale Integration-> few gates on one chipMSI- Medium scale Integration- 100 gates on a chipLSI - Large Scale Integration - 1000 gates on a chipVLSI - Very large scale IntegrationSLSI - Super Large Scale IntegrationBorders between VLSI and SLSI are not strict.

Microprocessor Programming

- Machine language
  - Instruction written in binary format
- Assembly language
  - Text based format → Add A , B
- High level Language

Microprocessor programming
* Machine language
Instruction written in binary format
- Assembly language

Text based format add a, b
- *High level language

Z80 instructions and alphanumeric codes

Z80 Instructions and Alphanumeric Codes

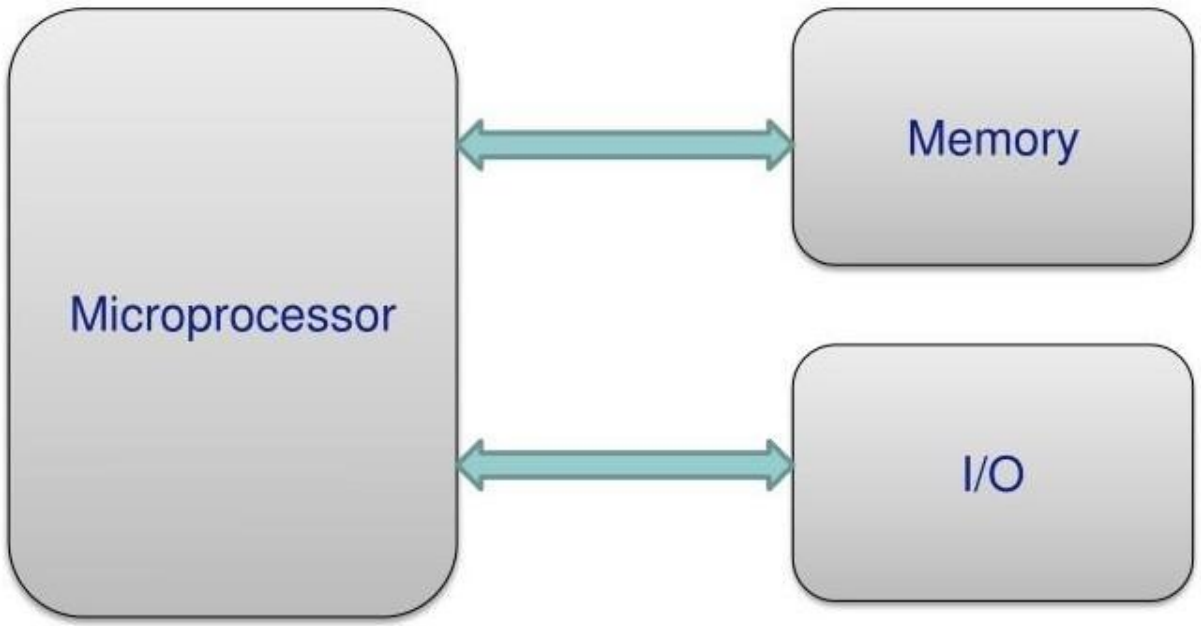
- **8-bit** word length
- 158 instructions

- **ASCII** – American Standard Code for Information Interchange.
  - Each character has its equivalent binary format in a **7-bit code**

- **EBCDIC** – Extended Binary Coded Decimal Interchange Code – **8-bit code**

Microprocessor Based System

Microprocessor-Based System



Microprocessor Unit
Progremmable logic unit with a designed set of instructions

Microprocessor Unit

- **Programmable logic unit with a designed set of instructions**
- **What it does:**
  - Fetches the instructions from the memory, one by one
  - Reads the input data from the input units
  - Performs the data manipulation specified by the instruction
  - Writes the data to the output devices

MPU frequently communicates with the memory, I/O devices

Microprocessor Unit

- **MPU frequently communicates with the memory, I/O devices**
  - Fetch, Decode, and Execute operations

- **Can it be interrupted ?**
  - Program initiated operation – interrupt done by a program.
  - Peripheral initiated operation – interrupt done by external devices
    - E.g. important data on the input during writing to the printer

What does it needs to do so..

What does it needs to do so..

- Group of logic circuits
- Set of signal to transfer information
- Control signals for timing
- Clock circuits

What does it needs to do so…Group of logic circuitsSet of signal to transfer informationControl signals for timingClock circuits

Program-initiated operations and Buses

- Microprocessor and Memory Operations
  - Memory Read
    - Reads instructions or data from the memory
  - Memory Write
    - Writes instructions and data into memory
  - I/O Read
    - Accepts data from input devices
  - I/O Write
    - Writes data to output devices

Program-initiated operationsand BusesanMicroprocessor and Memory OperationsMemory ReadReads instructions or data from the memoryMemory WriteWrites instructions and data into memoryI/O ReadAccepts data from input devicesI/O WriteWrites data to output devices

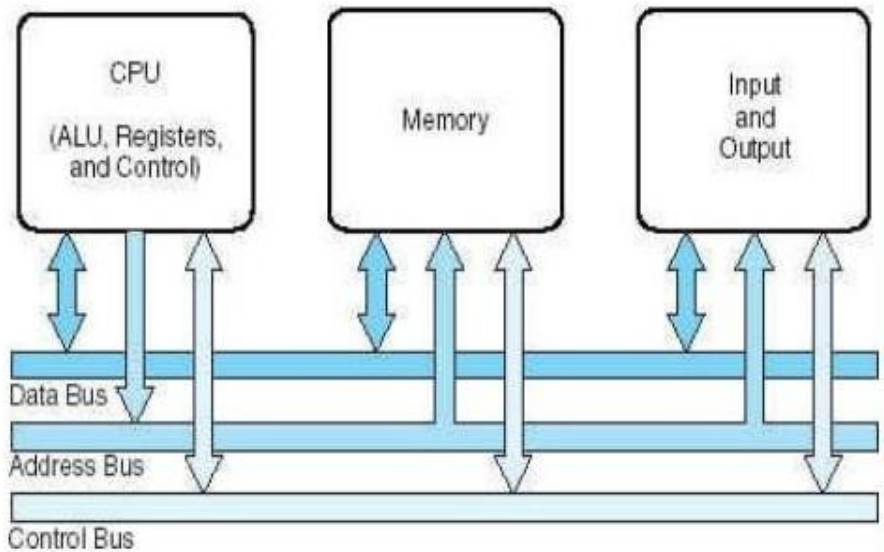
Program-initiated operations and Buses

- **From where to read or to write?**
  - We need an address! **Right**?
- **How the input/output will know about the operation?**
  - We need a control signal to tell them
- **MPU Operations Steps:**
  - Identify the address
  - Send synchronization SIGNAL– control signal
  - Transfer the binary data
- So, how many buses do we need?

Program-initiated operationsand BusesFrom where to read or to write?We need an address! Right?How the input/output will know about the operation?We need a control signal to tell themMPU Operations Steps:Identify the addressSend synchronization SIGNAL- control signal•Transfer the binary dataSo, how many buses do we need?

Buses

- **Address Bus**
  - Identify the memory locations
- **Data Bus**
  - Holds the data during transfer operation
- **Control Lines**
  - For timing signal



BusesAddress BusIdentify the memorylocationsCPU(ALU, Registers,and Control)MemoryInputandOutputData BusData BusHolds the data duringAddress Bustransfer operationControl BusControl Lines° For timing signal

Buses

- **Address Bus Size - bits**
  - Depends on the number of memory locations that can be accessed
  - Z80 has 16 address lines to address 2^{16} locations
- **Data Bus Size - bits**
  - Depends on the data to be transferred
  - Z80 has 8 bits data bus
- **What is the maximum memory size Z80 can use?**

BusesAddress Bus Size - bitsDepends on the number of memory locations thatcan be accessedZ80 has 16 address lines to address 216 locationsData Bus Size - bitsDepends on the data to be transferredZ80 has 8 bits data busWhat is the maximum memory size Z80 can use?

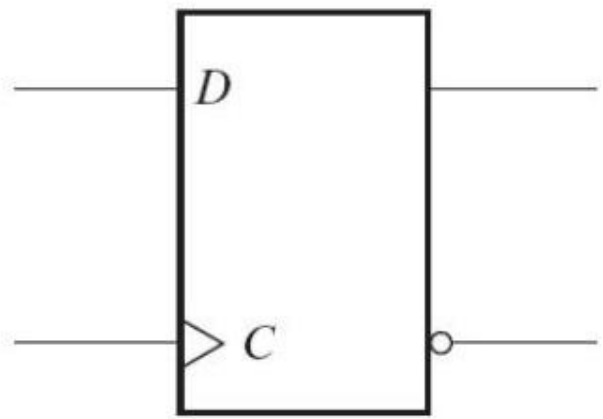
Externally Initiated operation

- **Interruptions categories :**
  - **Reset** – e.g. timer to reset everything in the MPU
  - **Interrupt** – stop temporarily and do something , then come back.
  - **Wait**: the memory can not handle the MPU request , wait signal must be generated.
  - **Bus Request:** sometimes the processor is too slow to hand a request that can be handled faster by another device.
    - E.g transfer large amount of data through the DMA could be faster than using the MPU

Memory

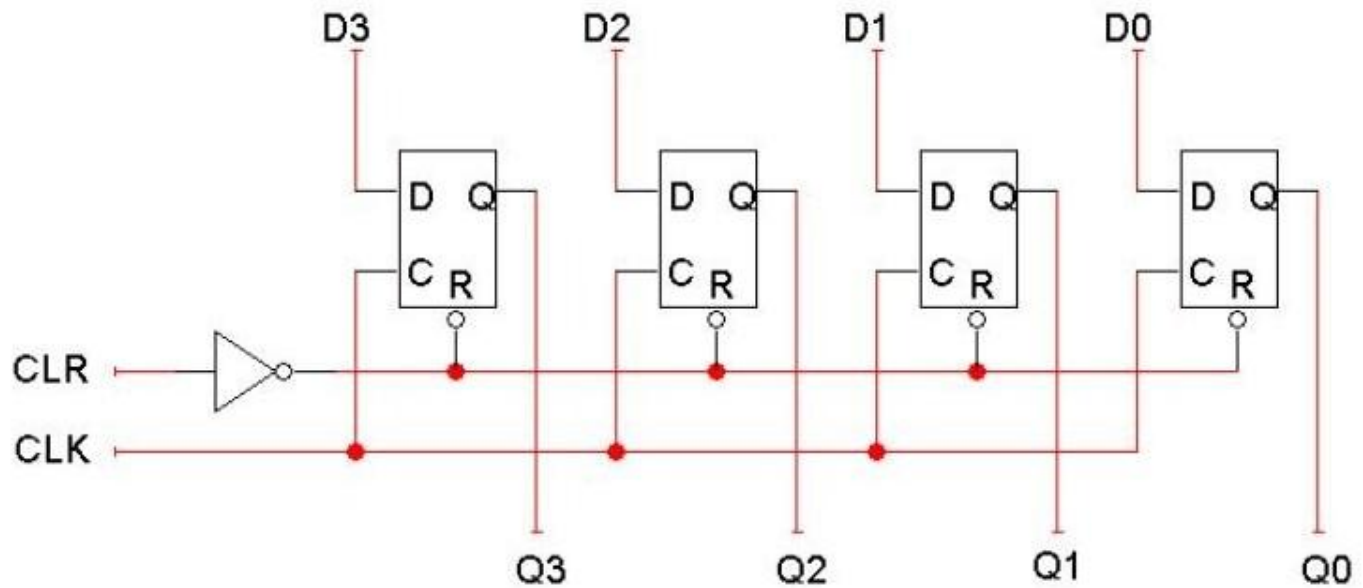
- Memory Cell

D	Q(t+1)	
0	0	Reset
1	1	Set

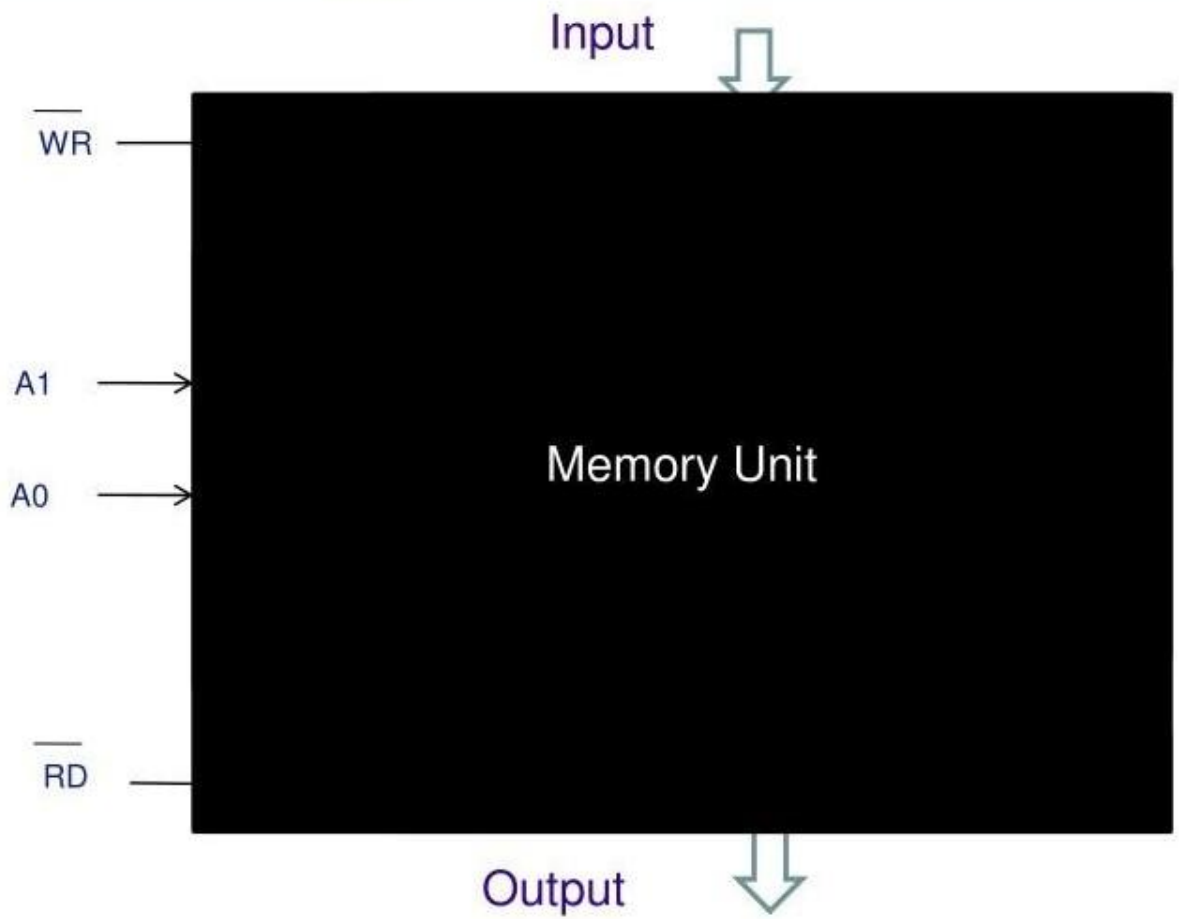


Memory Continue

- 4-bit Register



4 X 8 bit register



How the MPU Writes into the Memory?

- MPU places the 16 bit address on the address bus
  - Memory interfacing circuits will decode address to specify the target register
- MPU Places a byte on the data bus
- MPU sends a control signal (Memory Write) to the memory to write

How the MPU Writes into the Memory?•MPU places the 16 bit address on the address busMemory interfacing circuits will decode address tospecify the target registerMPU Places a byte on the data bus• MPU sends a control signal (Memory Write) tothe memory to write

How the MPU reads from the Memory?

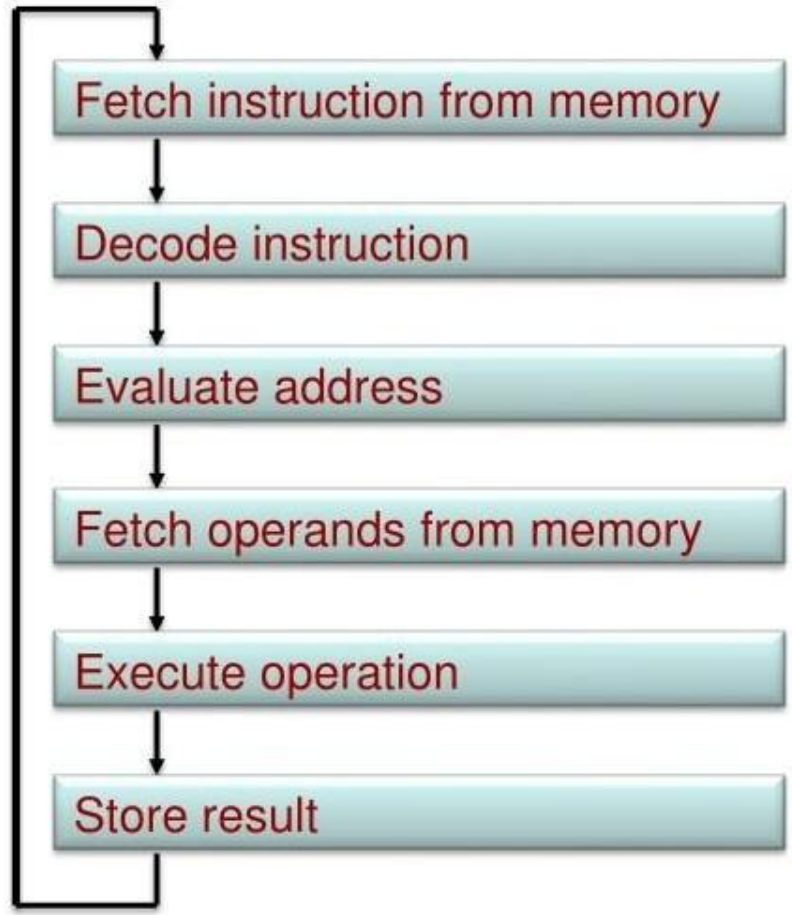
- MPU places the 16 bit address on the address bus
  - Memory interfacing circuits will decode address to specify the target register
- MPU sends a control signal (Memory Read) to the memory to enable the output buffer
- The memory puts the data on the data bus and the processor will read it

How the MPU reads from the Memory?MPU places the 16 bit address on the address busMemory interfacing circuits will decode address tospecify the target registerMPU sends a control signal (Memory Read) tothe memory to enable the output buffer•The memory puts the data on the data bus and theprocessor will read it

LECTURE 13
MICROPROCESSOR

Instruction Processing

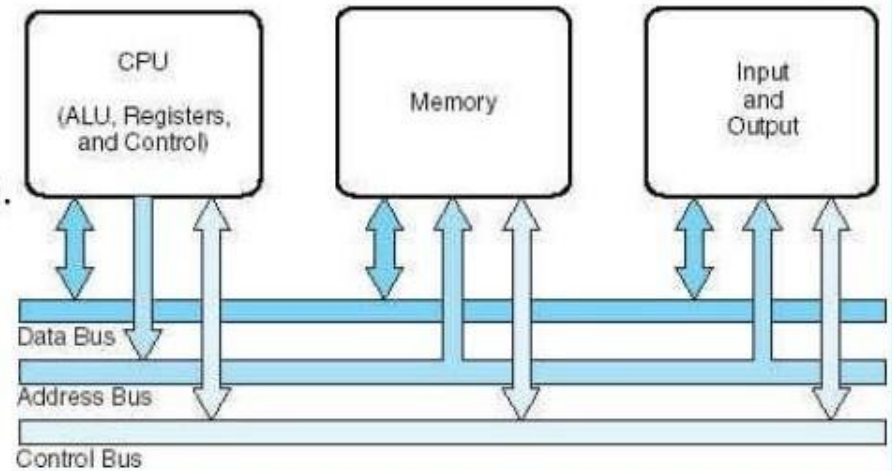
Von Neumann execution cycle



The Modified Von Neumann Model

- **The data bus:**

  Moves data from main memory to the CPU registers (and vice versa).



