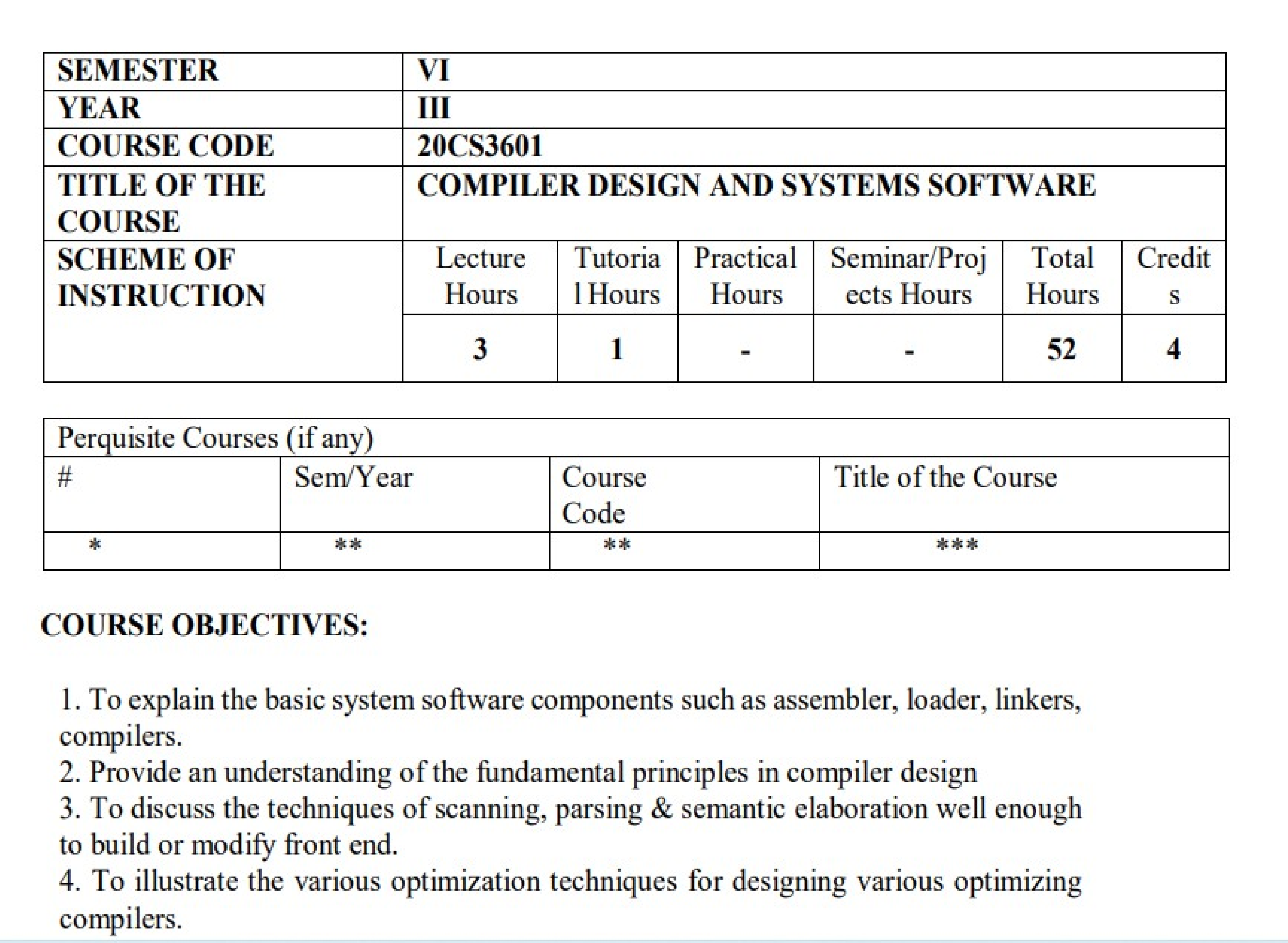
DAYANANDA SAGAR UNIVERSITY

- DEPARTMENT OF CSE

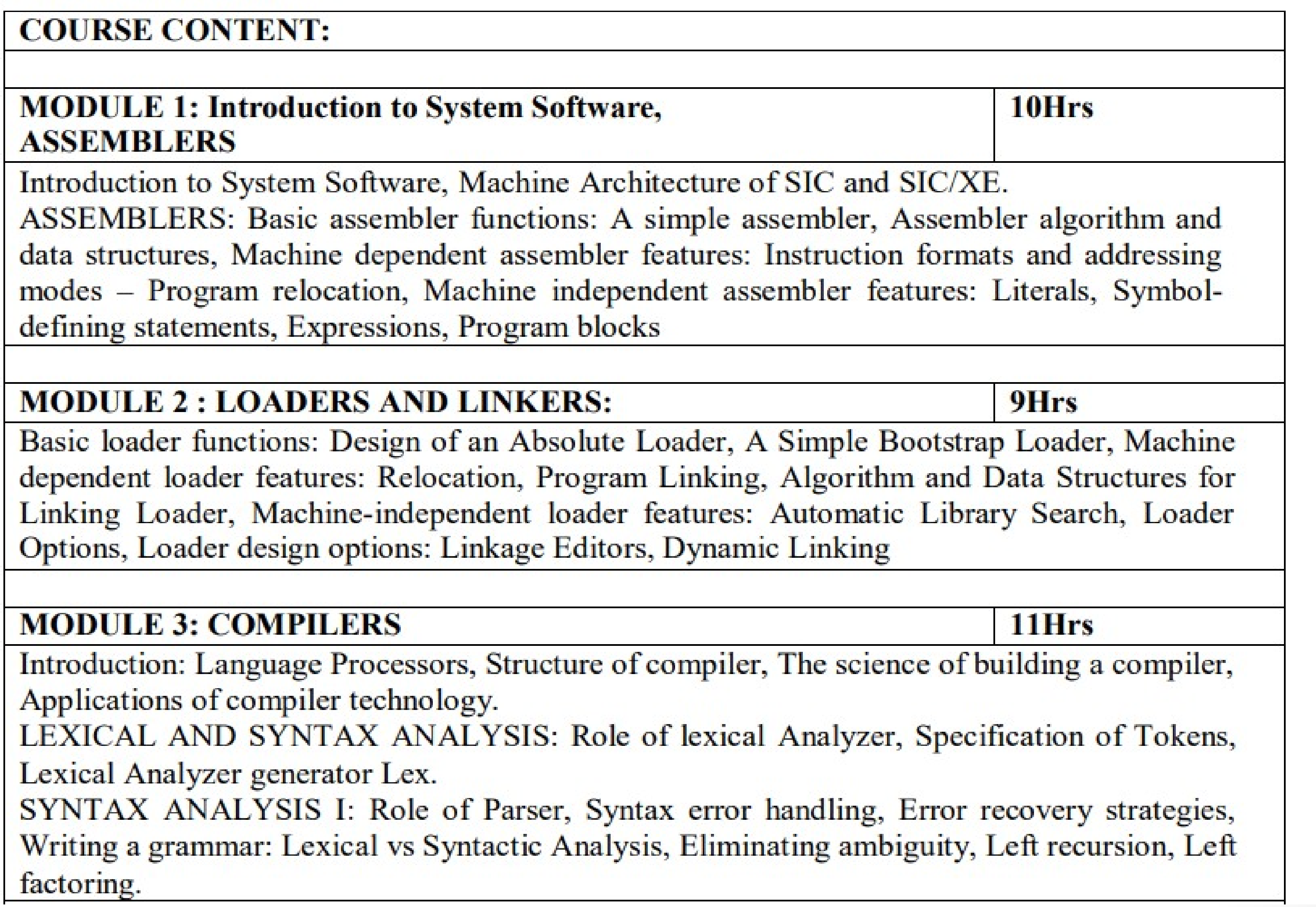


SEMESTER	**VI**					
YEAR	**III**					
COURSE CODE	**20CS3601**					
TITLE OF THE COURSE	**COMPILER DESIGN AND SYSTEMS SOFTWARE**					
SCHEME OF INSTRUCTION	Lecture Hours	Tutorial Hours	Practical Hours	Seminar/Projects Hours	Total Hours	Credits
	3	**1**	**-**	**-**	**52**	**4**

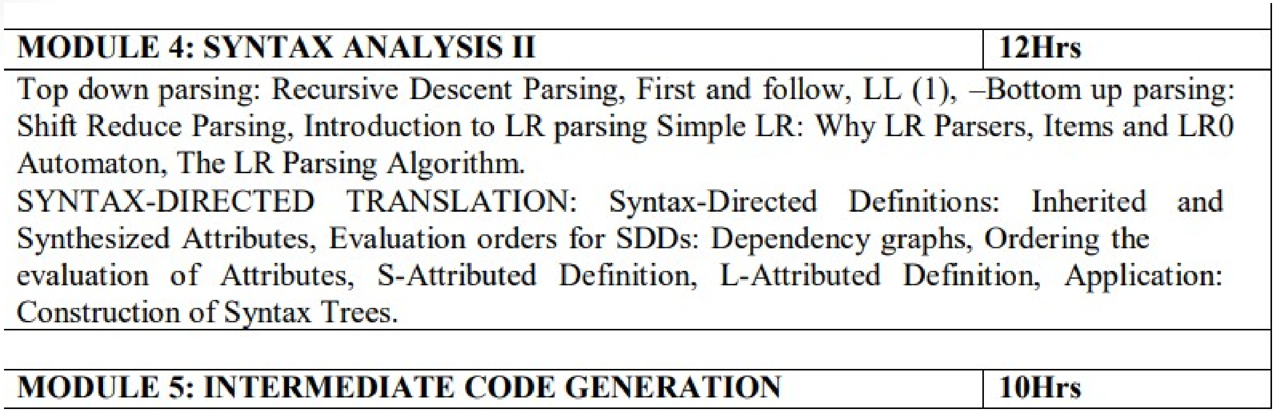
Perquisite Courses (if any)			
#	Sem/Year	Course Code	Title of the Course
*	**	**	***

COURSE OBJECTIVES:

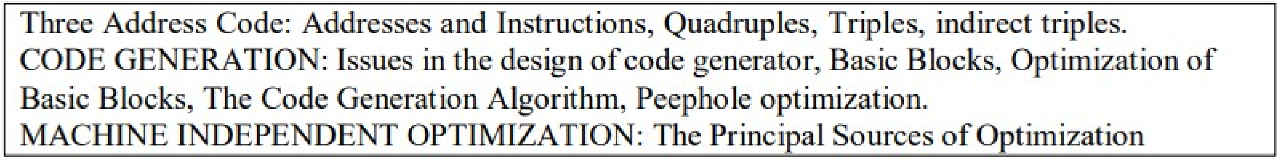
1. To explain the basic system software components such as assembler, loader, linkers, compilers.
2. Provide an understanding of the fundamental principles in compiler design
3. To discuss the techniques of scanning, parsing & semantic elaboration well enough to build or modify front end.
4. To illustrate the various optimization techniques for designing various optimizing compilers.