- **The address bus:**

  Holds the address of the data that the data bus is currently accessing.

- **The control bus:**

  Carries the necessary control signals that specify how the information transfer is to take place.

Buses

- **Address Bus Size - bits**
  - Depends on the number of memory locations that can be accessed
  - Z80 has 16 address lines to address 2^{16} locations
- **Data Bus Size - bits**
  - Depends on the data to be transferred
  - Z80 has 8 bits data bus
- **What is the maximum memory size Z80 can use?**

BusesAddress Bus Size - bitsDepends on the number of memory locations thatcan be accessedZ80 has 16 address lines to address 216 locationsData Bus Size - bitsDepends on the data to be transferredZ80 has 8 bits data bus• What is the maximum memory size Z80 can use?

Externally Initiated operation

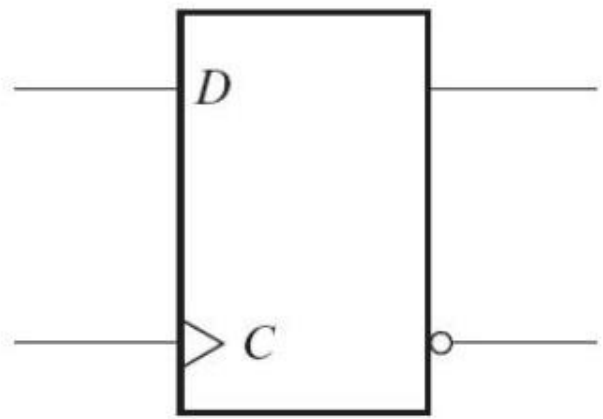
- **Interruptions categories :**
  - **Reset** – e.g. timer to reset everything in the MPU
  - **Interrupt** – stop temporarily and do something , then come back.
  - **Wait**: the memory can not handle the MPU request , wait signal must be generated.
  - **Bus Request:** sometimes the processor is too slow to hand a request that can be handled faster by another device.
    - E.g transfer large amount of data through the DMA could be faster than using the MPU

Externally Initiated operationInterruptions categories :Reset - e. g. timer to reset everything in the MPUInterrupt - stop temporarily and do something , then come back.Wait: the memory can not handle the MPU request , wait signalmust be generated.Bus Request: sometimes the processor is too slow to hand arequest that can be handled faster by another device.E.g transfer large amount of data through the DMA could befaster than using the MPU

Memory

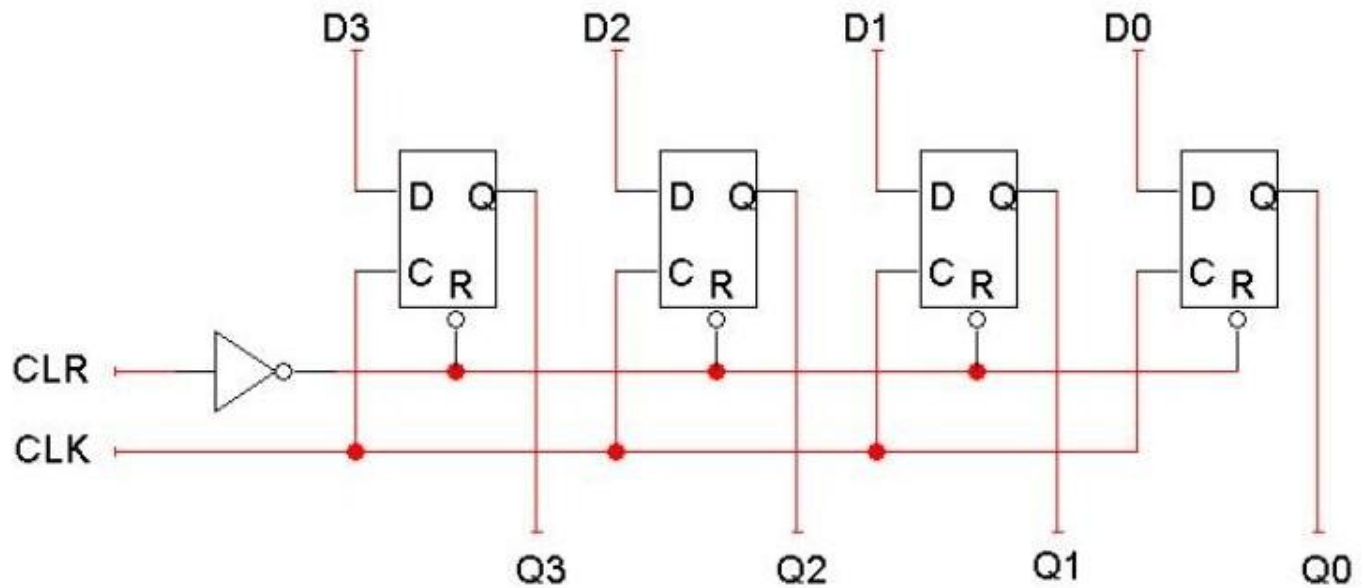
- Memory Cell

D	Q(t+1)	
0	0	Reset
1	1	Set

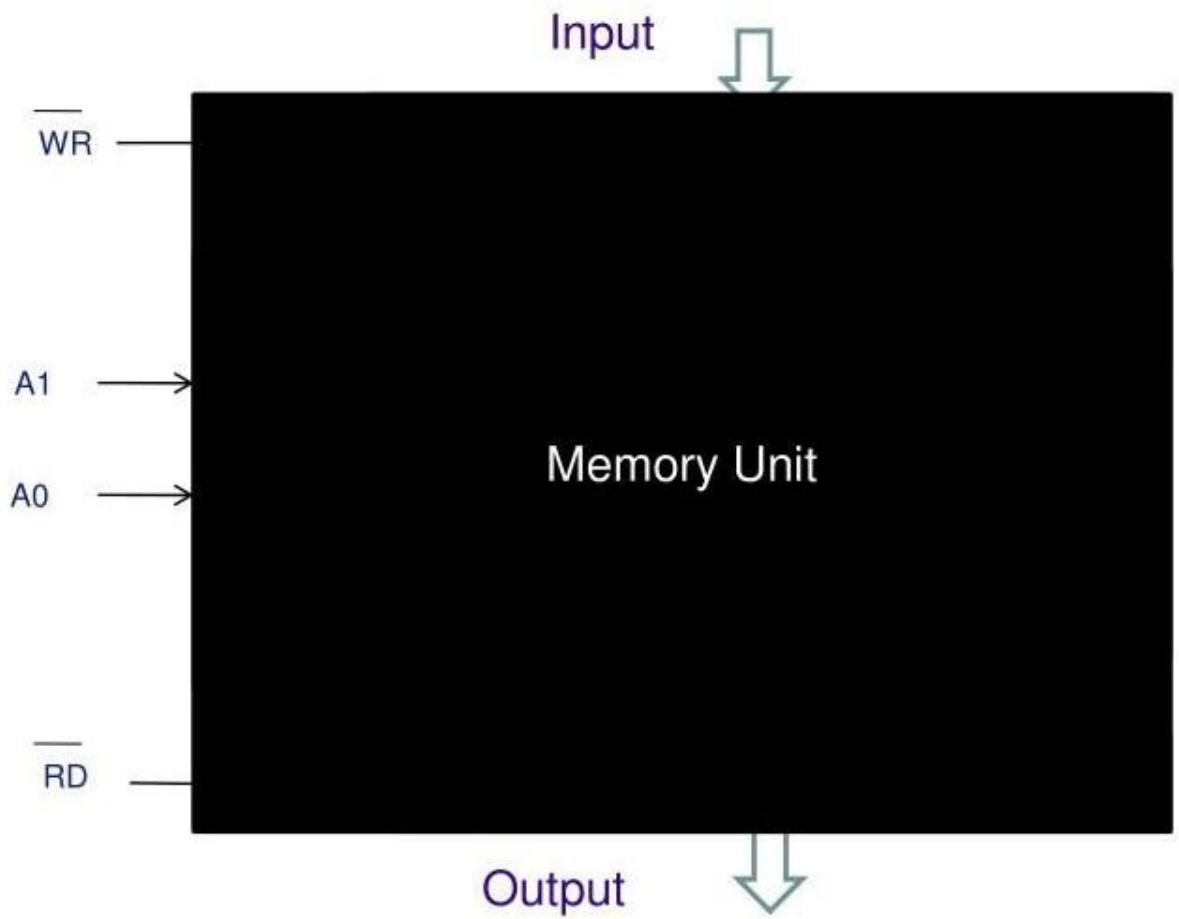


Memory Continue

- 4-bit Register



4 X 8 bit register



How the MPU Writes into the Memory?

- MPU places the 16 bit address on the address bus
  - Memory interfacing circuits will decode address to specify the target register
- MPU Places a byte on the data bus
- MPU sends a control signal (Memory Write) to the memory to write

How the MPU reads from the Memory?

- MPU places the 16 bit address on the address bus
  - Memory interfacing circuits will decode address to specify the target register
- MPU sends a control signal (Memory Read) to the memory to enable the output buffer
- The memory puts the data on the data bus and the processor will read it

LECTURE 14

Microcontrollers

Fundamentals of Logic Design

By Dana Utebayeva

Micro-Controller

- A single chip Computer (to some extent)
- Has CPU
  1. RAM
  2. EEPROM
  3. I/O in form of pins
  4. Peripherals (Timer , Communication modes , ADC etc)

Background

- Line Following Robots
- Wireless keyboards
- They were made using Microcontrollers

- Suppose we want to make a Line following Robot
- What do we do ?
- Use a computer with 2.4Ghz Intel core I7 with 4 Gb RAM , 500 Gb Hard disk , 1 Gb Graphics Card ??

Why not a Computer ?

- PC is a general purpose computer.
- Can run thousand of softwares
- Microsoft ppt in which you are seeing this presentation
- Games (NFS , AOE , Call of Duty)
- Highly expensive

Why MCU

- Small reflected by the word “MICRO”
- Inexpensive
- Ideal for doing repetitive tasks
- Easy to use
- Highly Efficient and fast

Selecting a MCU

- Two family of MCU extremely popular
  a) AVR
  b) PIC

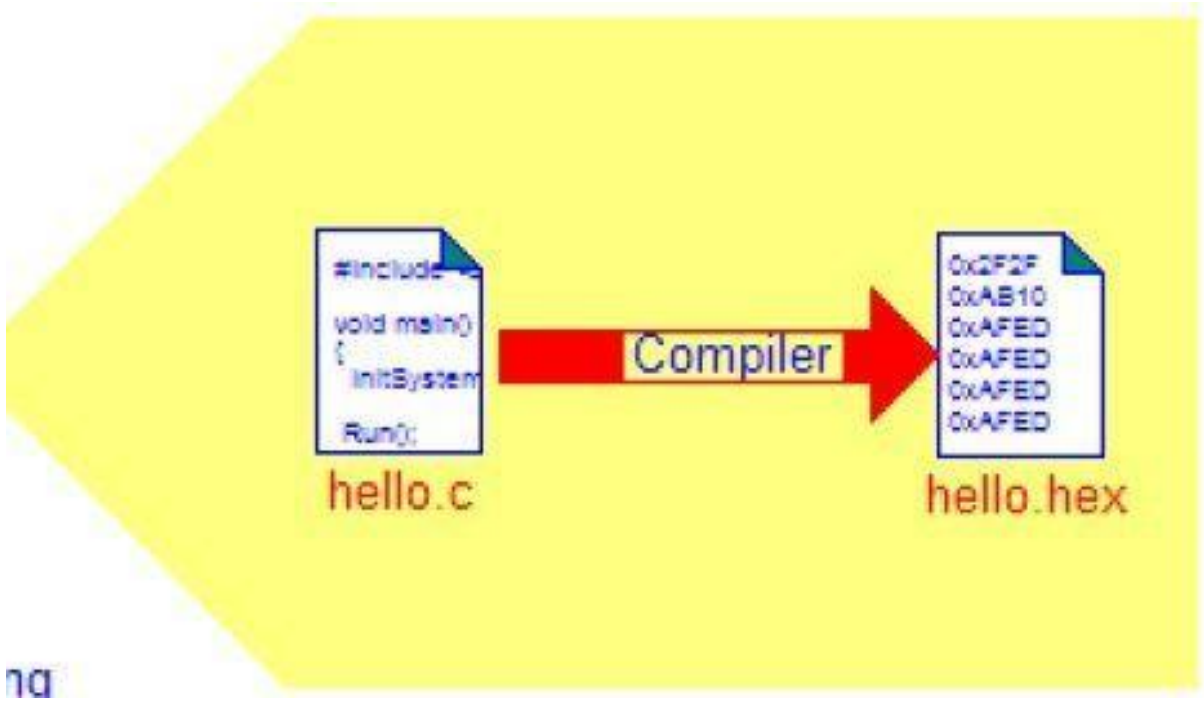
- We use AVR series of MCU from Atmel
- The instructions are fed once in the form of a Hex file

Tools Required -> CVAVR

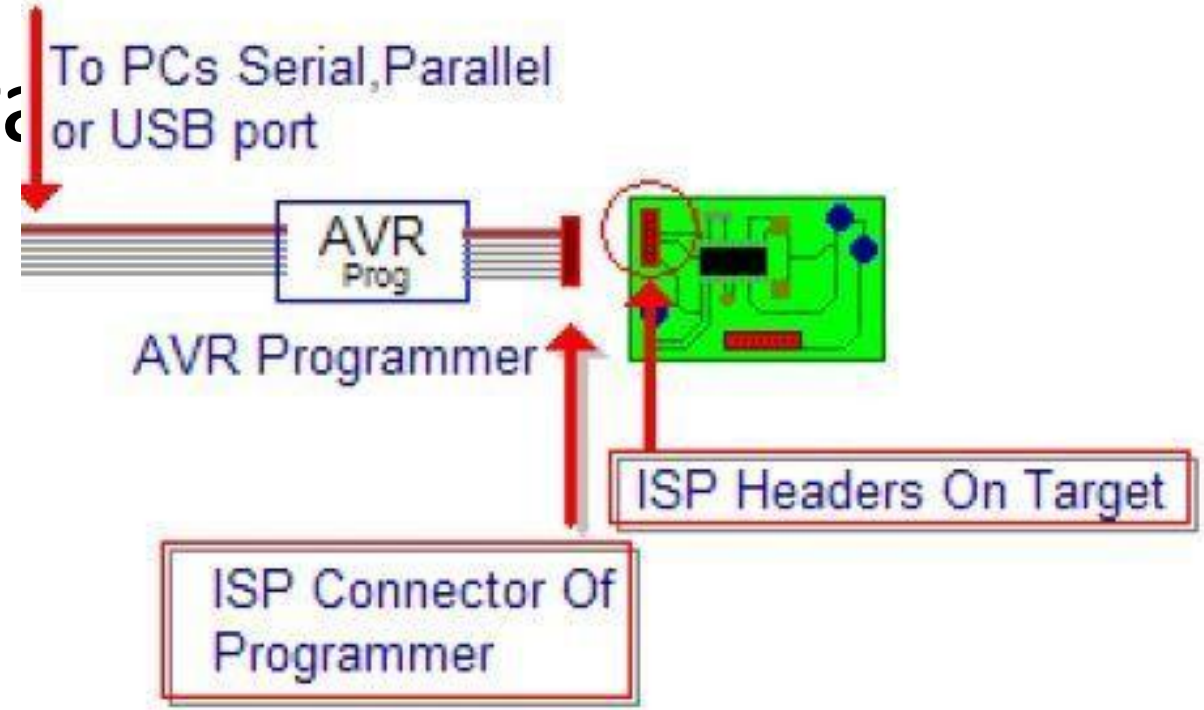
PC Running IDE for entering,editing and compiling source program.

Compiler -> CVAVR

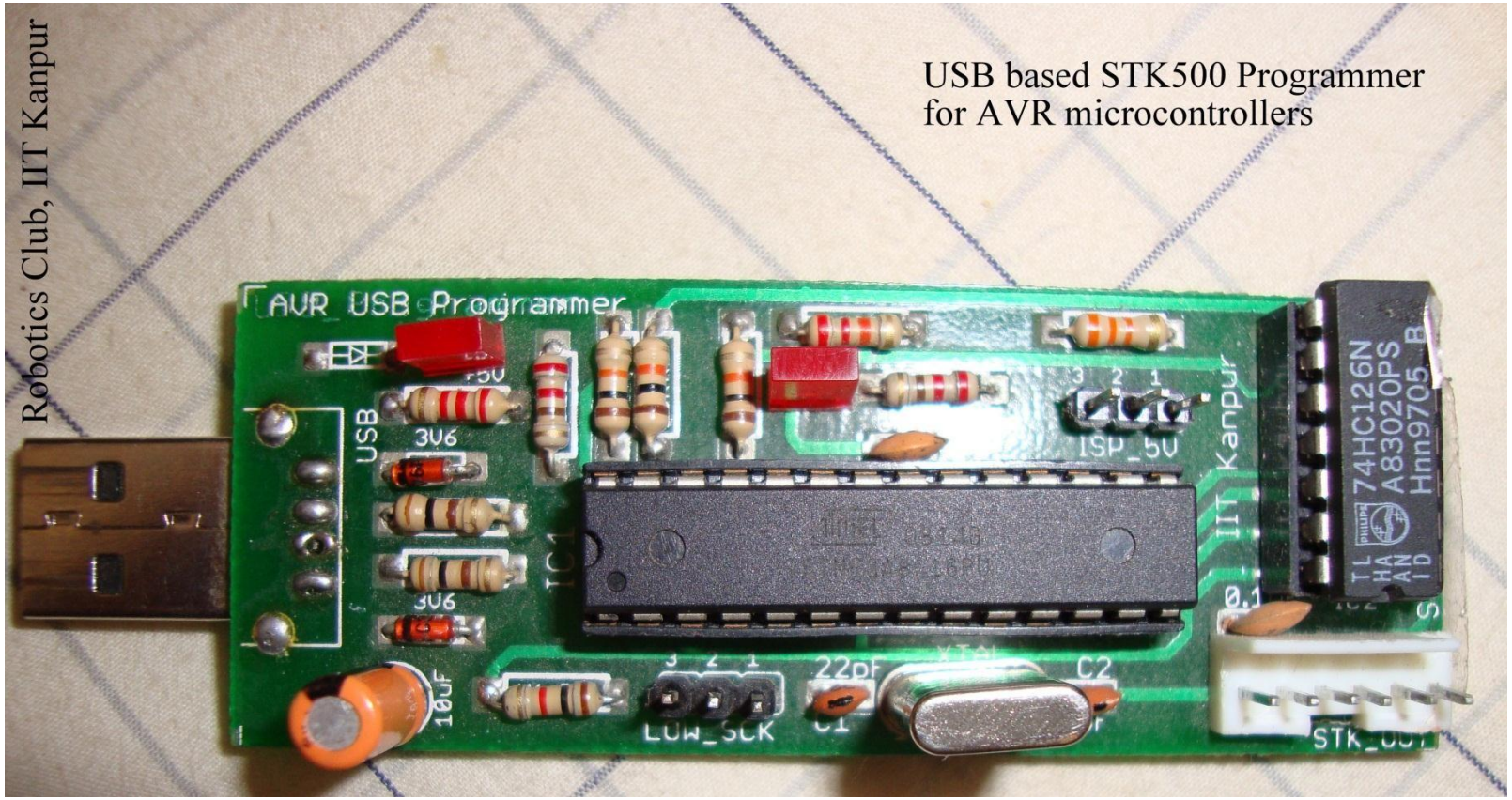
- The code is written in C language so we need to convert it into the format that Atmega understands