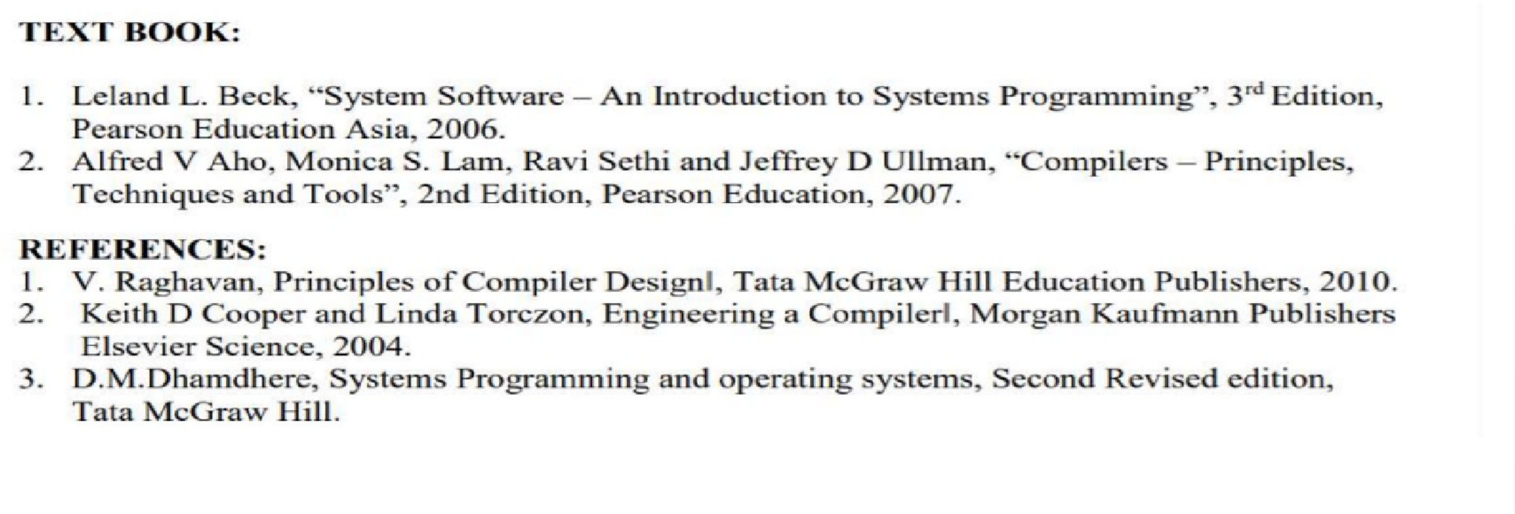
COURSE CONTENT:	
MODULE 1: Introduction to System Software, ASSEMBLERS	**10Hrs**
Introduction to System Software, Machine Architecture of SIC and SIC/XE. ASSEMBLERS: Basic assembler functions: A simple assembler, Assembler algorithm and data structures, Machine dependent assembler features: Instruction formats and addressing modes – Program relocation, Machine independent assembler features: Literals, Symbol-defining statements, Expressions, Program blocks	
MODULE 2 : LOADERS AND LINKERS:	**9Hrs**
Basic loader functions: Design of an Absolute Loader, A Simple Bootstrap Loader, Machine dependent loader features: Relocation, Program Linking, Algorithm and Data Structures for Linking Loader, Machine-independent loader features: Automatic Library Search, Loader Options, Loader design options: Linkage Editors, Dynamic Linking	
MODULE 3: COMPILERS	**11Hrs**
Introduction: Language Processors, Structure of compiler, The science of building a compiler, Applications of compiler technology. LEXICAL AND SYNTAX ANALYSIS: Role of lexical Analyzer, Specification of Tokens, Lexical Analyzer generator Lex. SYNTAX ANALYSIS I: Role of Parser, Syntax error handling, Error recovery strategies, Writing a grammar: Lexical vs Syntactic Analysis, Eliminating ambiguity, Left recursion, Left factoring.	



MODULE 4: SYNTAX ANALYSIS II	12Hrs
Top down parsing: Recursive Descent Parsing, First and follow, LL (1), –Bottom up parsing: Shift Reduce Parsing, Introduction to LR parsing Simple LR: Why LR Parsers, Items and LR0 Automaton, The LR Parsing Algorithm. SYNTAX-DIRECTED TRANSLATION: Syntax-Directed Definitions: Inherited and Synthesized Attributes, Evaluation orders for SDDs: Dependency graphs, Ordering the evaluation of Attributes, S-Attributed Definition, L-Attributed Definition, Application: Construction of Syntax Trees.	
MODULE 5: INTERMEDIATE CODE GENERATION	**10Hrs**



Three Address Code: Addresses and Instructions, Quadruples, Triples, indirect triples.
CODE GENERATION: Issues in the design of code generator, Basic Blocks, Optimization of Basic Blocks, The Code Generation Algorithm, Peephole optimization.
MACHINE INDEPENDENT OPTIMIZATION: The Principal Sources of Optimization



TEXT BOOK:

1. Leland L. Beck, "System Software – An Introduction to Systems Programming", 3rd Edition, Pearson Education Asia, 2006.
2. Alfred V Aho, Monica S. Lam, Ravi Sethi and Jeffrey D Ullman, "Compilers – Principles, Techniques and Tools", 2nd Edition, Pearson Education, 2007.

REFERENCES:

1. V. Raghavan, Principles of Compiler Designl, Tata McGraw Hill Education Publishers, 2010.
2. Keith D Cooper and Linda Torczon, Engineering a Compilerl, Morgan Kaufmann Publishers Elsevier Science, 2004.
3. D.M.Dhamdhere, Systems Programming and operating systems, Second Revised edition, Tata McGraw Hill.

Compiler

- The **compiler** is software that converts a program written in a high-level language (Source Language) to low-level language (Object/Target/Machine Language)..
- Some compilers
    - generate machine language
    - generate assembly language
    - more portable code such as C code,
- Some create abstract machine code.
- Some just generate data structures that are used by other parts of a program.
- That is the type of compiler that you will develop.

Why study compilers?

- It is useful for a computer scientist to study compiler design for several reasons.

1. Anyone who does any software development needs to use a compiler. It is a good idea to understand what is going on inside the tools that you use.
2. Studying compilers enables you to design and implement your own domain-specific language.
3. Compilers provide an essential interface between applications and architectures.
4. Compilers embody a wide range of theoretical techniques.
5. Compiler construction teaches programming and software engineering skills.

Why study compilers?

- It teaches you how real world applications are designed.

- It brings you closer to the language to exploit it. Compiler (a sophisticated program) bridges a gap between the language chosen & a computer architecture.

- Compiler improves software productivity by hiding low level details while delivering performance.