Tra Studio


To PCs Serial,Parallel
or USB port
AVR
Prog
AVR Programmer
ISP Headers On Target
ISP Connector Of
Programmer

Avr Programmer

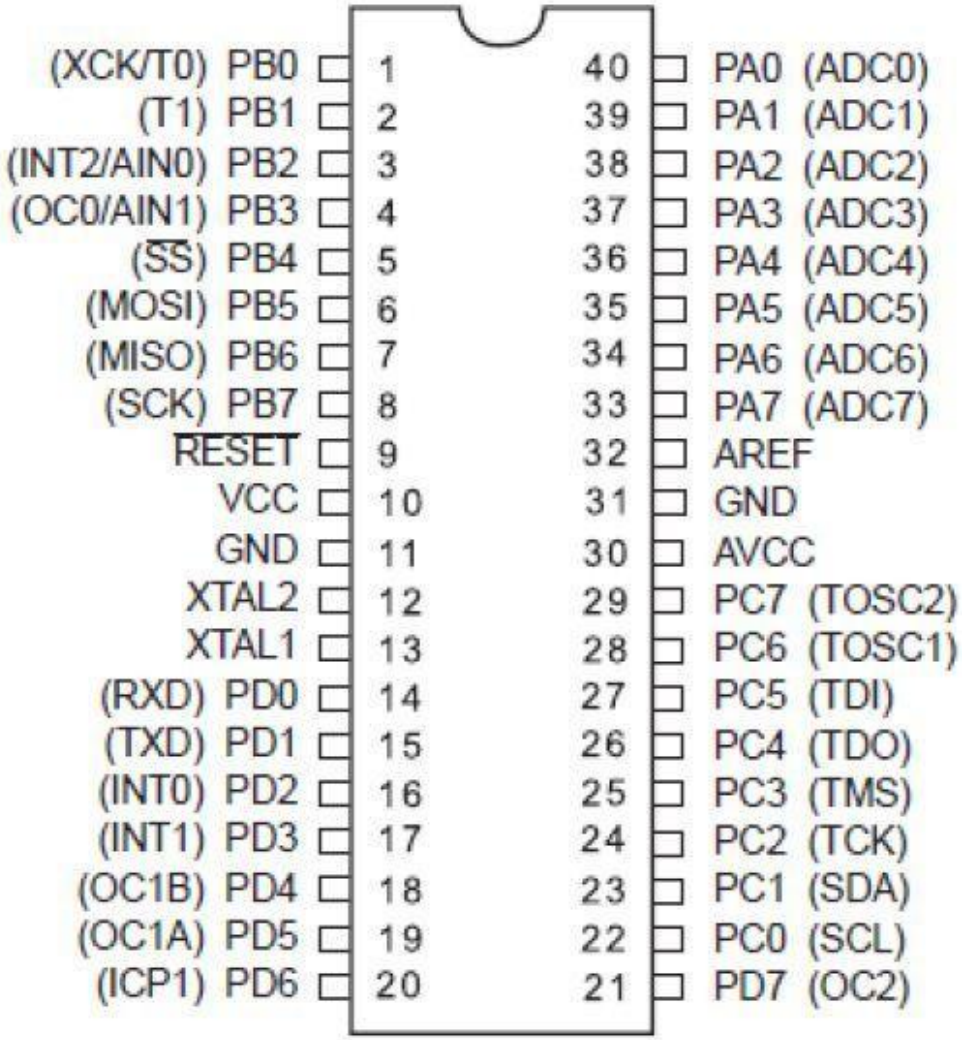


- So we need two softwares overall
    a) CVAVR –> Editor and Compiler
    b) Avr Studio –> Transfer Code to Atmega

Atmega 16

The ATmega16

- 40 pin IC.
- 32 pins for I/O.
- 8 pins reserved.
- I/O pins divided into 4 groups of 8 pins, called ports.
- Ports labeled as A, B, C and D.



Basics of C language

- If else block
- If(condition)
  {
  … …
  }
  else
  {
  … …
  }

While & For

- While (conditon)
  {
  … …
  }

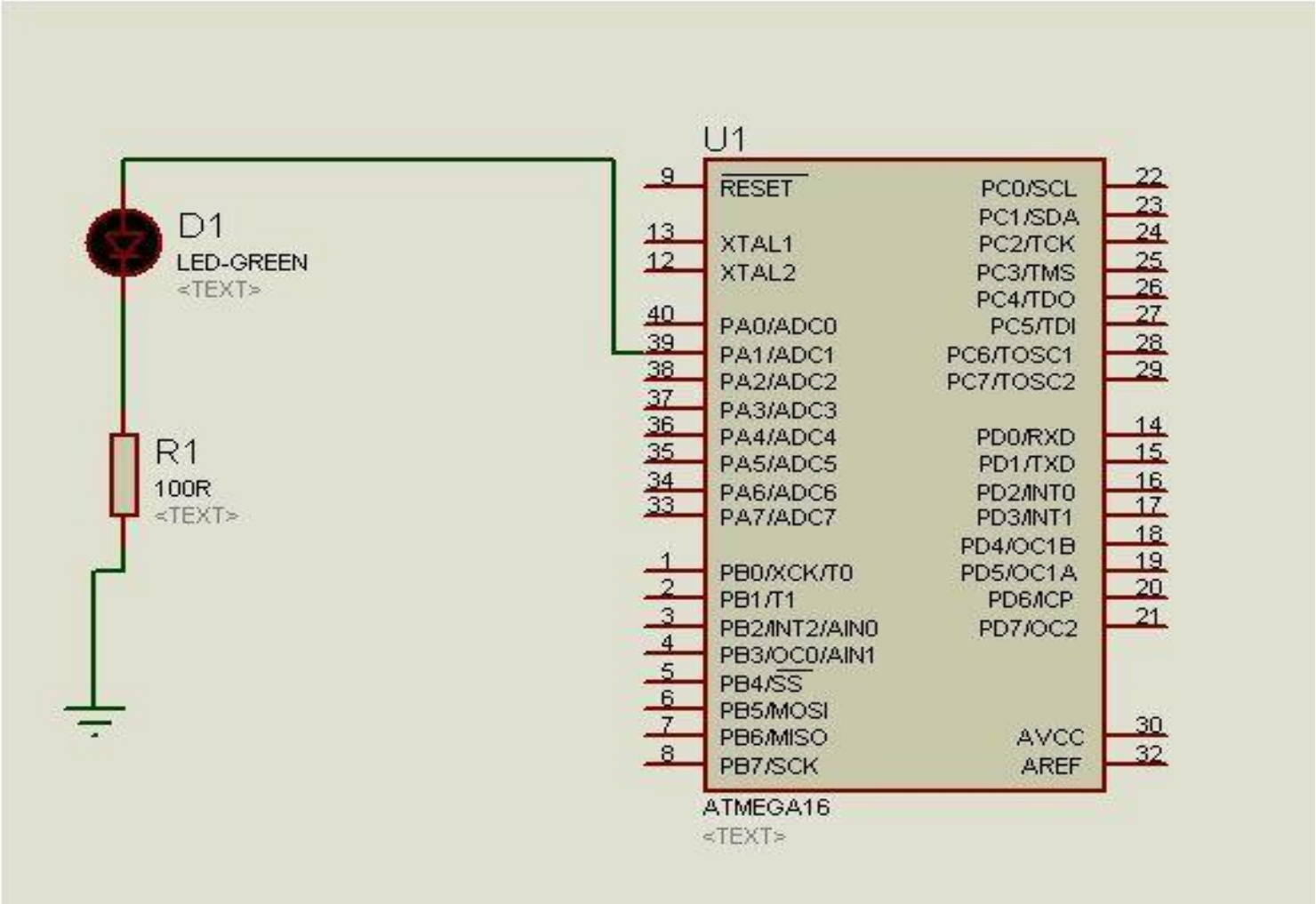
- for(initialisation; condition; increment)
  {
  … …
  }

Some C operators

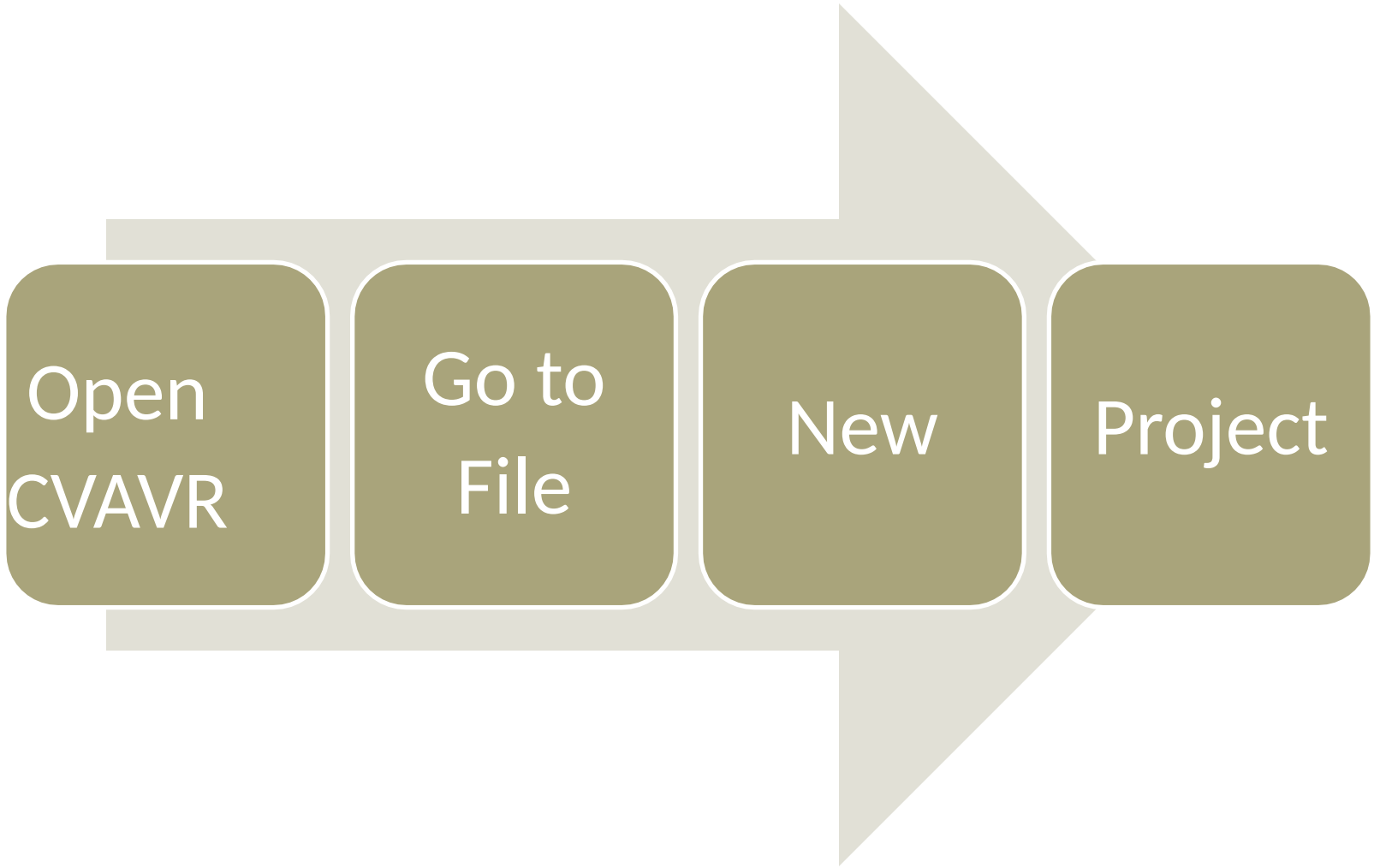
- | is bitwise OR.
  Eg. `10100111 | 11000101 = 11100111`
- & is bitwise AND.
  Eg. `10100111 & 11000101 = 10000101`
- ~ is bitwise NOT.
  Eg. `~10100110 = 01011001`
- << is shift left. >> is shift right.