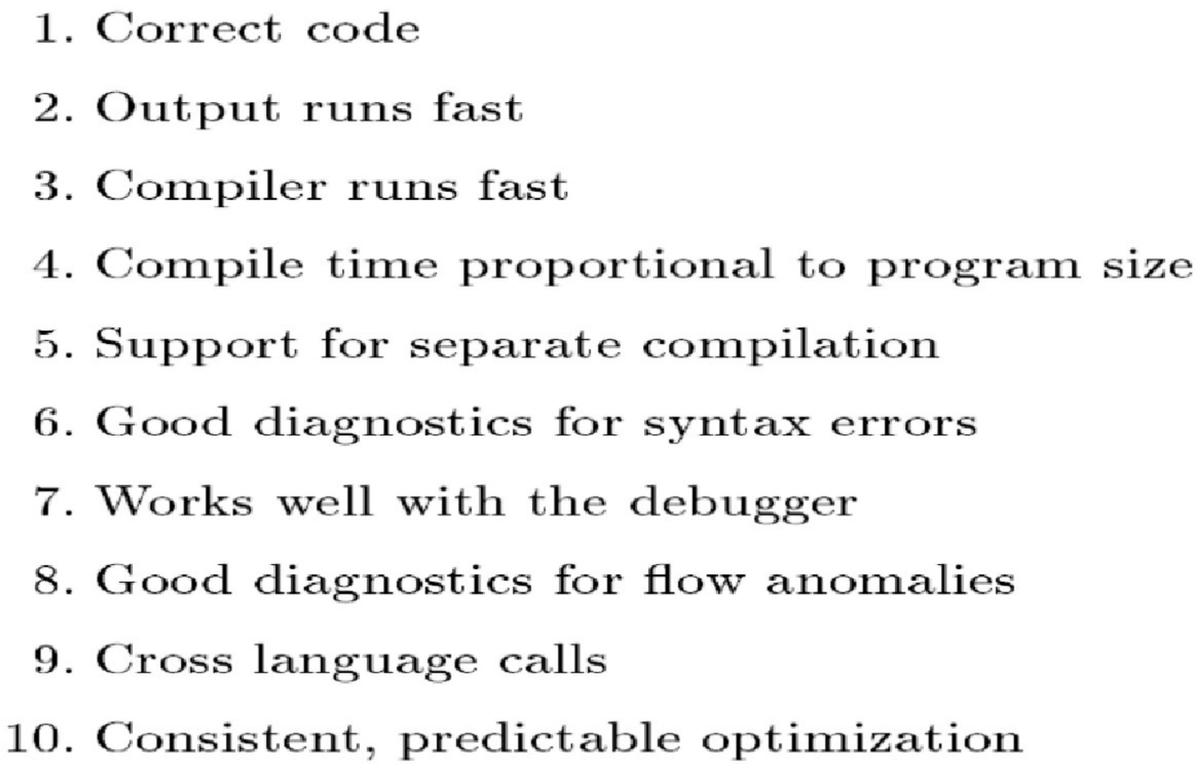
- Compiler provides techniques for developing other programming tools, like error detection tools.

- Program translation can be used to solve other problems, like Binary translation

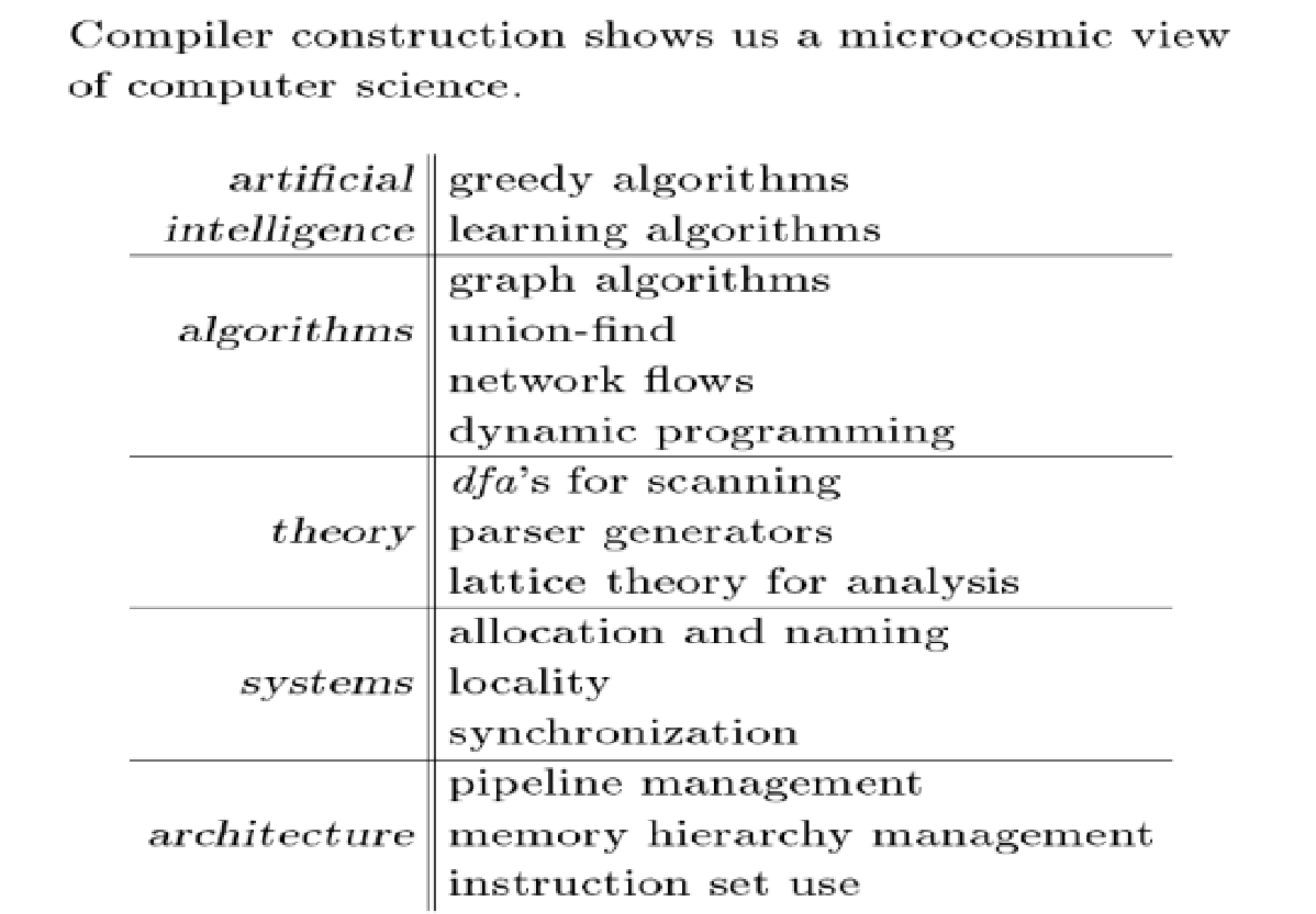
Machines have continued to change since they have been invented.

- Changes in Architecture□ Changes in Compilers
- New features present new problems
- Changing costs lead to different concerns
- Must re-engineer well known solution

Qualities in compiler



1. Correct code
2. Output runs fast
3. Compiler runs fast
4. Compile time proportional to program size
5. Support for separate compilation
6. Good diagnostics for syntax errors
7. Works well with the debugger
8. Good diagnostics for flow anomalies
9. Cross language calls
10. Consistent, predictable optimization



Compiler construction shows us a microcosmic view of computer science.

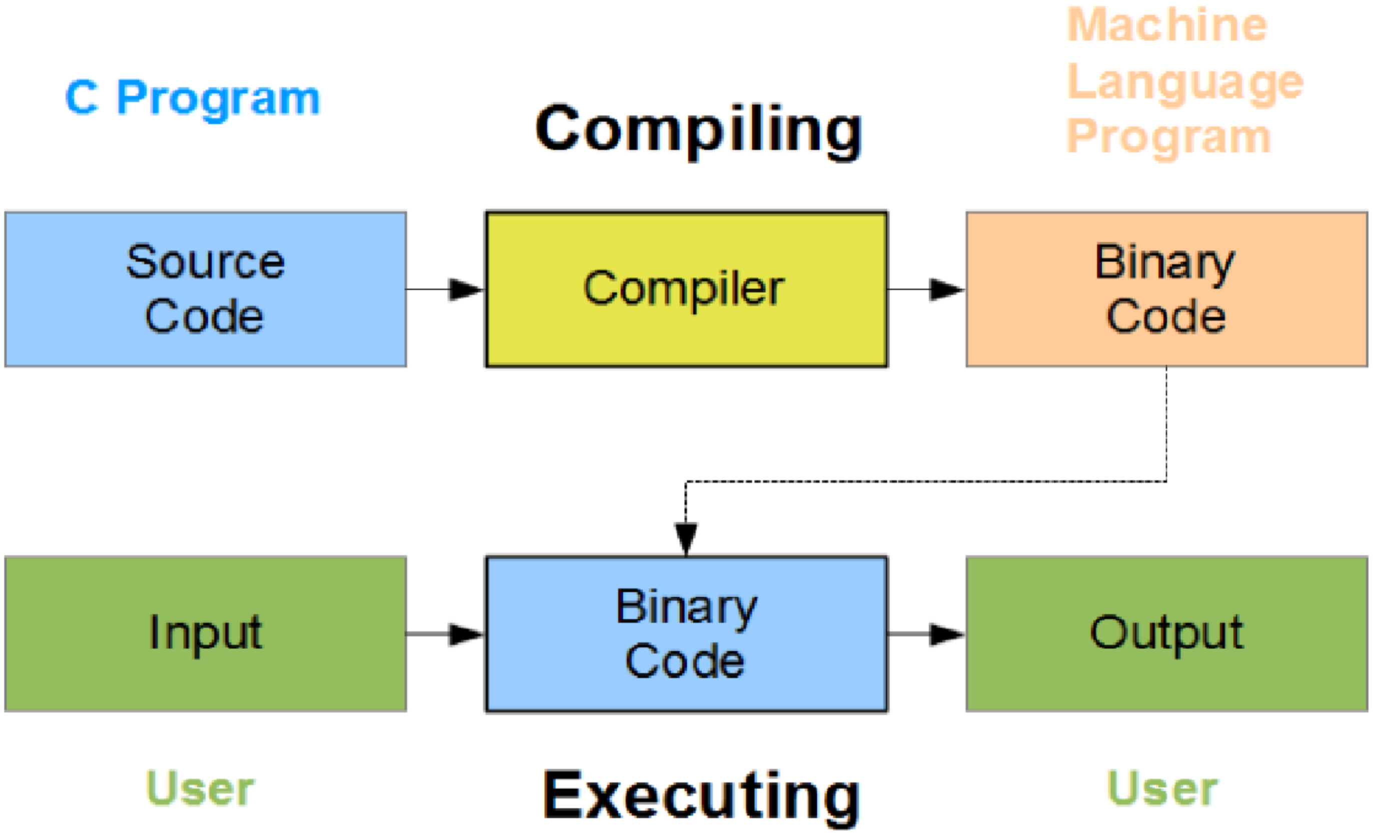
artificial intelligence	greedy algorithms learning algorithms
algorithms	graph algorithms union-find network flows dynamic programming
theory	*dfa*'s for scanning parser generators lattice theory for analysis
systems	allocation and naming locality synchronization
architecture	pipeline management memory hierarchy management instruction set use

Program Execution

- • Execution of a program written in HLL is basically a 2-step process

1.The source program is compiled first i.e. translated into the object program.

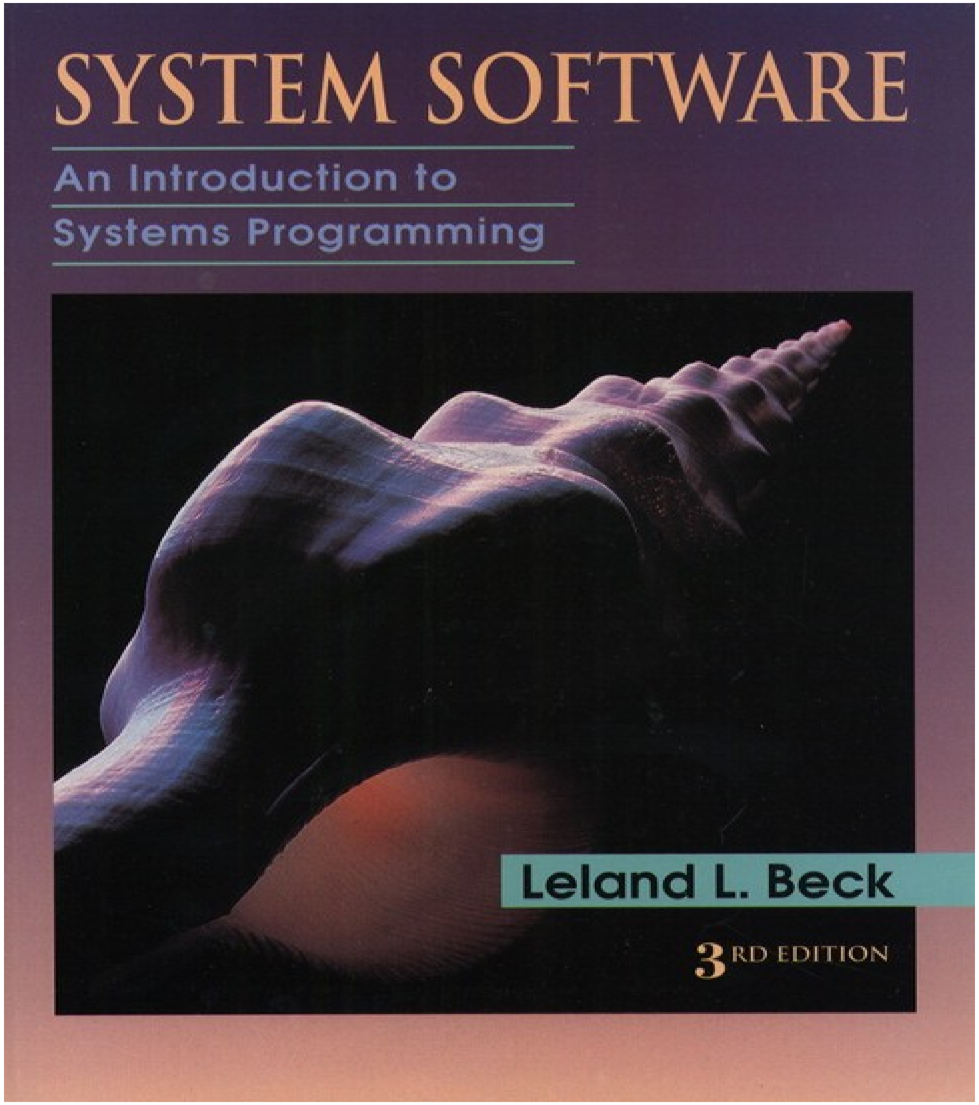
2. The resulting object program is loaded into memory and executed


C Program
Compiling
Machine Language Program
Source Code
Compiler
Binary Code
Input
Binary Code
Output
User
Executing
User

System Software:

An Introduction to Systems Programming

Book



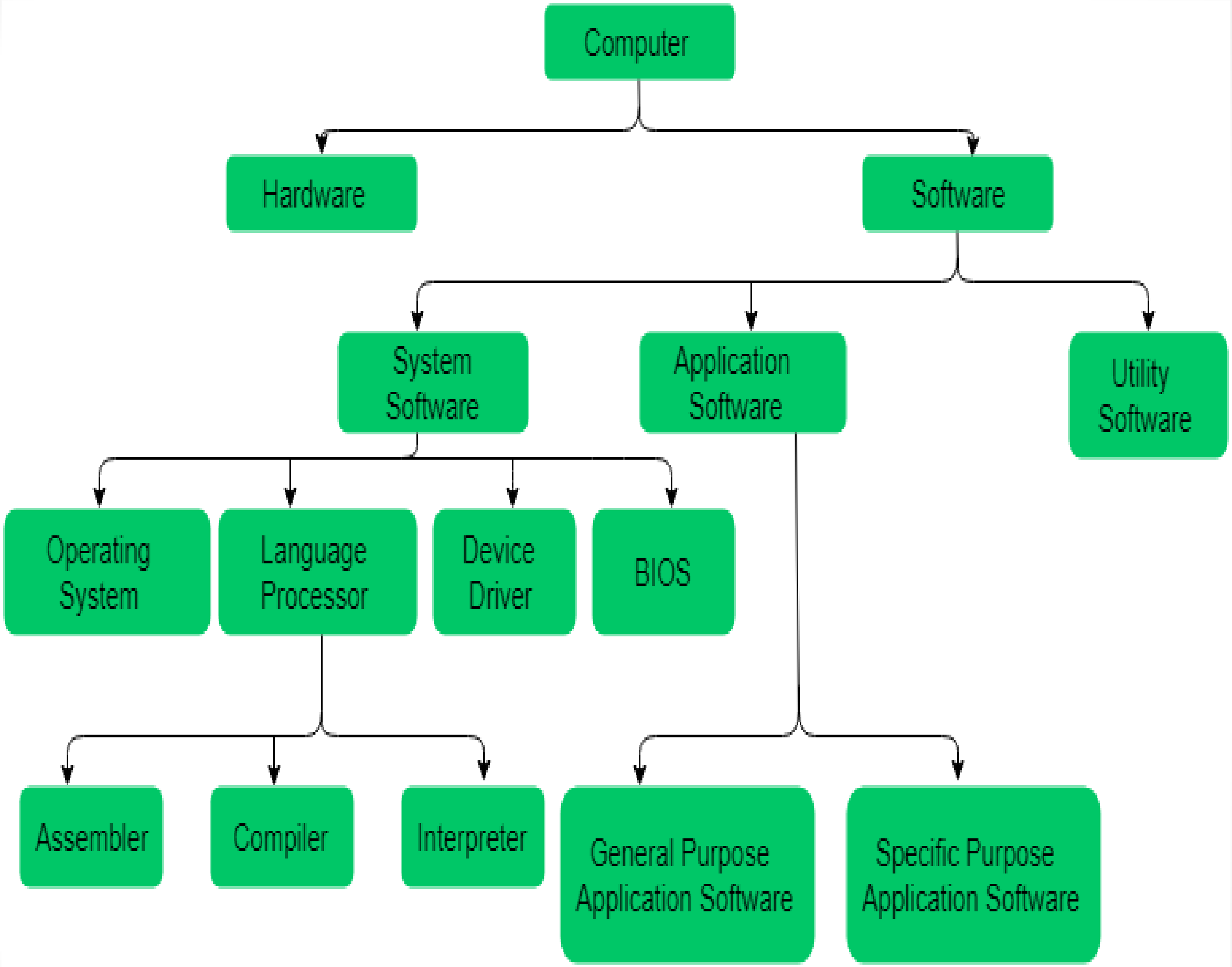
SYSTEM SOFTWARE

An Introduction to Systems Programming

Leland L. Beck

3RD EDITION

Chapter 1
Background



Computer

Hardware

Software

System Software

Application Software

Utility Software

Operating System

Language Processor

Device Driver

BIOS

Assembler

Compiler

Interpreter

General Purpose Application Software

Specific Purpose Application Software

Outline

- Introduction
- System Software and Machine Architecture
- The Simplified Instructional Computer (SIC)
  - SIC Machine Architecture
  - SIC/XE Machine Architecture
  - SIC Programming Examples

1.1 Introduction

- System Software consists of a variety of programs that support the operation of a computer.
- The programs implemented in either software and (or) firmware that makes the computer hardware usable.
- The software makes it possible for the users to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.
- BIOS (Basic Input Output System).

1.2 System Software and Machine Architecture

- System Software vs Application Software
  - One characteristic in which most system software differs from application software is machine dependency.
  - System programs are intended to support the operation and use of the computer itself, rather than any particular application.

- Examples of system software
  - Text editor, assembler, compiler, loader or linker, debugger, macro processors, operating system.