- Lets Begin by blinking a simple LED

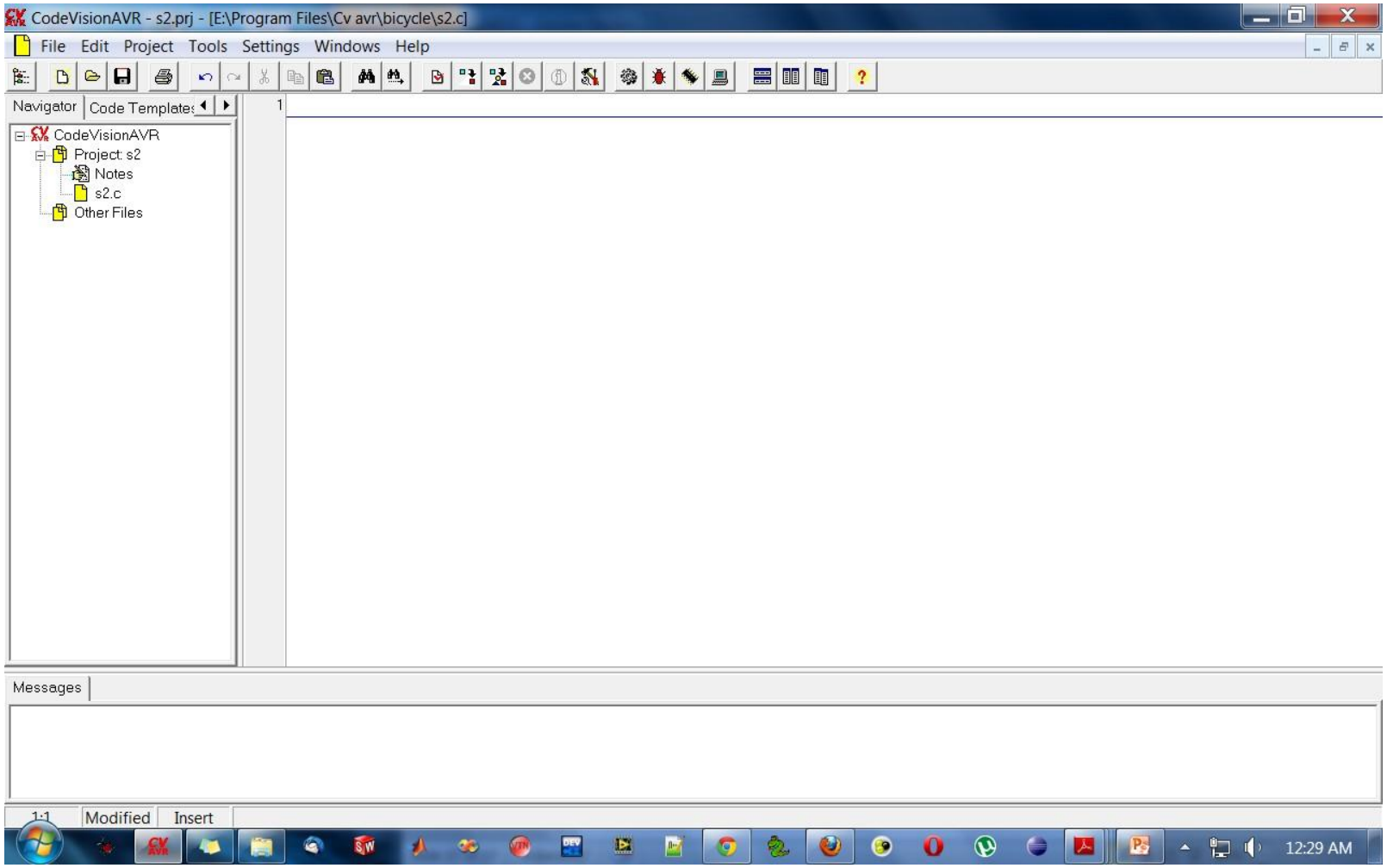
Circuit Diagram



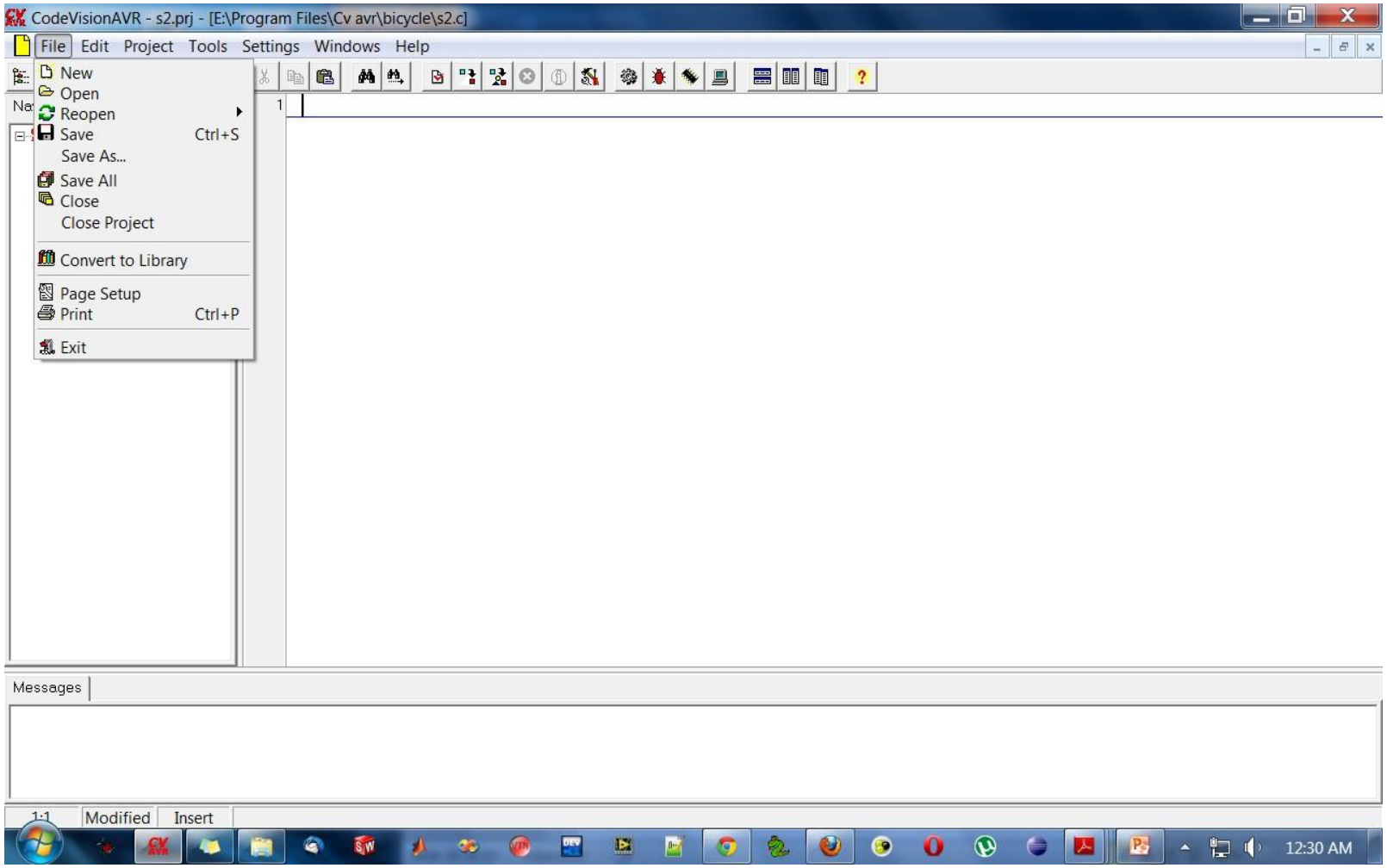
Getting Started with CVAVR


Open CVAVR
Go to File
New
Project

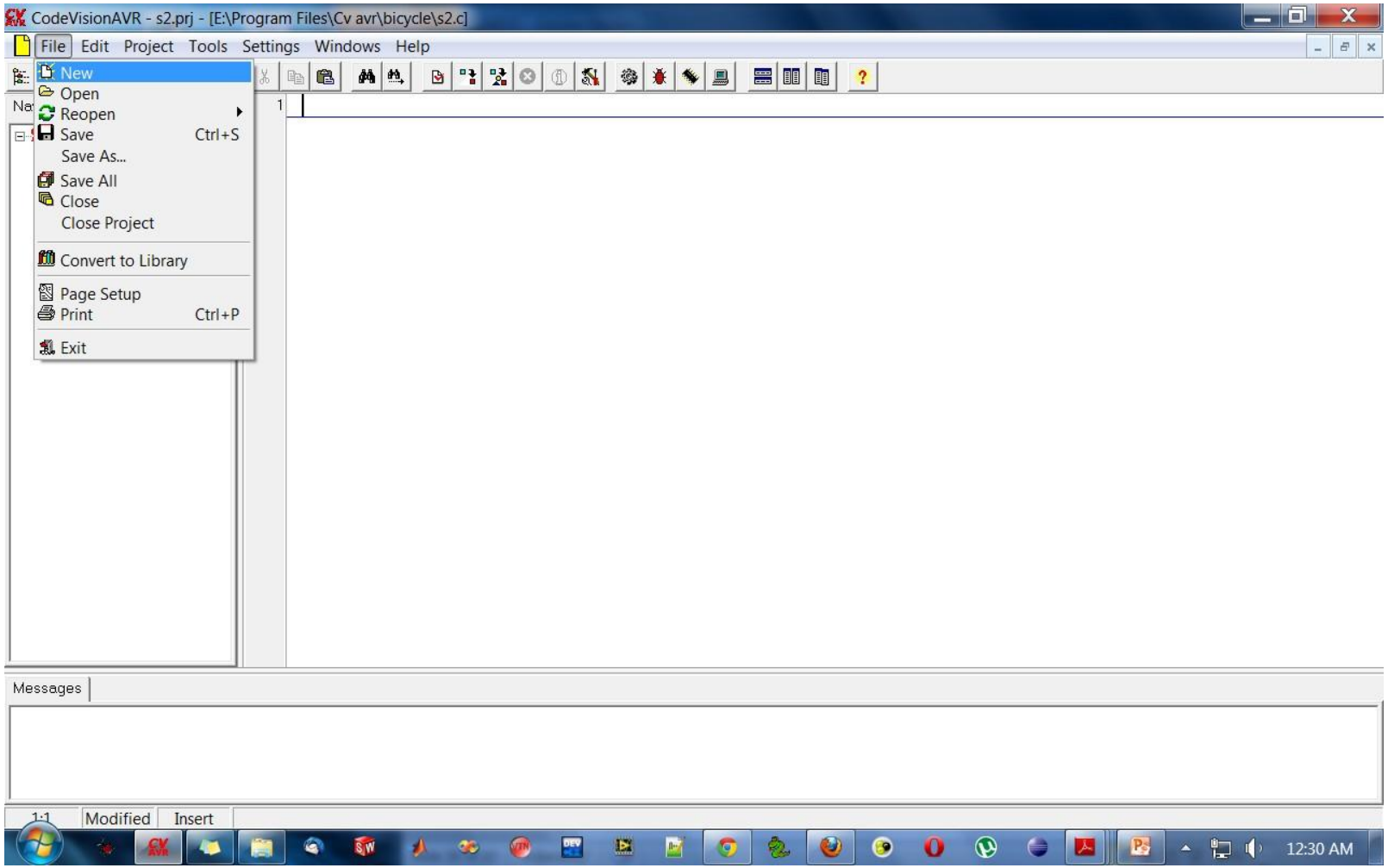
Open CVAVR


CodeVisionAVR - s2.prj - [E:\Program Files\Cv avr\bicycle\s2.c]
File Edit Project Tools Settings Windows Help
Navigator
Code Templates
CodeVisionAVR
Project: s2
Notes
s2.c
Other Files
1
Messages
1:1
Modified
Insert
12:29 AM

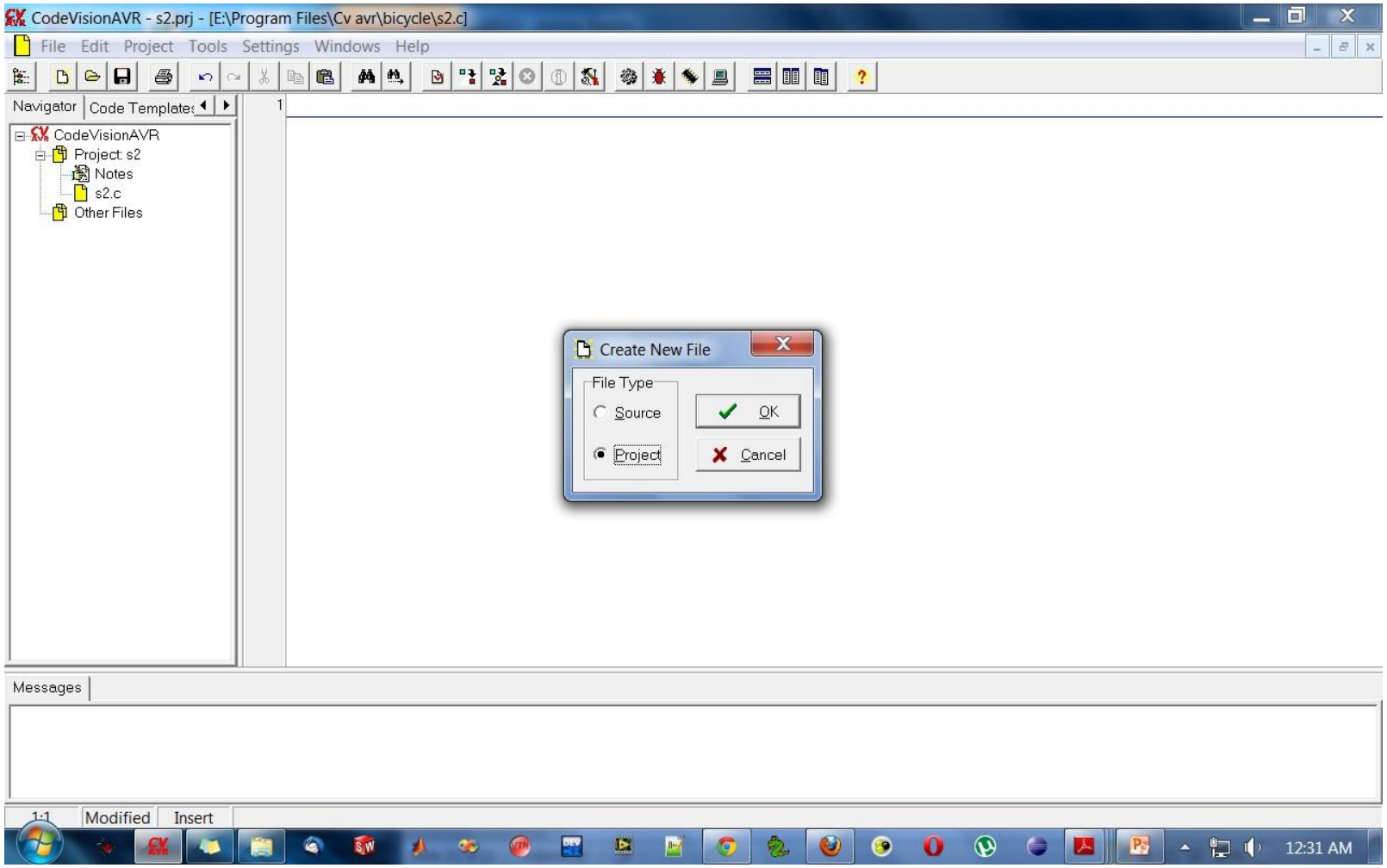
Go to File


CodeVisionAVR - s2.prj - [E:\Program Files\Cv avr\bicycle\s2.c]
File
Edit
Project
Tools
Settings
Windows
Help
New
Open
Reopen
Save Ctrl+S
Save As...
Save All
Close
Close Project
Convert to Library
Page Setup
Print Ctrl+P
Exit
Messages
1:1
Modified
Insert
12:30 AM

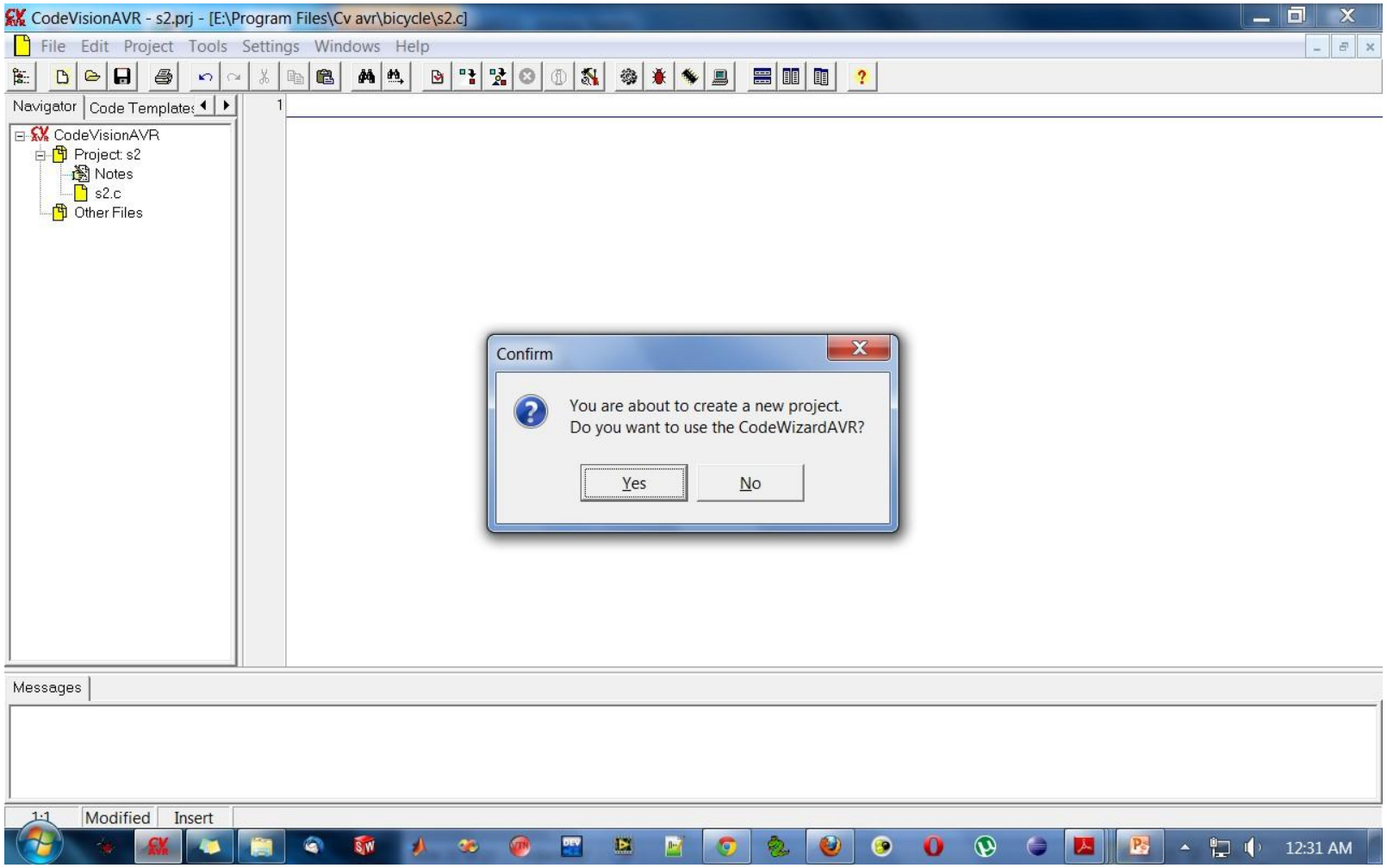
Click on New


CodeVisionAVR - s2.prj - [E:\Program Files\Cv avr\bicycle\s2.c]
File Edit Project Tools Settings Windows Help
New
Open
Reopen
Save Ctrl+S
Save As...
Save All
Close
Close Project
Convert to Library
Page Setup
Print Ctrl+P
Exit
Messages
Modified
Insert
12:30 AM

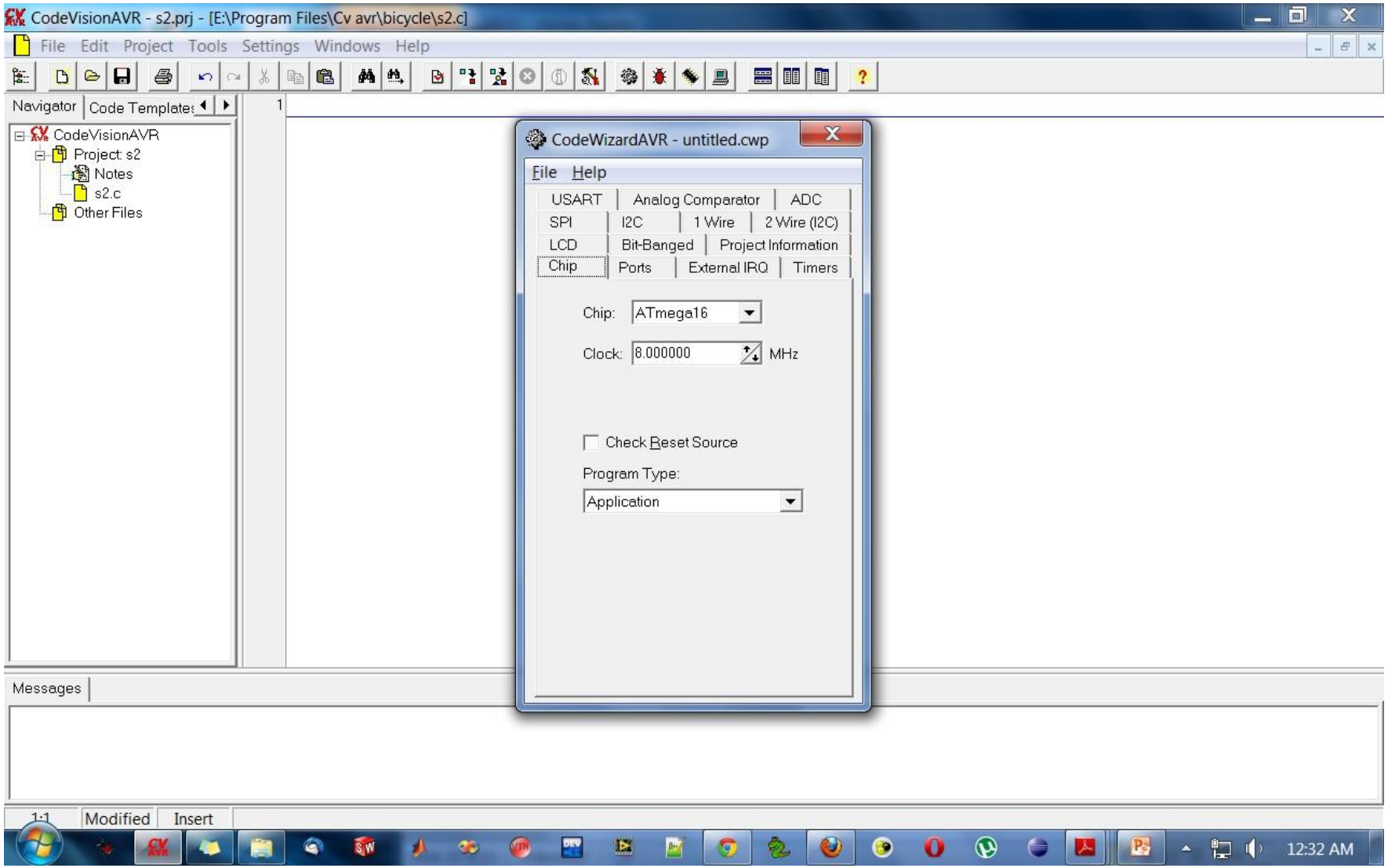
Select Project- > Click OK



Click YES


CodeVisionAVR - s2.prj - [E:\Program Files\Cv avr\bicycle\s2.c]
File Edit Project Tools Settings Windows Help
Navigator
Code Templates
CodeVisionAVR
Project: s2
Notes
s2.c
Other Files
1
Confirm
You are about to create a new project.
Do you want to use the CodeWizardAVR?
Yes
No
Messages
1:1
Modified
Insert
12:31 AM

Select Chip

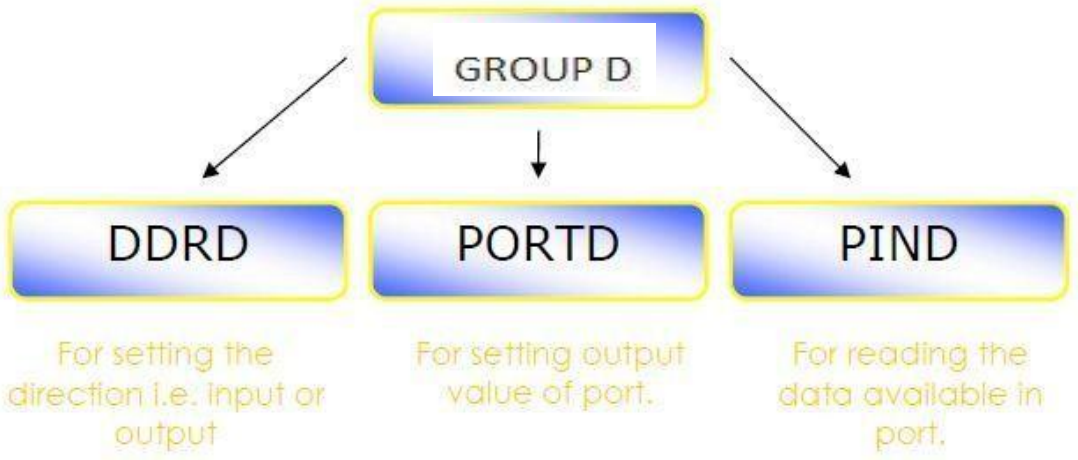

CodeVisionAVR - s2.prj - [E:\Program Files\Cv avr\bicycle\s2.c]
File Edit Project Tools Settings Windows Help
Navigator
Code Templates
CodeVisionAVR
Project: s2
Notes
s2.c
Other Files
CodeWizardAVR - untitled.cwp
File Help
USART
Analog Comparator
ADC
SPI
I2C
1 Wire
2 Wire (I2C)
LCD
Bit-Banged
Project Information
Chip
Ports
External IRQ
Timers
Chip: ATmega16
Clock: 8.000000 MHz
Check Reset Source
Program Type:
Application
Messages
Modified
Insert
12:32 AM

Introduction to I/O

- Atmega has total of 40 pins out of which 32 pins can be used as Input or Output
- These 32 pins are divided into 4 groups of 8 pins PORTA, PORTB , PORTC , PORTD

Accessing digital IO in C

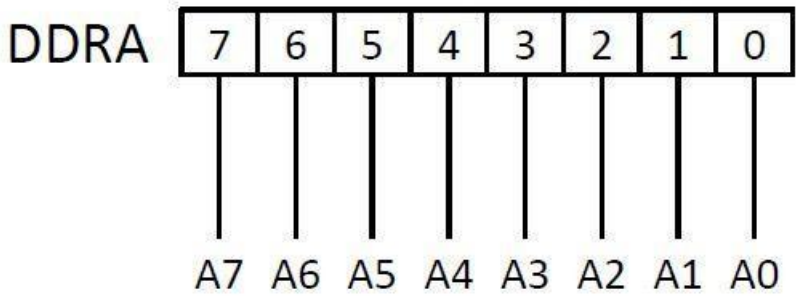
Each PORT in AVR has three related Registers.