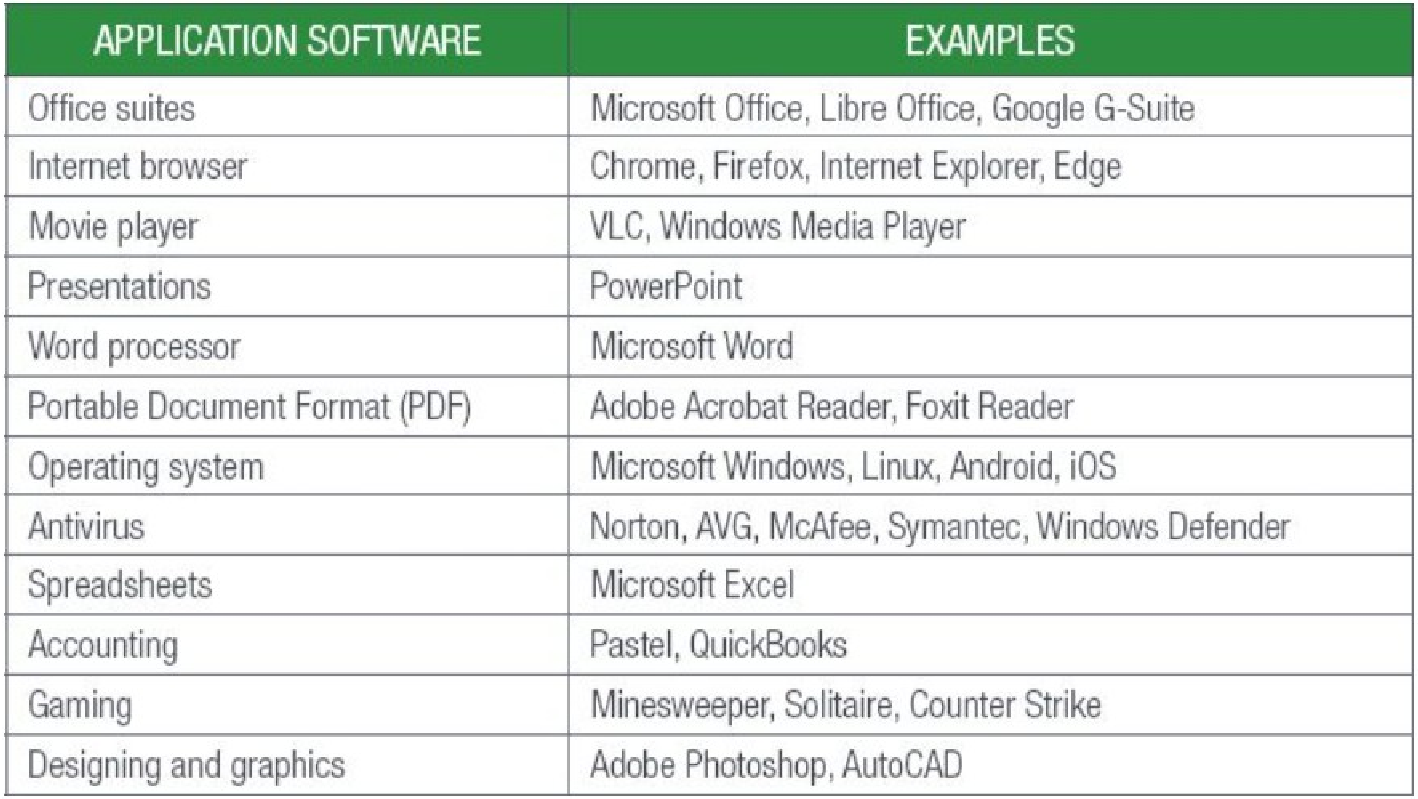
1.2 System Software and Machine Architecture

- **Text editor**
  - To create and modify the program
- **Compiler and assembler**
  - You translated these programs into machine language
- **Loader or linker**
  - The resulting machine program was loaded into memory and prepared for execution
- **Debugger**
  - To help detect errors in the program.

Macro processor

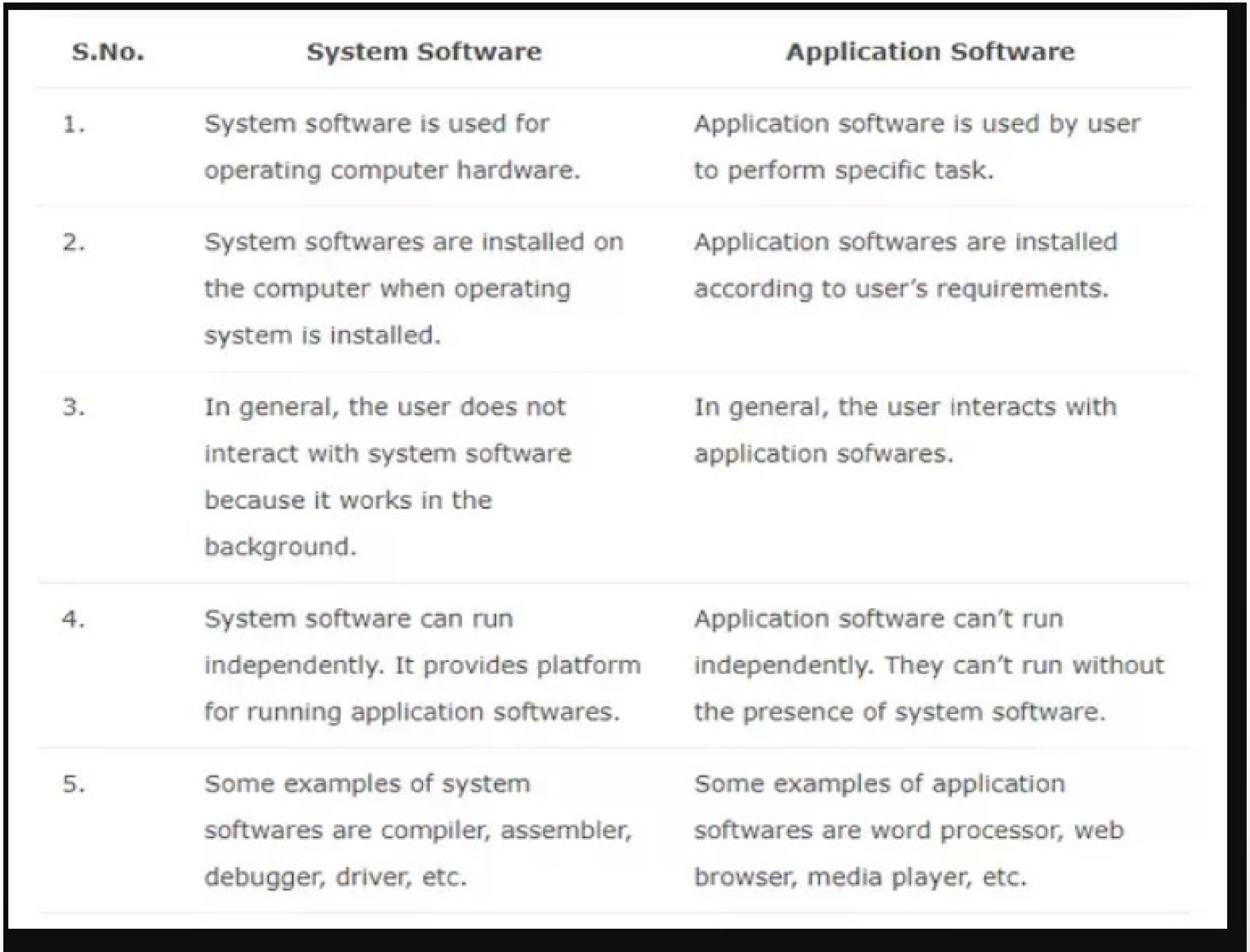
- translate macros instructions into its definition

Application Software Examples.



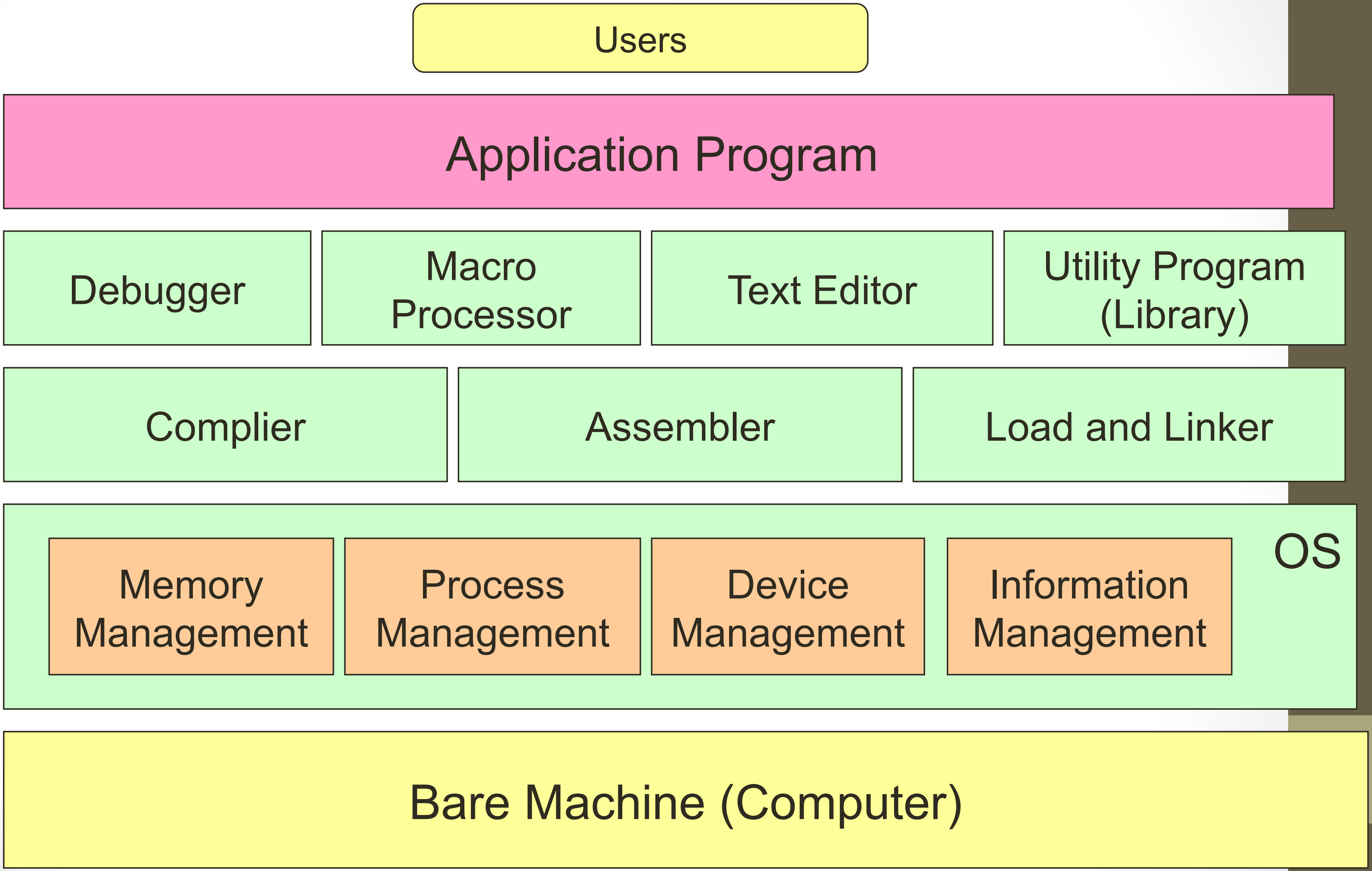
APPLICATION SOFTWARE	EXAMPLES
Office suites	Microsoft Office, Libre Office, Google G-Suite
Internet browser	Chrome, Firefox, Internet Explorer, Edge
Movie player	VLC, Windows Media Player
Presentations	PowerPoint
Word processor	Microsoft Word
Portable Document Format (PDF)	Adobe Acrobat Reader, Foxit Reader
Operating system	Microsoft Windows, Linux, Android, iOS
Antivirus	Norton, AVG, McAfee, Symantec, Windows Defender
Spreadsheets	Microsoft Excel
Accounting	Pastel, QuickBooks
Gaming	Minesweeper, Solitaire, Counter Strike
Designing and graphics	Adobe Photoshop, AutoCAD