Data Direction register (DDR)

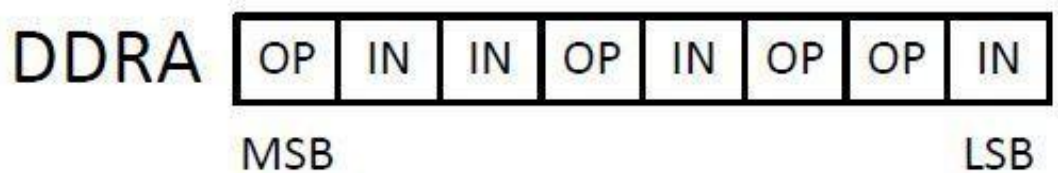
- This sets direction for all pins (32)
- Direction for these pins can be Input or Output
- To blink an LED we need to set pin as “OUTPUT” but “HOW“ ?

- DDRA = 0b00000001 ;
- DDRA = 0x01 ;
- 1 Stands for Output & 0 stands for Input



Interpretation of DDR values

- If a bit on the **DDR** register is `0`, then the corresponding pin on the associated port is set as input.
- Similarly, if the bit is `1`, then the pin is set as output.
- Example: if `DDRA = 0b10010110`, then:



What is Next ?

- We have set the Pin as Output
- What else do we need to light the LED ??
- Supply of 5 Volts !!! This is given by PORT Register

PORT Register

- Only after you have set the Pin to Output you can control them through this Register
- It is a 8 bit register . It corresponds to the pin in same manner as that of DDR Register
- Used to set output value (0 or 1) only if the corresponding Pin has been set as output by DDR Register
- PORTA= 0b 00000001;
  or
- PORTA= 0x01 ;
- 1 stands for 5V
- 0 stands for 0V

Simple Questions

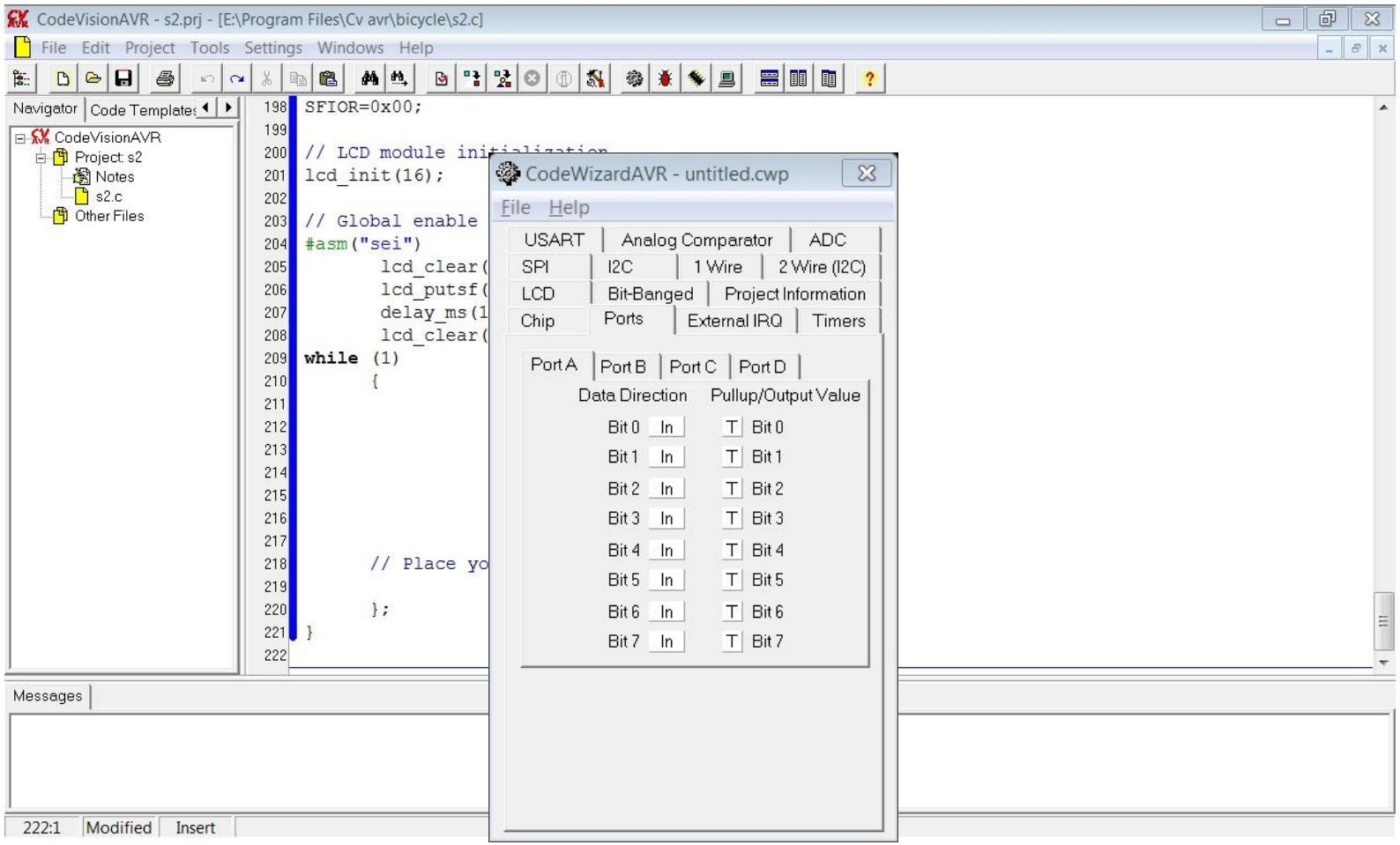
- DDRA= 0b 00101100
- DDRD = 0xf4
- DDRC = 0b 01111110
- DDRB = 0x3b

Assume all 32 pins set as output

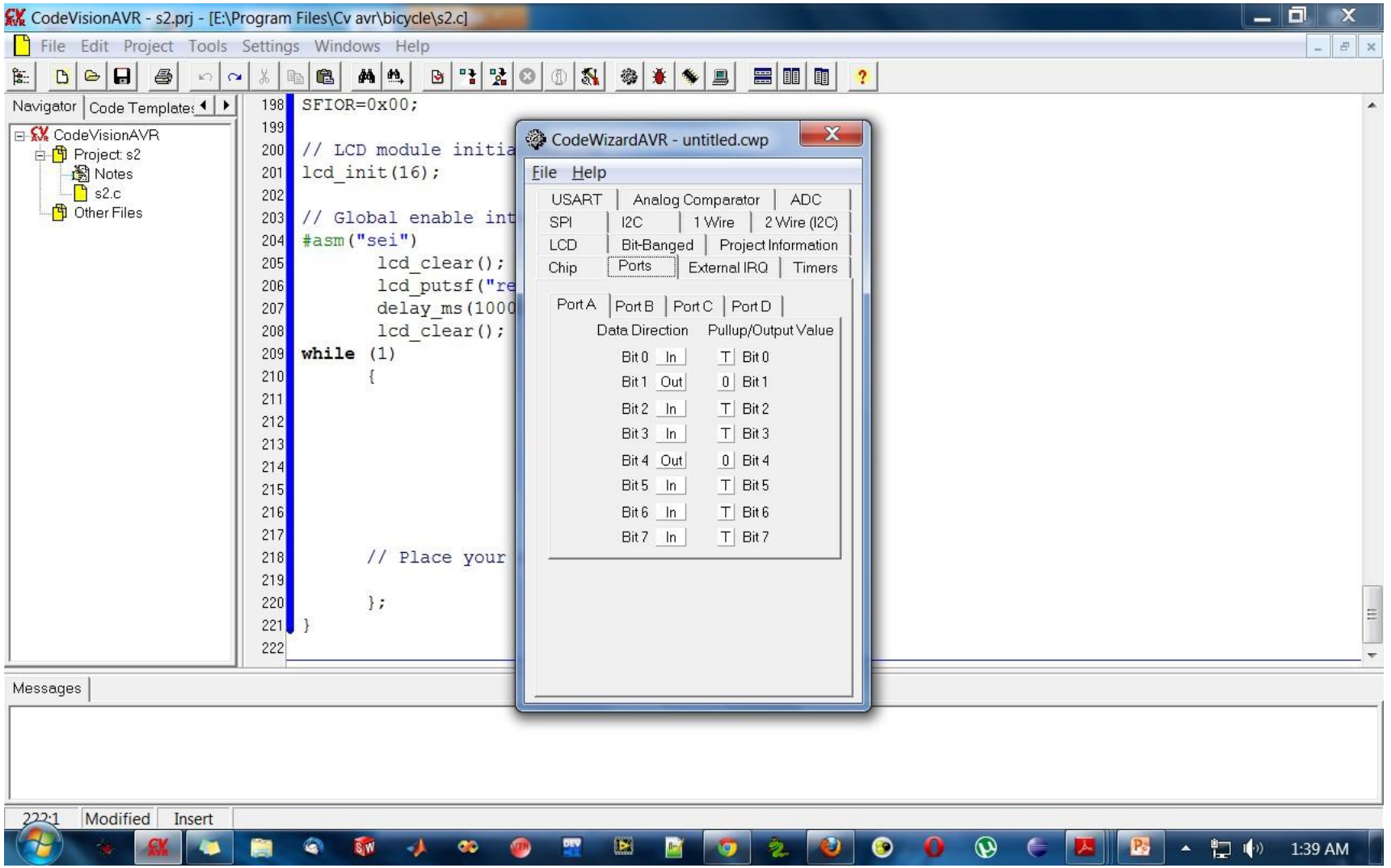
- PORTA = 0b00001100;
- PORTD = 0b11110000;
- PORTB.4=1;
- PORTC.2=1;

Setting I/O

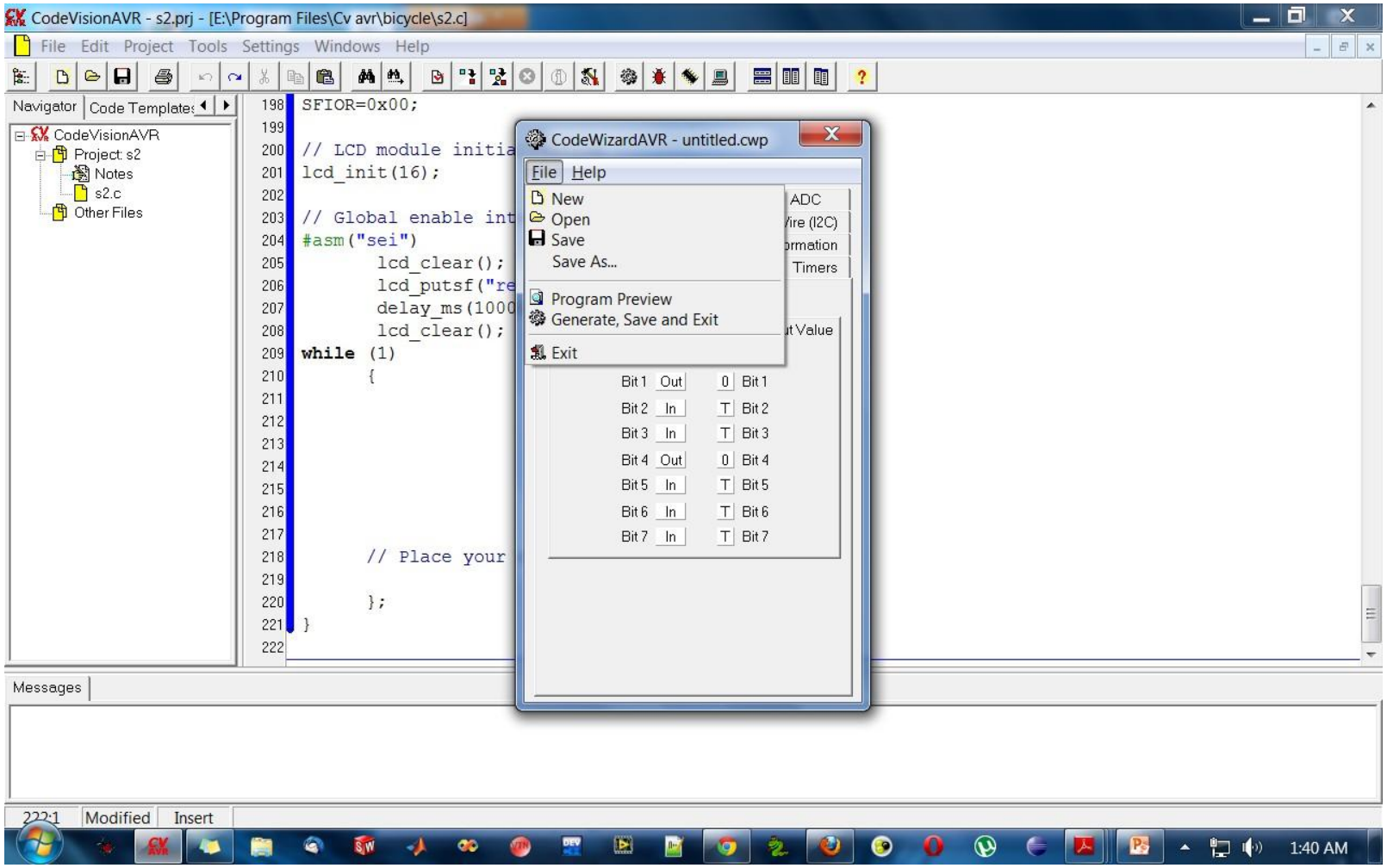
Go to Ports


CodeVisionAVR - s2.prj - [E:\Program Files\Cv avr\bicycle\s2.c]
File Edit Project Tools Settings Windows Help
Navigator
Code Templates
CodeVisionAVR
Project: s2
Notes
s2.c
Other Files
198 SFIOR=0x00;
199
200 // LCD module initialization
201 lcd_init(16);
202
203 // Global enable
204 #asm("sei")
205 lcd_clear(
206 lcd_putsf(
207 delay_ms(1
208 lcd_clear(
209 while (1)
210 {
211
212
213
214
215
216
217
218 // Place yo
219
220 };
221 }
222
CodeWizardAVR - untitled.cwp
File Help
USART
Analog Comparator
ADC
SPI
I2C
1 Wire
2 Wire (I2C)
LCD
Bit-Banged
Project Information
Chip
Ports
External IRQ
Timers
Port A
Port B
Port C
Port D
Data Direction
Pullup/Output Value
Bit 0 In T Bit 0
Bit 1 In T Bit 1
Bit 2 In T Bit 2
Bit 3 In T Bit 3
Bit 4 In T Bit 4
Bit 5 In T Bit 5
Bit 6 In T Bit 6
Bit 7 In T Bit 7
Messages
222:1
Modified
Insert

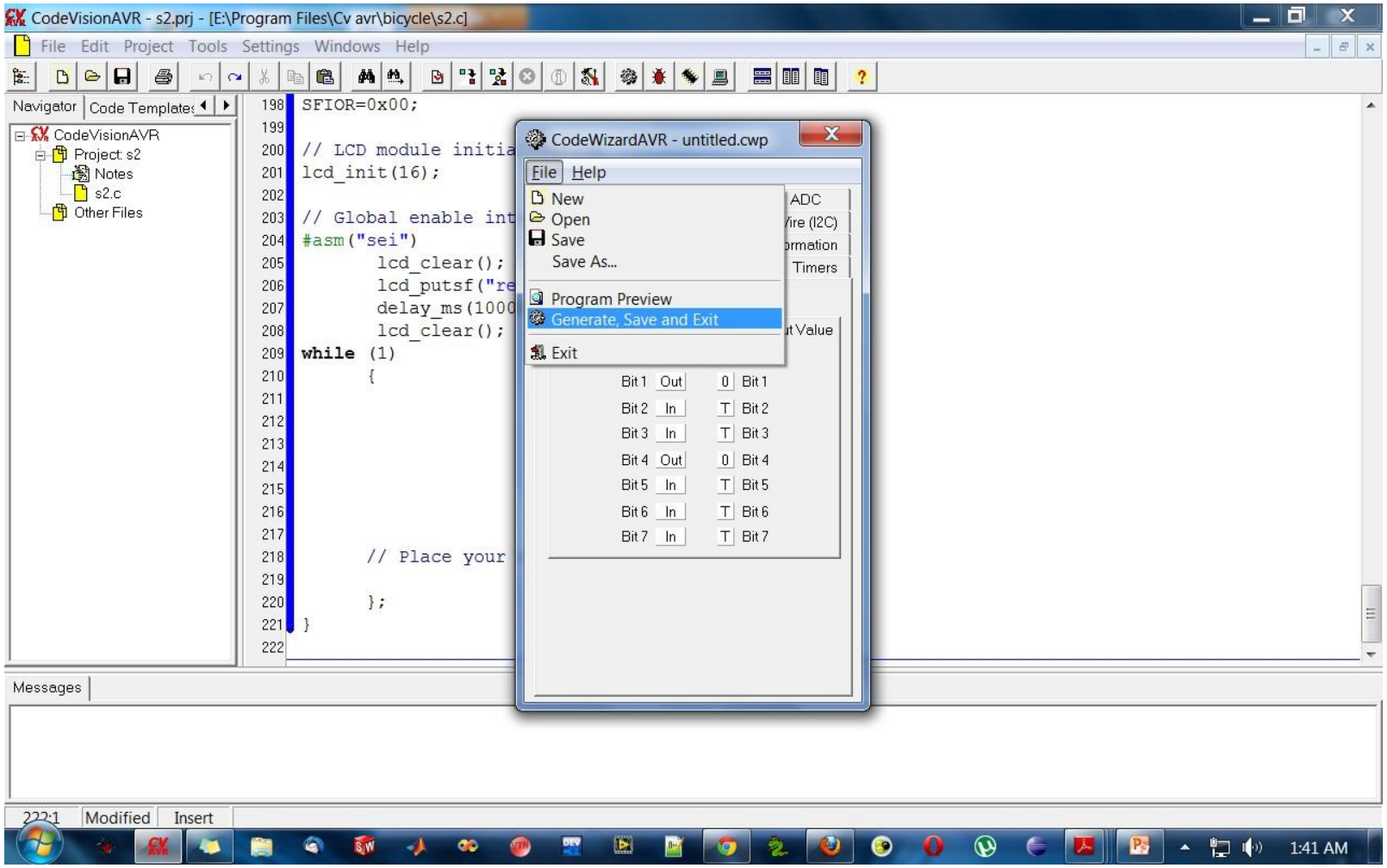
- Click on In to make that pin Output
- Can do so for all four ports



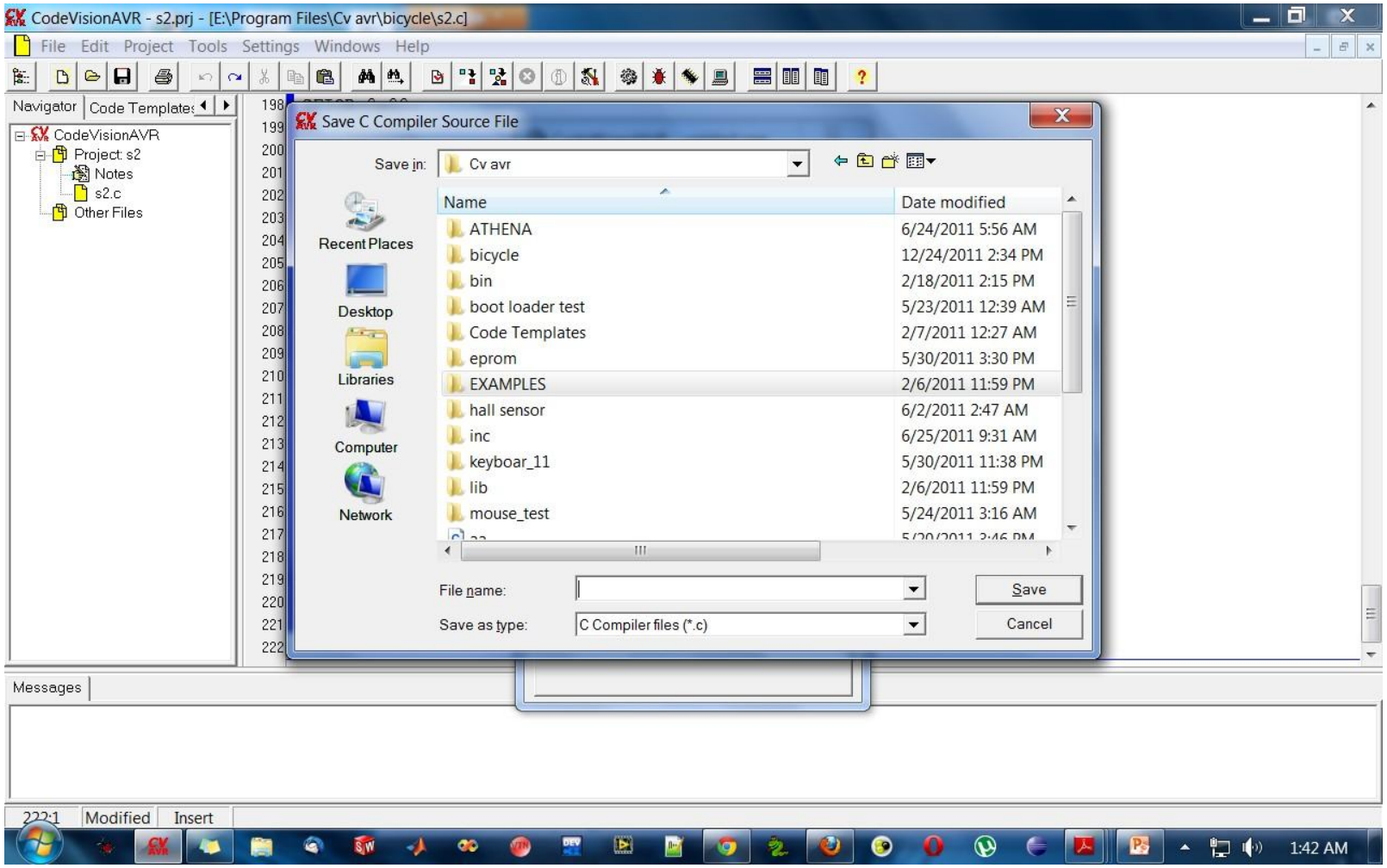
Click on File



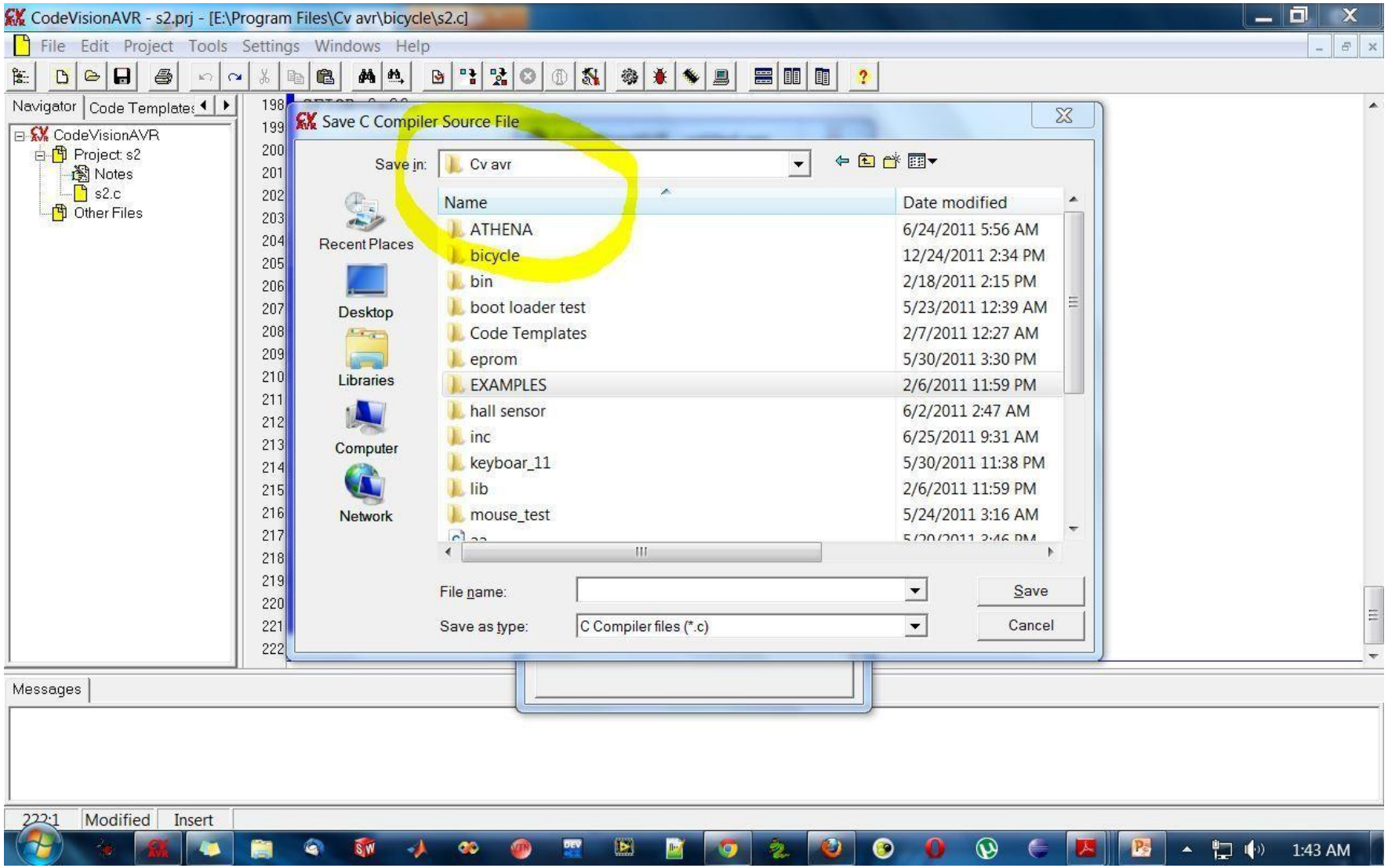
Generate Save and Exit



Enter name (3 times)

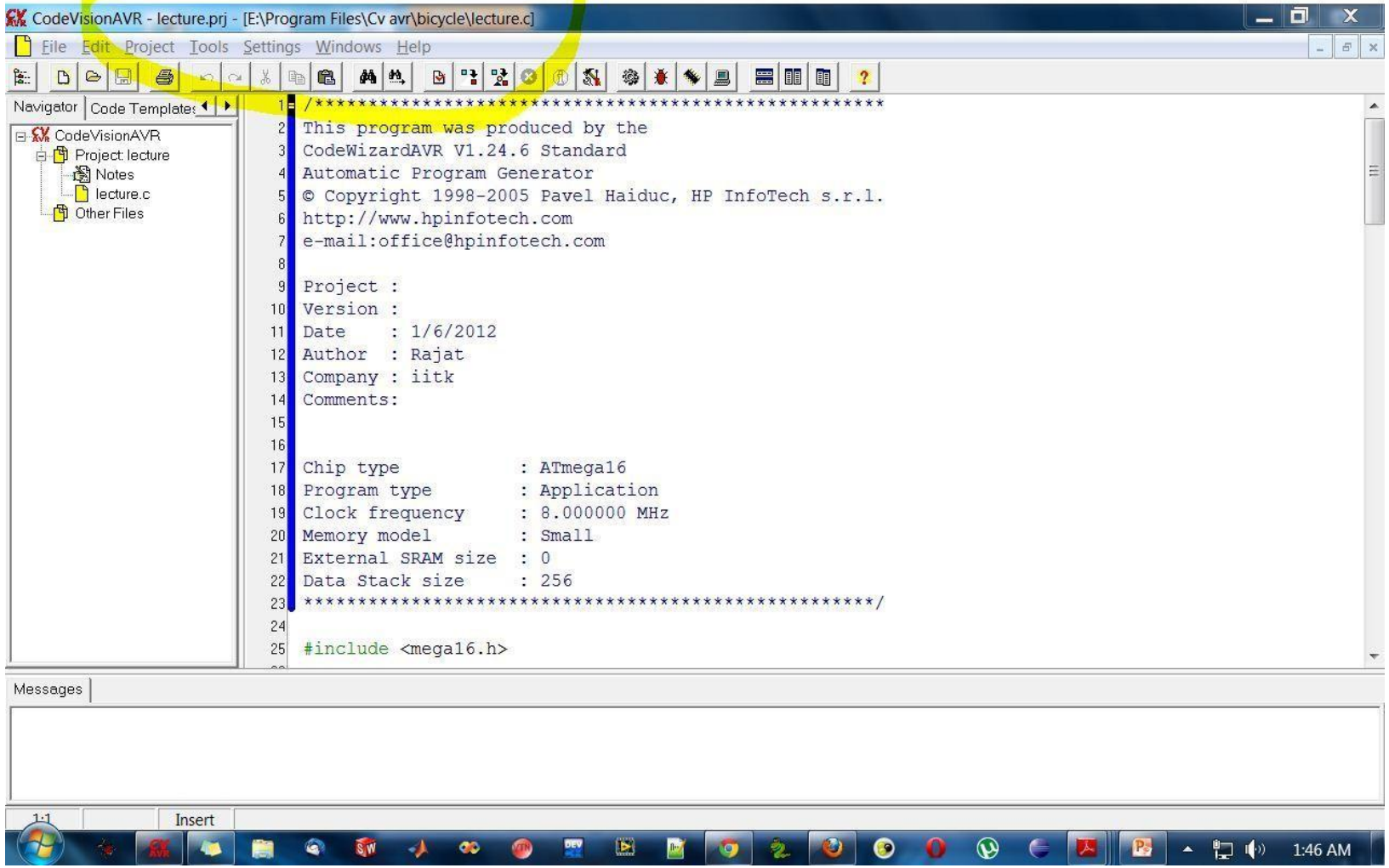


Where is the code stored ?



Then Click Save

Name of Project & Location



Writing the Code

- NOTE : We write our code in While block
- While (1)
  {
  PORTA.1=1; // sets the Pin to 5 volts
  PORTA.1=0; // sets the Pin to 0 volts
  }

- This makes the LED to blink but we cannot see blinking !!!

- This is because Atmega runs at a frequency of 8000000 Hz
- We need to introduce delay so as to see blinking
- Use header file delay.h

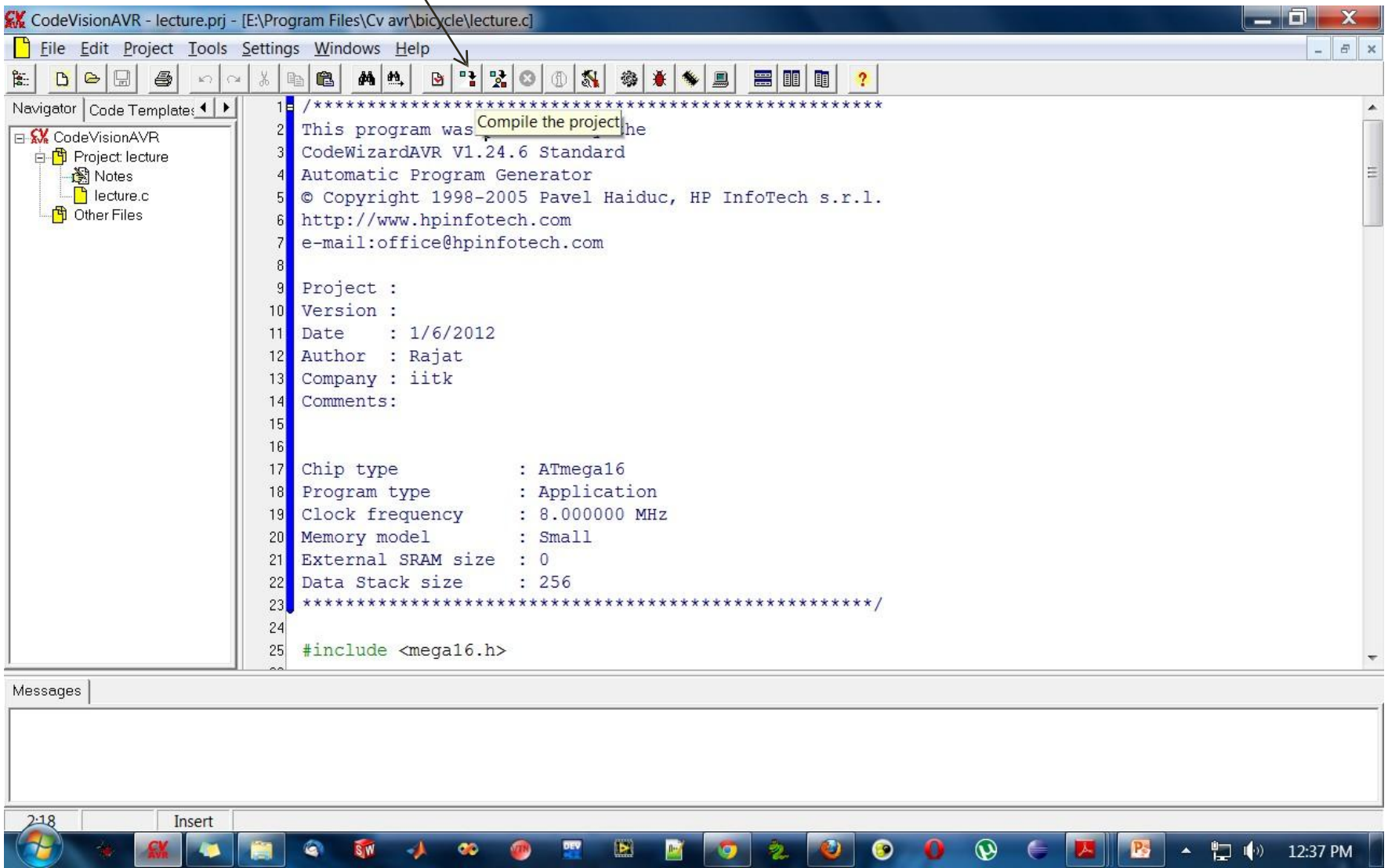
•Function to be used delay_ms(time in millis);

```
While (1)
{
delay_ms(1000);
PORTA.1=1;
delay_ms(1000);
PORTA.1=0;
}
```

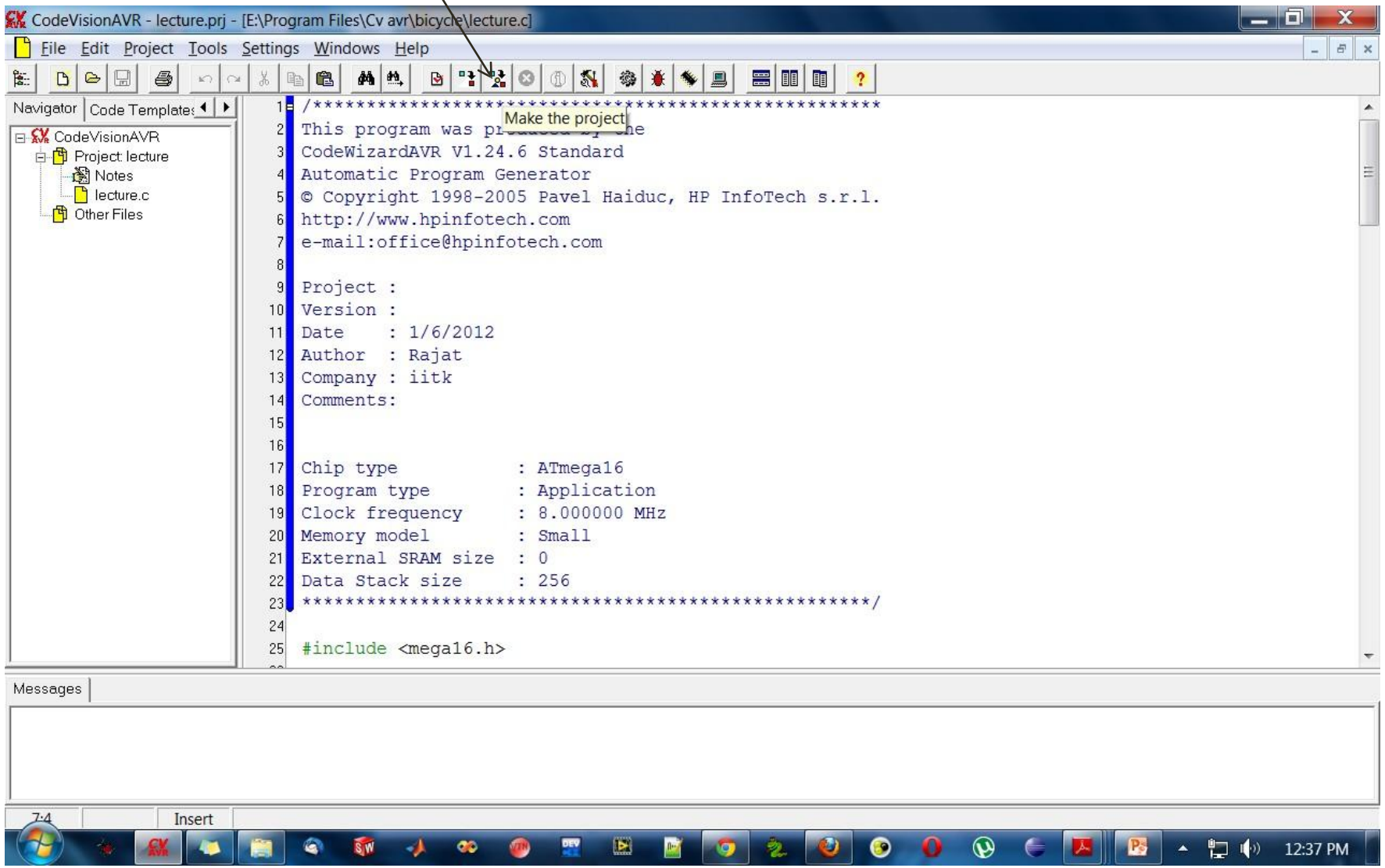
How to compile

- Code is written in C language but Atmega understands Hex file
  so we need to convert the C file to Hex file