System Software vs Application Software



S.No.	System Software	Application Software
1.	System software is used for operating computer hardware.	Application software is used by user to perform specific task.
2.	System softwares are installed on the computer when operating system is installed.	Application softwares are installed according to user's requirements.
3.	In general, the user does not interact with system software because it works in the background.	In general, the user interacts with application sofwares.
4.	System software can run independently. It provides platform for running application softwares.	Application software can't run independently. They can't run without the presence of system software.
5.	Some examples of system softwares are compiler, assembler, debugger, driver, etc.	Some examples of application softwares are word processor, web browser, media player, etc.

System Software Concept



Users

Application Program

Debugger

Macro Processor

Text Editor

Utility Program (Library)

Complier

Assembler

Load and Linker

OS

Memory Management

Process Management

Device Management

Information Management

Bare Machine (Computer)

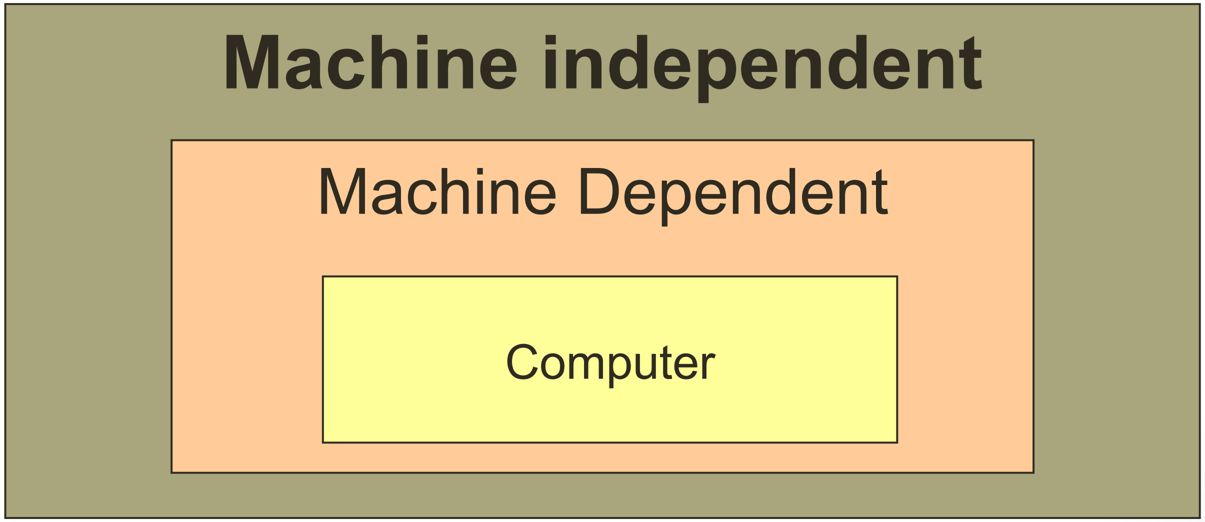
System Software and Machine Architecture

- **Machine dependent**

Instruction Set, Instruction Format, Addressing Mode, Assembly language …

- **Machine independent**

General design logic/strategy, literals, symbol-defining statements, program blocks, control sections and program linking…



1.3 The Simplified Instructional Computer

- SIC refers to Simplified Instruction Computer which is a hypothetical computer that has been designed to include the hardware features most often found on real machines, while avoiding unusual and irrelevant complexities.
- This allows to clearly separate the central concepts of a system software from the implementation details associated with a particular machine.
- Like many other products, SIC comes in two versions
  - The standard model
  - An XE version “extra equipment's”
  - The two versions has been designed to be upward compatible

1.SIC (Simplified Instructional Computer)

2.SIC/XE (Extra Equipment)

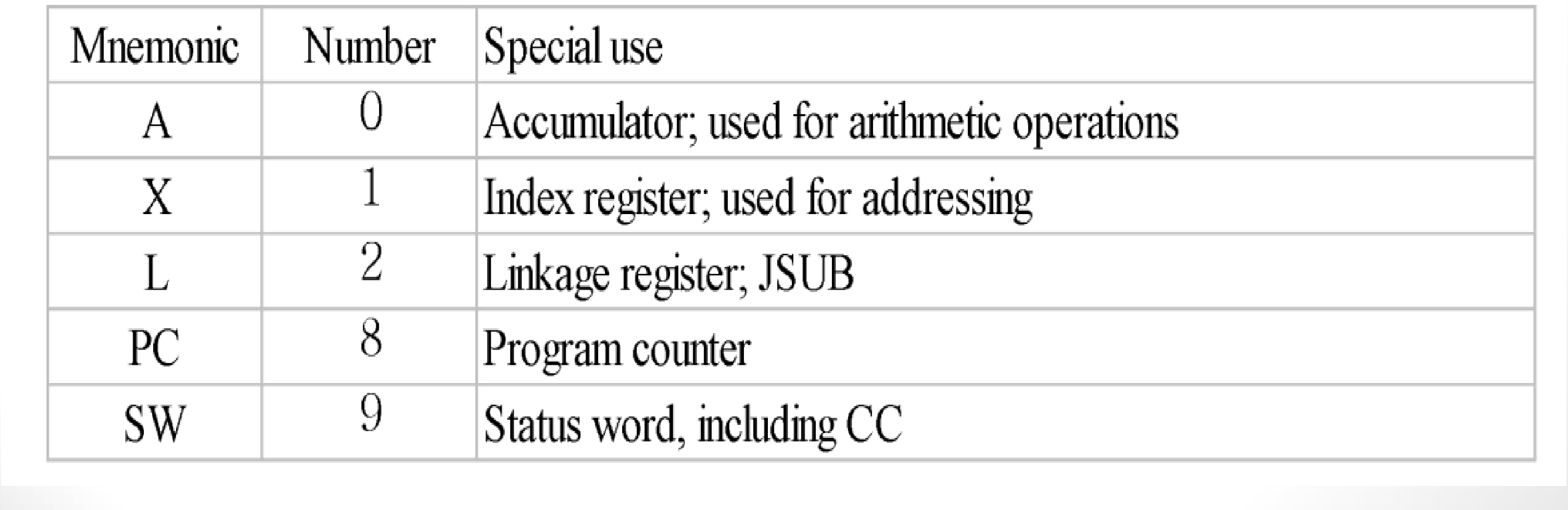
SIC Machine Architecture

1.Memory

- All addresses are byte addresses with words associated by the location of their Lowest numbered byte.
- Consist of 8-bit bytes.
- 3 consecutive bytes form a word (24 bits)
- Total size of the memory is 32768(2^15) bytes

2.Registers

There are 5 special purpose registers each of which are 24 bits in



Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; JSUB
PC	8	Program counter
SW	9	Status word, including CC

SIC Machine Architecture

3.Data Formats

- Integers are stored as 24-bit binary number
- 2’s complement representation for negative values
- Characters are stored using 8-bit ASCII codes
- No floating-point hardware on the standard version of SIC.

4. Instruction format

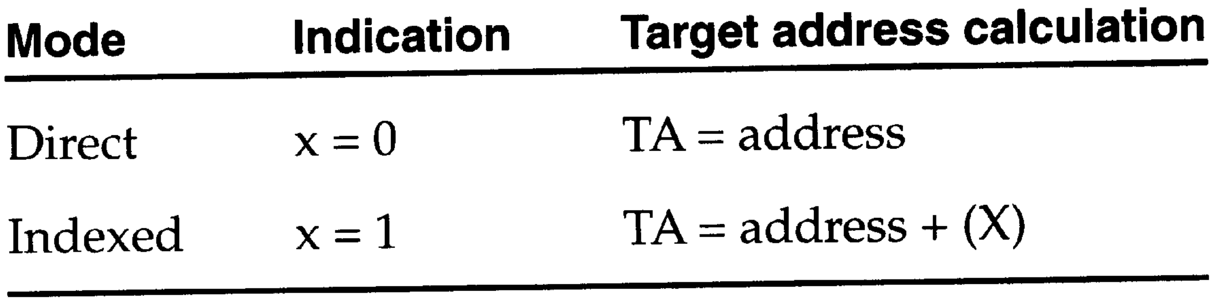
- 24-bit format
- The flag bit x is used to indicate indexed-addressing mode

8	1	15
opcode	x	address

SIC Machine Architecture

5.Addressing Modes

- There are two addressing modes available
  - Indicated by x bit in the instruction
  - (X) represents the contents of reg. X



Mode	Indication	Target address calculation
Direct	x = 0	TA = address
Indexed	x = 1	TA = address + (X)

SIC Machine Architecture

6. Instruction set

a. Integer arithmetic operations: ADD, SUB, MUL, DIV, etc.

All arithmetic operations involve register A and a word in memory, with the result being left in the register

b. comparison: COMP

COMP compares the value in register A with a word in memory, this instruction sets a condition code CC to indicate the result

c. conditional jump instructions: JLT, JEQ, JGT

These instructions test the setting of CC and jump accordingly.

d. subroutine linkage: JSUB, RSUB

- **JSUB** jumps to the subroutine, placing the return address in register L
- **RSUB** returns by jumping to the address contained in register L.

e. Load and store registers

Instructions to store and load registers are LDA, LDX, STA, STX.

SIC Machine Architecture

7. I/O operations

- I/O are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A.
- Each device is assigned a unique 8-bit code as an operand.
- There are 3 I/O instructions

1.The **Test Device (TD)** instruction tests whether the addressed device is ready to send or receive a byte of data.(CC is set to <,if it is ready and set to =, if its not ready).

2. **Read Data (RD),** reads the data from an input device, if it is ready to send the data.

3. **Write Data (WD)** writes the data onto an output device, if its ready to receive the data

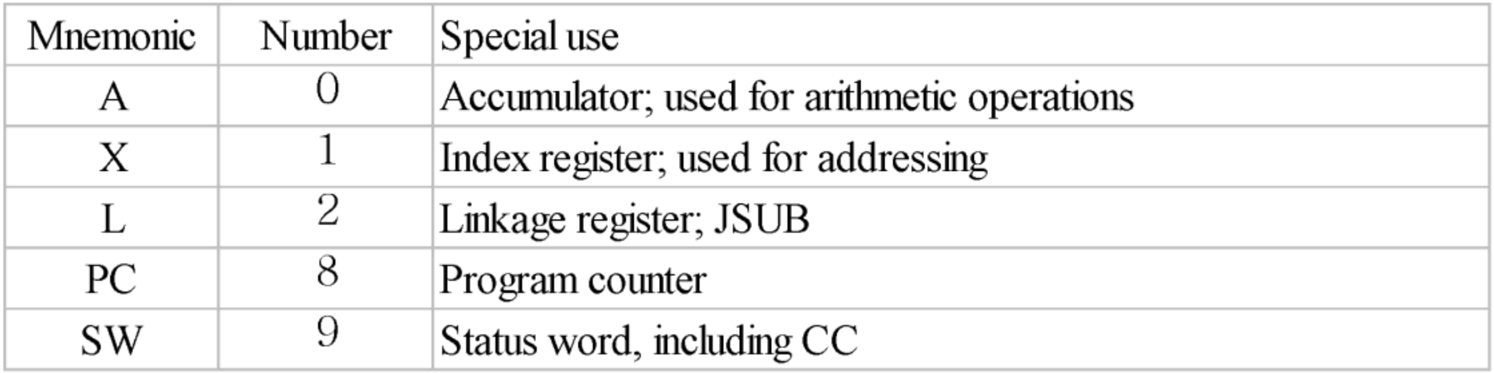
SIC/XE Machine Architecture

1.Memory:

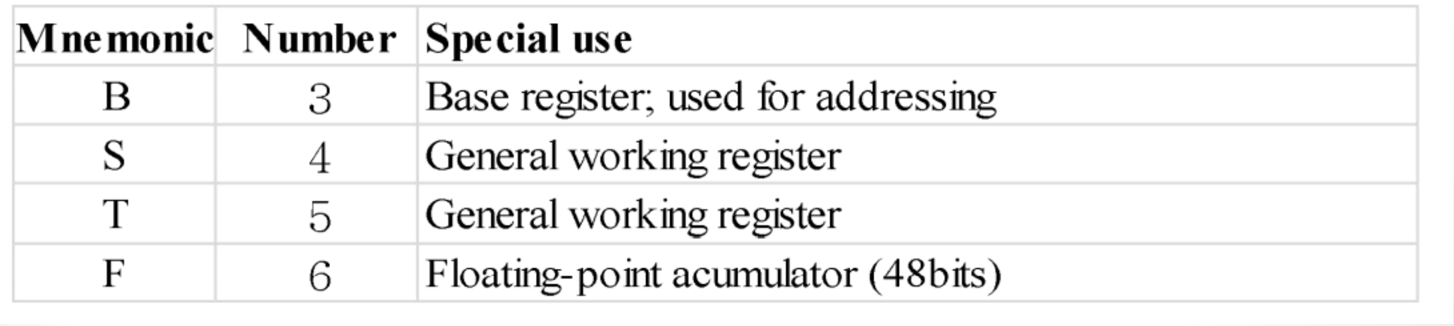
Structure is same as that of the SIC standard version.

The maximum memory available is **1MegaByte(2^20) bytes.**

2. Registers In addition to the registers of SIC standard version, the following registers are provided in SIC/XE.



Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; JSUB
PC	8	Program counter
SW	9	Status word, including CC



Mnemonic	**Number**	**Special use**
B	3	Base register; used for addressing
S	4	General working register
T	5	General working register
F	6	Floating-point acumulator (48bits)

SIC/XE Machine Architecture

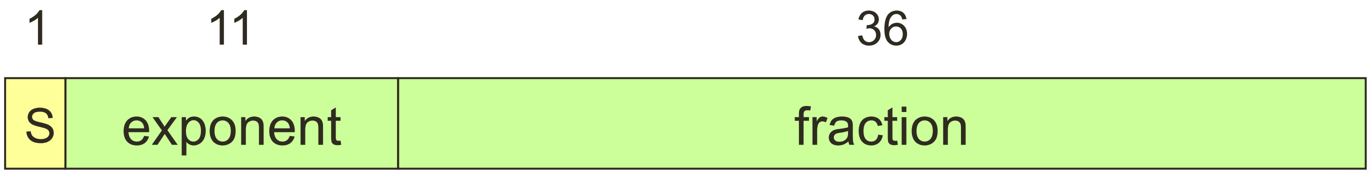
7. Input/Output:

- In addition to the I/O instructions provided in SIC Standard version, SIC/XE provides I/O channels used to perform input and output while CPU is executing other instructions.
- **SIO, TIO, HIO**: start, test, halt the operation of I/O channels respectively

SIC/XE Machine Architecture

3.Data format

- 24-bit binary number for integer, 2’s complement for negative values
- 48-bit floating-point data type
- The exponent is between 0 and 2047
- The absolute value of the number is represented by $f*2^{(e-1024)}$
- The sign of floating number is represented by
- 0: for +ve
- 1 for –ve