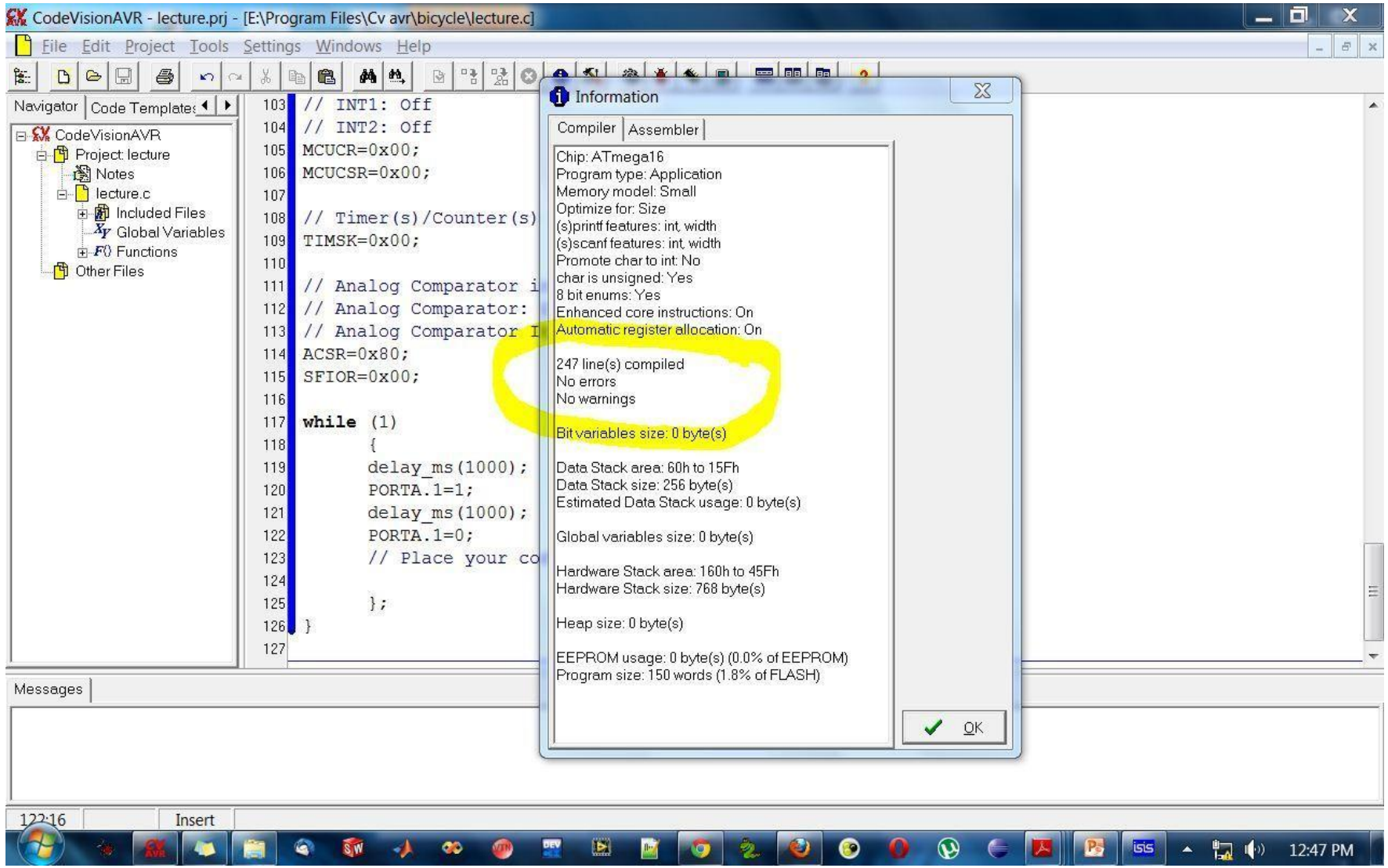
Compiling


CodeVisionAVR - lecture.prj - [E:\Program Files\Cv avr\bicycle\lecture.c]
File Edit Project Tools Settings Windows Help
Compile the project
Navigator
Code Templates
CodeVisionAVR
Project: lecture
Notes
lecture.c
Other Files
/***
This program was
CodeWizardAVR V1.24.6 Standard
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com
Project :
Version :
Date : 1/6/2012
Author : Rajat
Company : iitk
Comments:
Chip type : ATmega16
Program type : Application
Clock frequency : 8.000000 MHz
Memory model : Small
External SRAM size : 0
Data Stack size : 256
***/
#include <mega16.h>
Messages
2:18
Insert
12:37 PM

Make the Project



Check for errors



Hex File

- You can find the Hex file in Bin folder or the EXE folder of the directory where You installed CVAVR

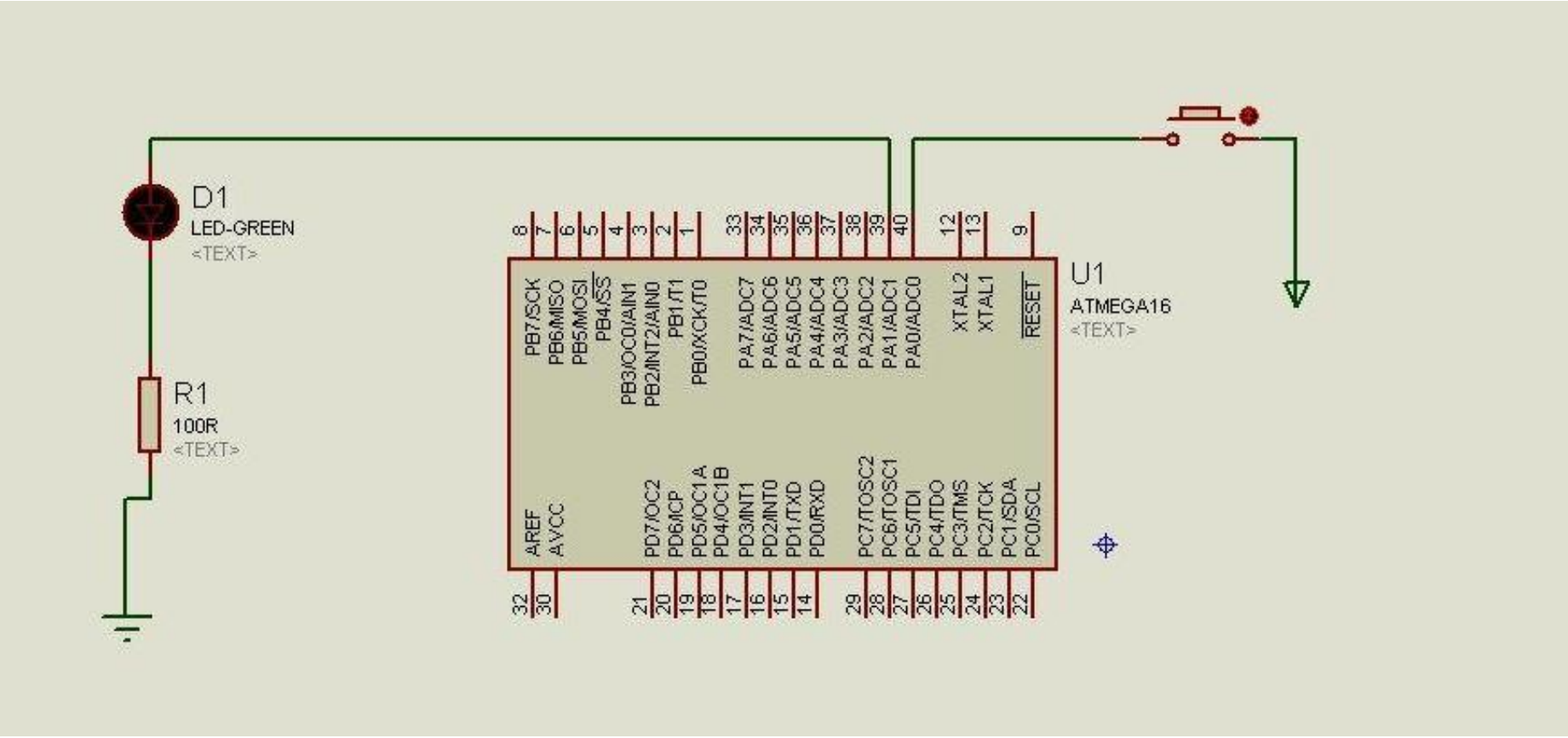
- So we Have our Code ready
- Feed this code to Atmega using Programmer (we will see this in workshop)
- Lets see the code in action

Lets add an Input

- Most Common Input Button

- Since we have already made A0 as Input
- We connect a button to that pin
- If button is pressed light the LED else turn it off
- First draw the Circuit Diagram

Circuit Diagram



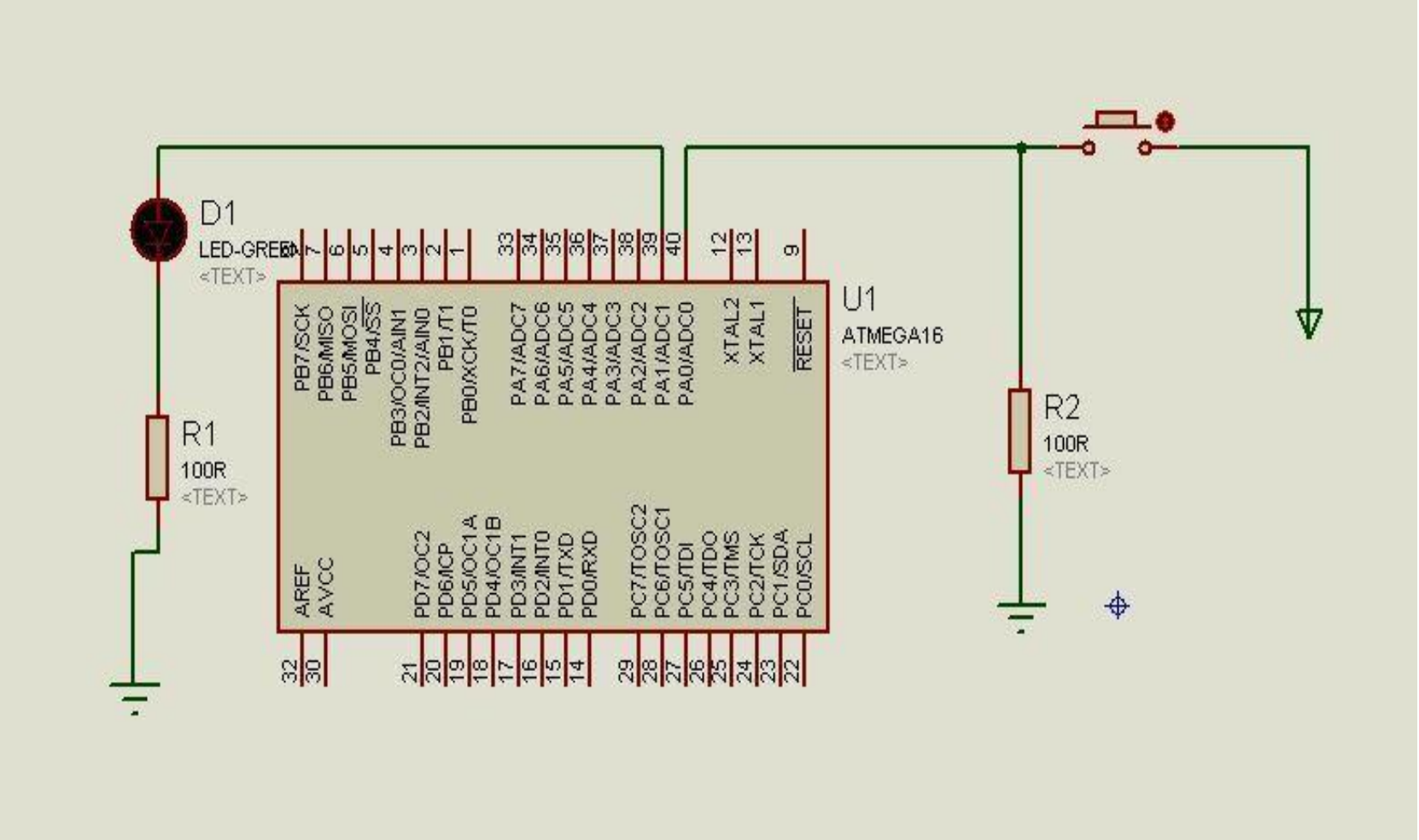
- Never leave any Input pin unconnected / floating at any point of time while your circuit is working
- In Last Circuit A0 is floating when button is not pressed so our Circuit Diagram is wrong

- What is the Voltage at the Floating PIN ?

- Not 5 V
- Not 0V
- Its UNDEFINED
- So never leave an input pin unconnected
- Use the Concept of Pull up / Pull down

- In Layman terms
  - PULL DOWN : Gives 0V when unconnected
  - PULL UP : Gives 5V when unconnected
- Connect the PIN to Ground through a resistance for pulling down
- Connect the PIN to 5V through a resistance for Pulling up

Correct Circuit Diagram



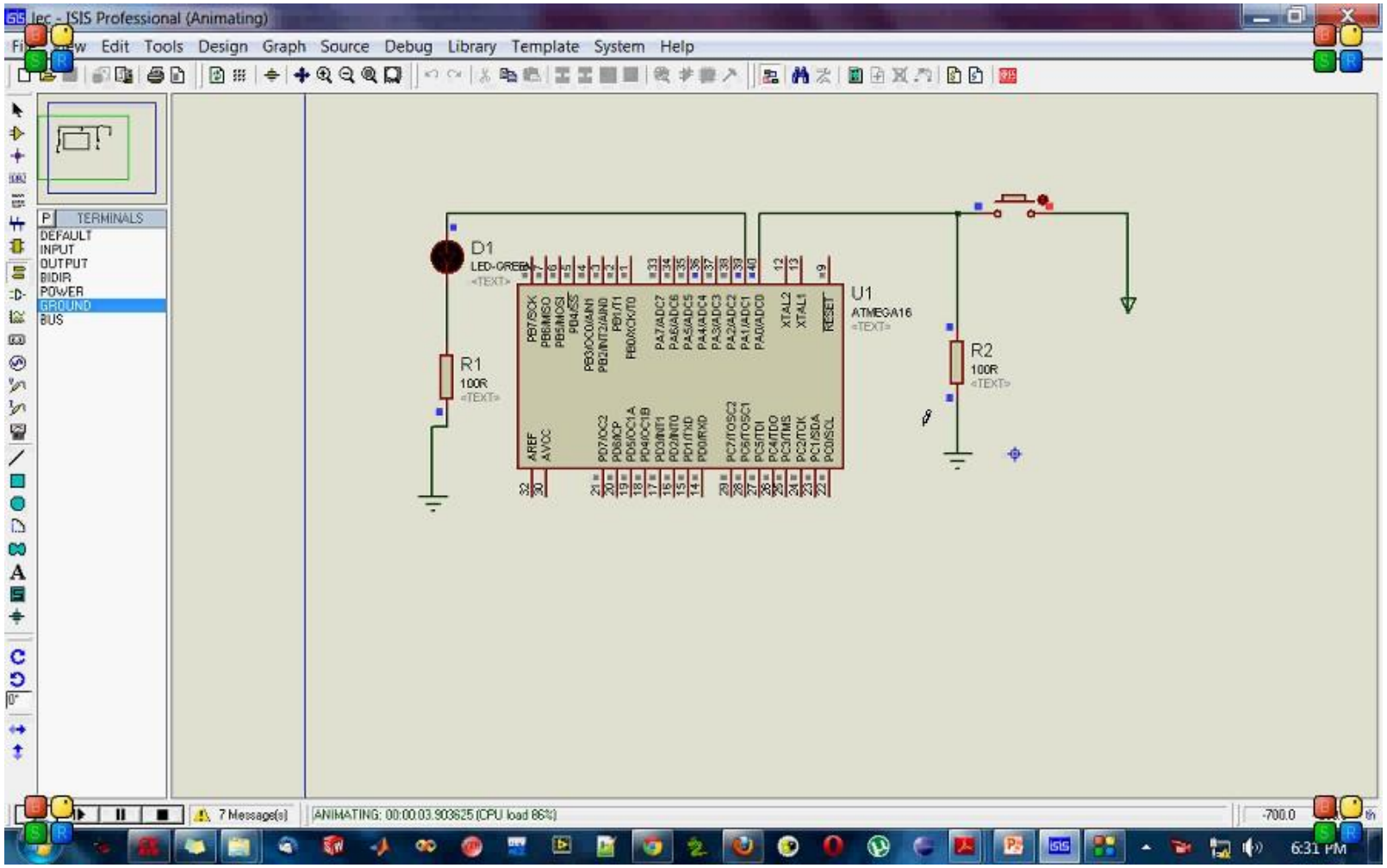
PIN Register

- It is a 8 bit register . It corresponds to the pin in same manner as that of DDR Register
- It is used to read voltage at a pin
- To be used only after the pin has been set as input by DDR register

Using Pin Register

```
int a;     // Define the variable a to store the value of
voltage  a=PINA.0;   // read value at pin A.0 (make sure it is
input)
If (a==1) // if voltage is 5V
{
PORTA.1=1; // Light the LED
}
else
{
PORTA.1=0; // Turn off the LED
}
```

Code in Action



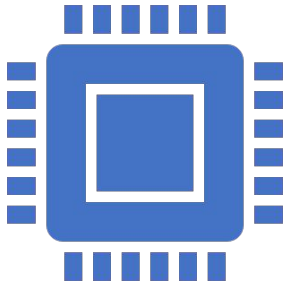
1) What is a Microprocessor?

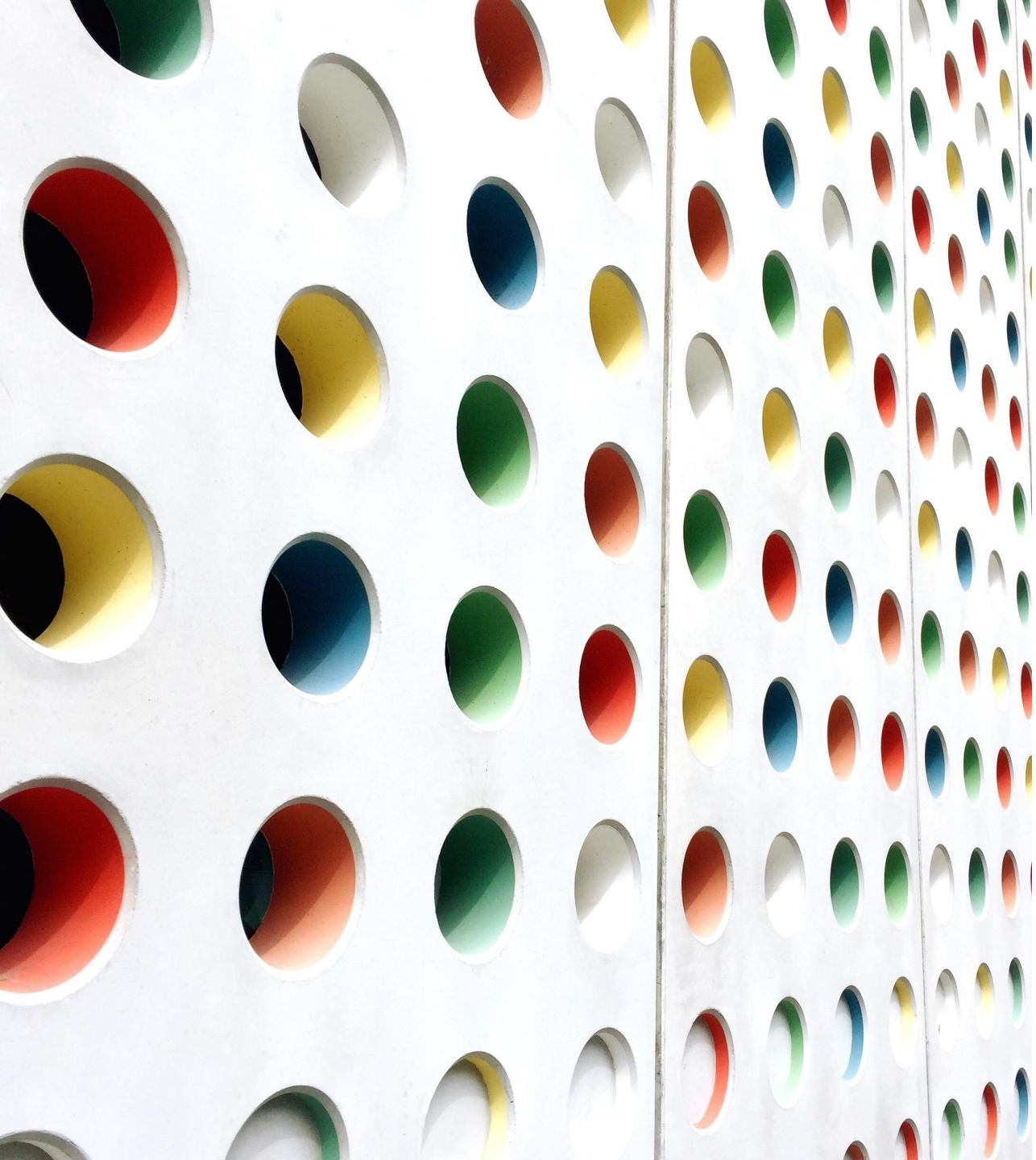
- In simple words, The microprocessor is useful in very intensive processes. It only contains a CPU (central processing unit) but there are many other parts needed to work with the CPU to complete a process. These all other parts are connected externally.
- Microprocessors are not made for a specific task as well as they are useful where tasks are complex and tricky like the development of software, games, and other applications that require high memory and where input and output are not defined.
- Do you understand? I think a bit, but it's ok, let’s understand by some daily life examples
- A) Household devices: Complex home security, Home computers, Video game systems and many more.
- B) Transportation and Industrial Devices: Automobiles, trains, planes, Computer servers, high tech medical devices, etc.
- Did you notice! All the above applications are complex and they need to process all complicated data.

- **2) What is Microcontroller?**
- The microcontroller is designed for a specific task or to perform the assigned task repeatedly. Once the program is embedded on a microcontroller chip, it can’t be altered easily and you may need some special tools to reburn it. As per application, the process is fixed in microcontroller. Hence, the output depends on the input given by the user or sensors or predefined inputs.
- The applications easily connect with concepts, so let's find out day to day life examples
- e.g. Calculator, Washing Machine, ATM machine, Robotic Arm, Camera, Microwave oven, Oscilloscope, Digital multimeter, ECG Machine, Printer so on and so forth.

- Microprocessor consists of only a Central Processing Unit, whereas Micro Controller contains a CPU, Memory, I/O all integrated into one chip.
- The microprocessor is useful in Personal Computers whereas Micro Controller is useful in an embedded system.
- The microprocessor uses an external bus to interface to RAM, ROM, and other peripherals, on the other hand, Microcontroller uses an internal controlling bus.
- Microprocessors are based on Von Neumann model Microcontrollers are based on Harvard architecture
- The microprocessor is complicated and expensive, with a large number of instructions to process but Microcontroller is inexpensive and straightforward with fewer instructions to process.

LECTURE 15 APPLICATIONS OF MICROPROCESSORS MICROCONTROLLERS

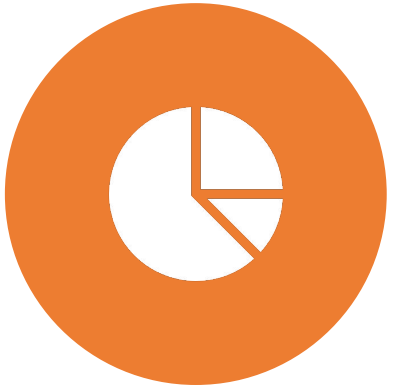




Applications of Microprocessors & Microcontrollers

Dana Utebayeva

Outline



PC, LAPTOPS AND TOP MODELS OF PROCESSORS

MOBILE PHONES

AI, MACHINE LEARNING AND DEEP LEARNING, COMPUTER VISION PROJECTS

Best Processors of PCs and Laptops

Intel Core I9-9980XE Extreme Edition Processor

Intel Core I9-7920X X-Series Processor

AMD YD297XAZAFWOF Ryzen Threadripper

AMD Ryzen 5 3600 6-Core

AMD Ryzen 5 2600

Mobile Processor

- **The mobile processor is used in mobile computers and cell phones. The CPU IC is designed for laptop computers to run without a fan, with a power rating of less than 10-15W, which is cool enough without a fan.**

5 chips (in alphabetic order)especially designed for AI

AMD Radeon Instinct

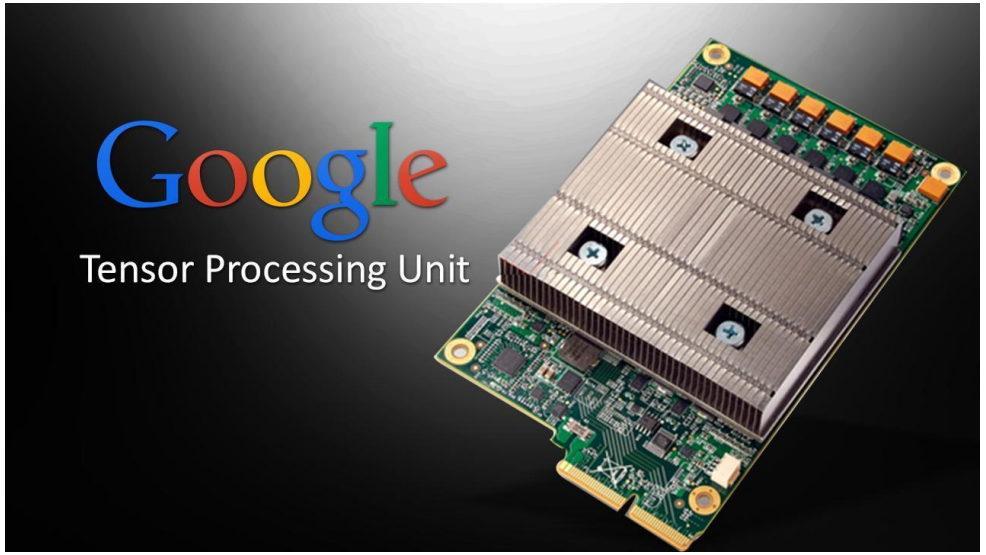
Usage

- Radeon Instinct is AMD's brand of deep learning oriented GPUs. It replaced AMD's FirePro S brand in 2016. Compared to the Radeon brand of mainstream consumer/gamer products, the Radeon Instinct branded products are intended to accelerate deep learning, artificial neural network, and high-performance computing/GPGPU applications.

Apple A11 Bionic Neural Engine

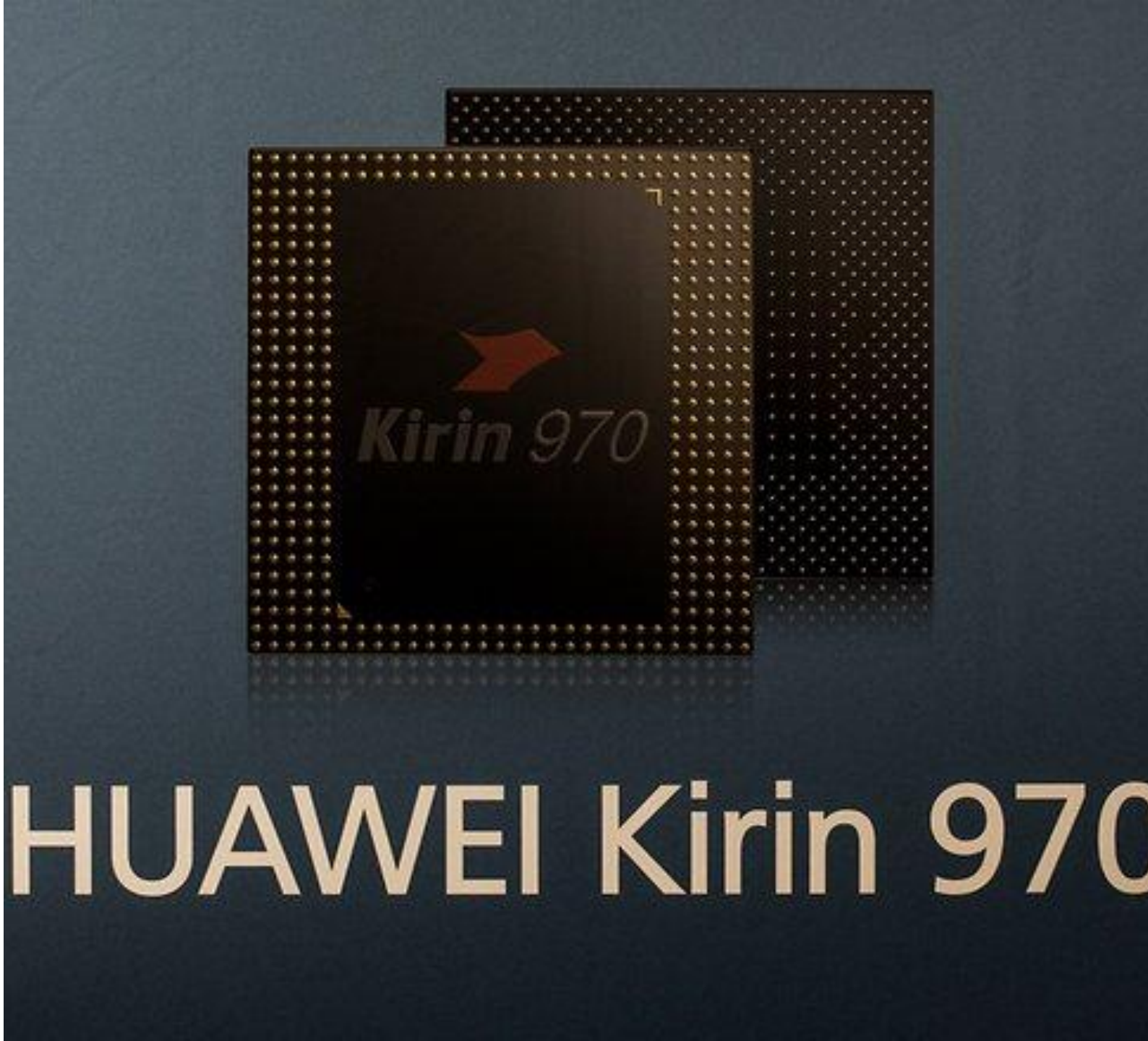
- The Apple A11 Bionic is a 64-bit ARM-based system on a chip (SoC), designed by Apple Inc. and manufactured by TSMC. It first appeared in the iPhone 8, iPhone 8 Plus, and iPhone X. The A11 includes dedicated neural network hardware that Apple calls a “Neural Engine”. This neural network hardware can perform up to 600 billion operations per second and is used for Face ID, Animoji and other machine learning tasks. The neural engine allows Apple to implement neural network and machine learning in a more energy-efficient manner than using either the main CPU or the GPU.

Google Tensor Processing Unit

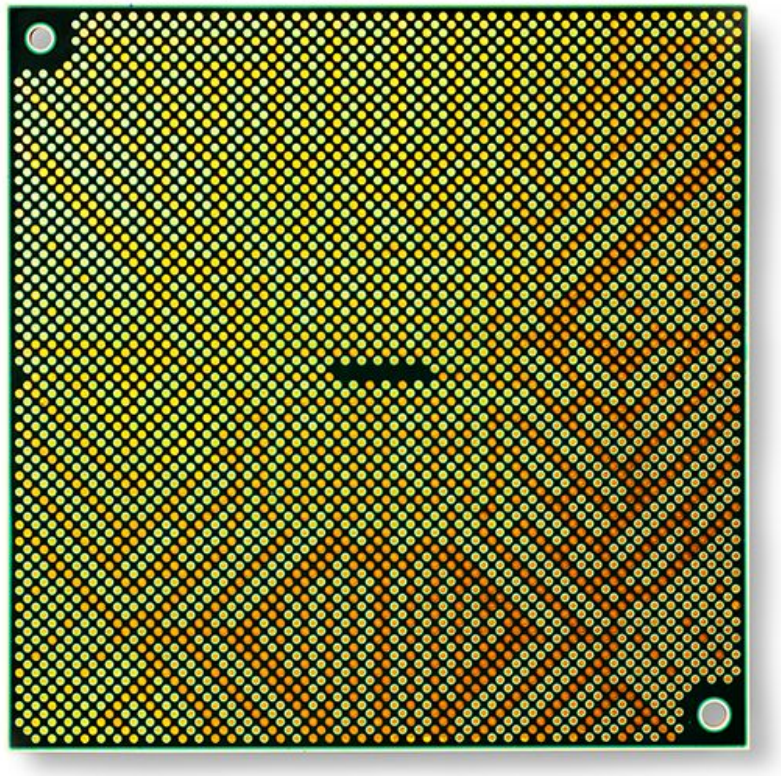


- A tensor processing unit (TPU) is an application-specific integrated circuit (ASIC) developed by Google specifically for machine learning. Compared to a graphics processing unit, it is designed for a high volume of low precision computation (e.g. as little as 8-bit precision) with higher IOPS per watt, and lacks hardware for rasterisation/texture mapping. The chip has been specifically designed for Google's TensorFlow framework. However, Google still uses CPUs and GPUs for other types of machine learning. Other AI accelerator designs are appearing from other vendors also and are aimed at embedded and robotics markets.

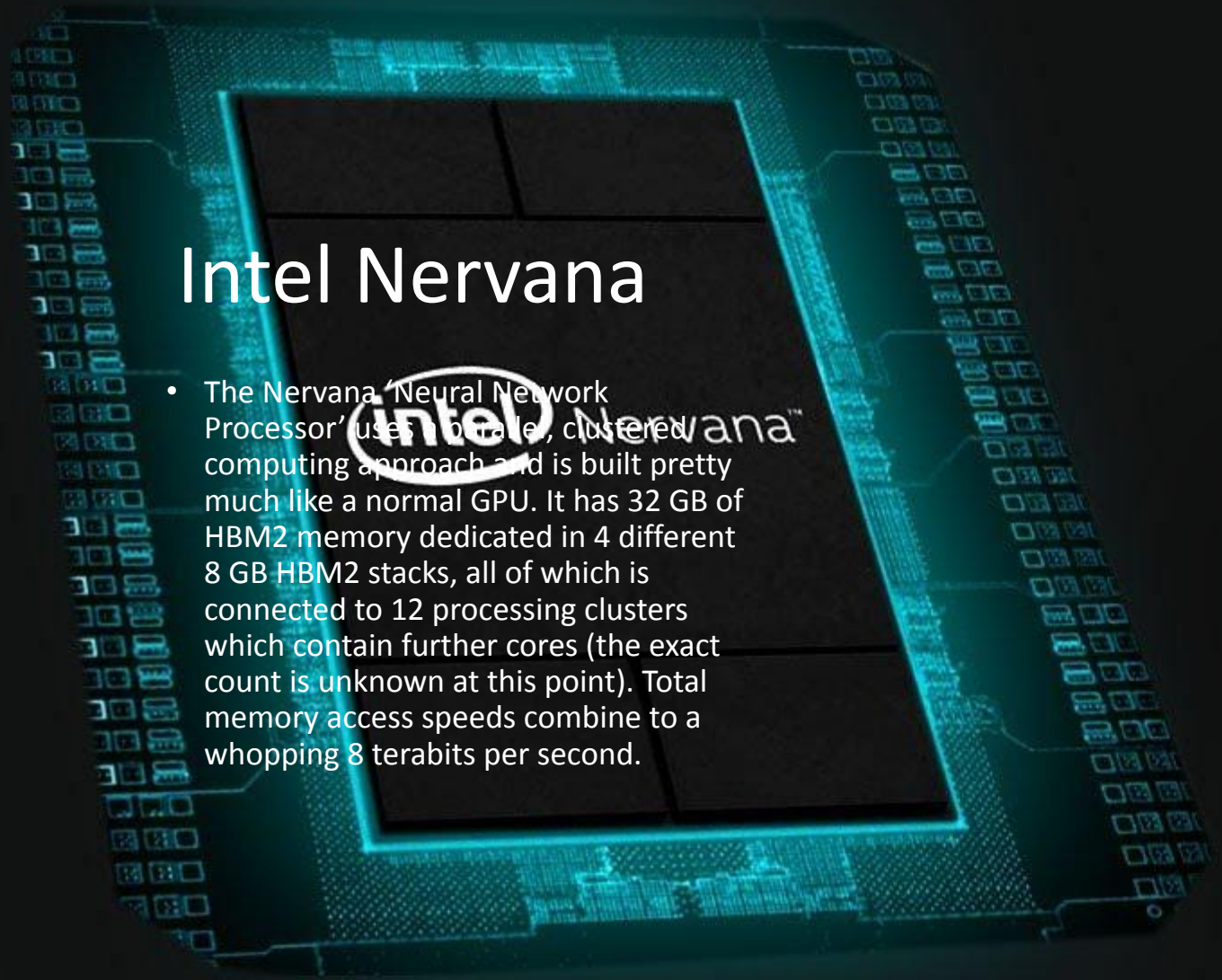
[Huawei Kirin 970]



IBM Power9



- Recently launched by IBM, Power9 is a chip which has a new systems architecture that is optimized for accelerators used in machine learning. Intel makes Xeon CPUs and Nervana accelerators and NVIDIA makes Tesla accelerators. IBM's Power9 is literally the Swiss Army knife of ML acceleration as it supports an astronomical amount of IO and bandwidth, 10X of anything that's out there today



Intel Nervana

- The Nervana ‘Neural Network Processor’ uses a parallel, clustered computing approach and is built pretty much like a normal GPU. It has 32 GB of HBM2 memory dedicated in 4 different 8 GB HBM2 stacks, all of which is connected to 12 processing clusters which contain further cores (the exact count is unknown at this point). Total memory access speeds combine to a whopping 8 terabits per second.

intel® Nervana™

ARTIFICIAL INTELLIGENCE

FUTURE

CHIP

Nvidia Tesla V100

- NVIDIA® Tesla® V100 is the world's most advanced data center GPU ever built to accelerate AI, HPC, and graphics. Powered by NVIDIA Volta™, the latest GPU architecture, Tesla V100 offers the performance of 100 CPUs in a single GPU—enabling data scientists, researchers, and engineers to tackle challenges that were once impossible.