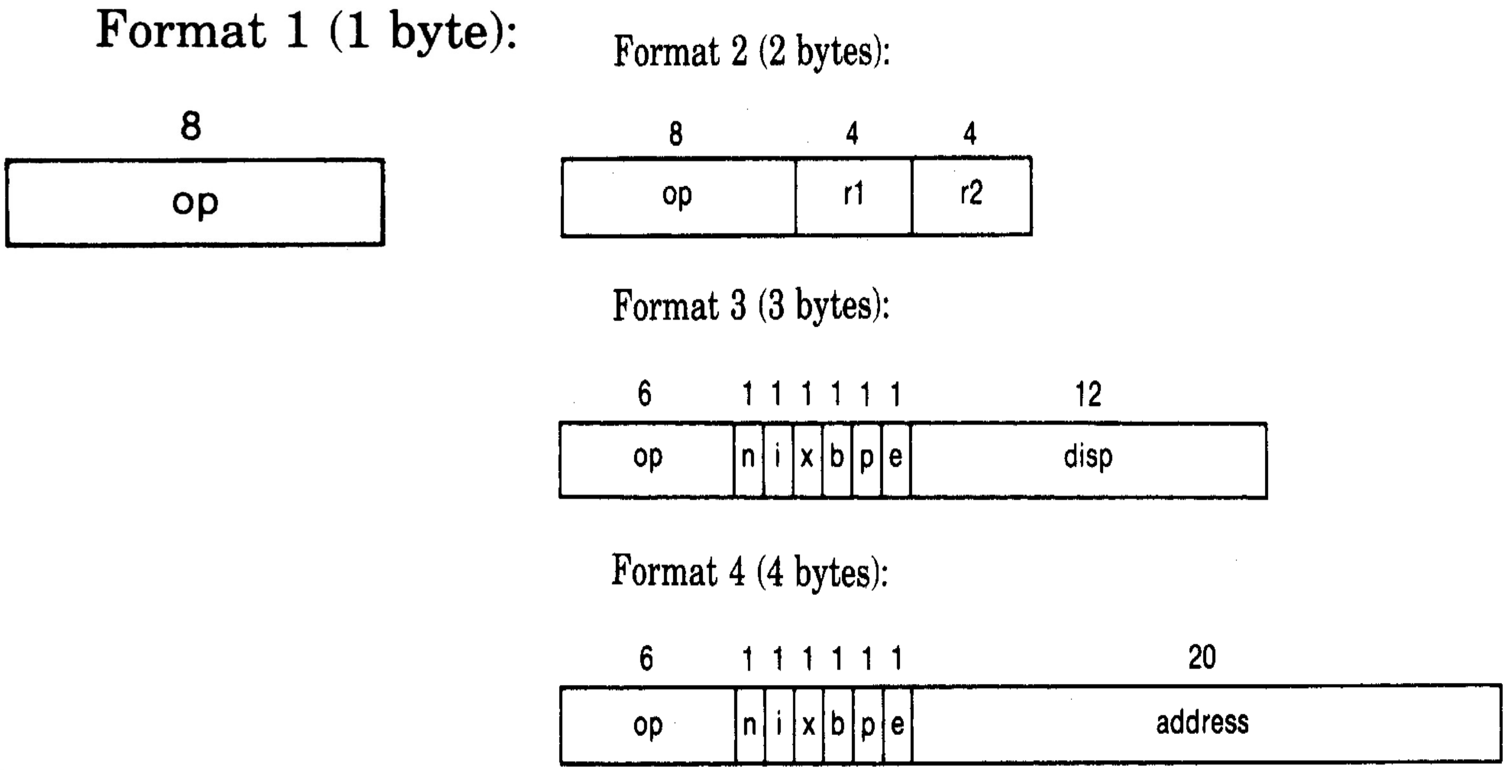

1 11 36

s exponent fraction

SIC/XE Machine Architecture

4.Instruction formats

- Relative addressing - format 3 (e=0)
- Extend the address to 20 bits - format 4 (e=1)
- Don’t refer memory at all - formats 1 and 2



Format 1 (1 byte):

8
op

Format 2 (2 bytes):

8	4	4
op	r1	r2

Format 3 (3 bytes):

6	1	1	1	1	1	1	12
op	n	i	x	b	p	e	disp

Format 4 (4 bytes):

6	1	1	1	1	1	1	20
op	n	i	x	b	p	e	address

SIC/XE Machine Architecture

5.Addressing modes

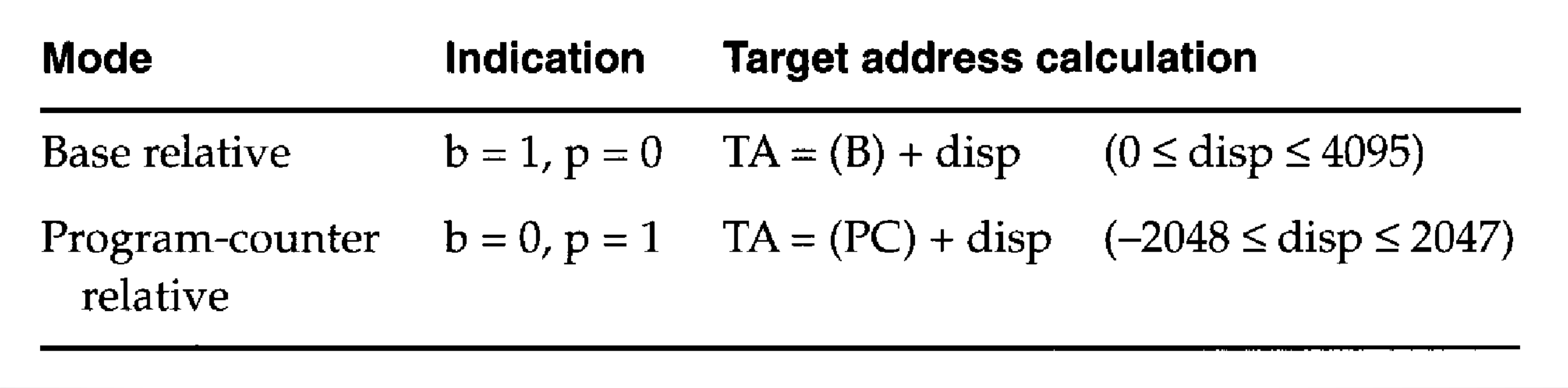
n i x b p e

1. (n, i) Simple n=0, i=0 or n=1, i=1
2. (i) Immediate n=0, i=1 **TA=Value**
3. (n) Indirect n=1, i=0 **TA=(Operand)**
4. Base relative b=1, p=0 **TA=(B)+disp**

   0 <= disp <= 4095

5.PC relative b=0, p=1 **TA=(PC)+disp**

   -2048 <= disp <= 2047



Mode	Indication	Target address calculation	
Base relative	b = 1, p = 0	TA = (B) + disp	(0 ≤ disp ≤ 4095)
Program-counter relative	b = 0, p = 1	TA = (PC) + disp	(–2048 ≤ disp ≤ 2047)

SIC/XE Machine Architecture

6.Instruction Set

In addition to the instruction set, provided in SIC Standard version, SIC/XE provides following instructions.

- new registers: LDB, STB, etc.
- **floating-point arithmetic:** ADDF, SUBF, MULF, DIVF
- **register move:** RMO
- **register-register arithmetic:** ADDR, SUBR, MULR, DIVR
- **supervisor call**: SVC generates an interrupt for OS.

SIC/XE Machine Architecture

1.Base relative addressing:

bit b=1and p=0 and disp field is interpreted as a 12 bit unsigned integer in format 3.

2.Program counter relative addressing:

bit b=0and p=1 and disp field is interpreted as a 12 bit signed integer, with negative values represented in 2’s complement notation.

3.Direct addressing : The displacement and address fields will be taken as the target address respectively in format 3 and format 4, if the bits b and p are both set to 0.

4.Indexed addressing:

If the bit x is set to 1, the content of register X is also added in the target address calculation.

SIC/XE Machine Architecture

5.Immediate addressing :

if the bits i=1 and n=0, the target address itself is the operand value and no memory is referenced.

6.Indirect addressing:

if the bits i=0, and n=1, the word at the location given by the target address contains the address of the operand value.

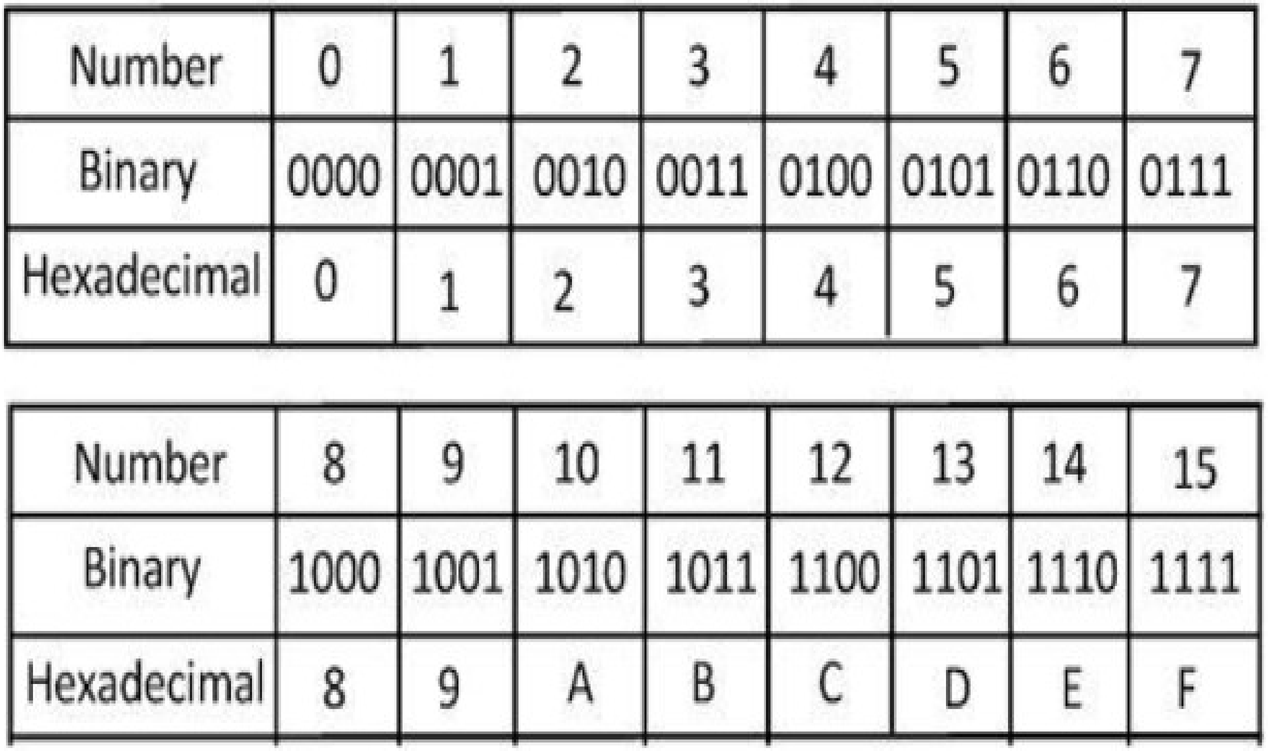
7. Simple addressing: if the bits i =n=0 or i=n=1, the target address is taken as the location of the operand.

Calculate the target address generated for the following machine instruction:

- **I. 032600 h**
- **II. 03C300 h**
- **III. 022030 h**
- **IV. 010030 h**
- **V. 003600 h**
- **VI. 0310C303 h**

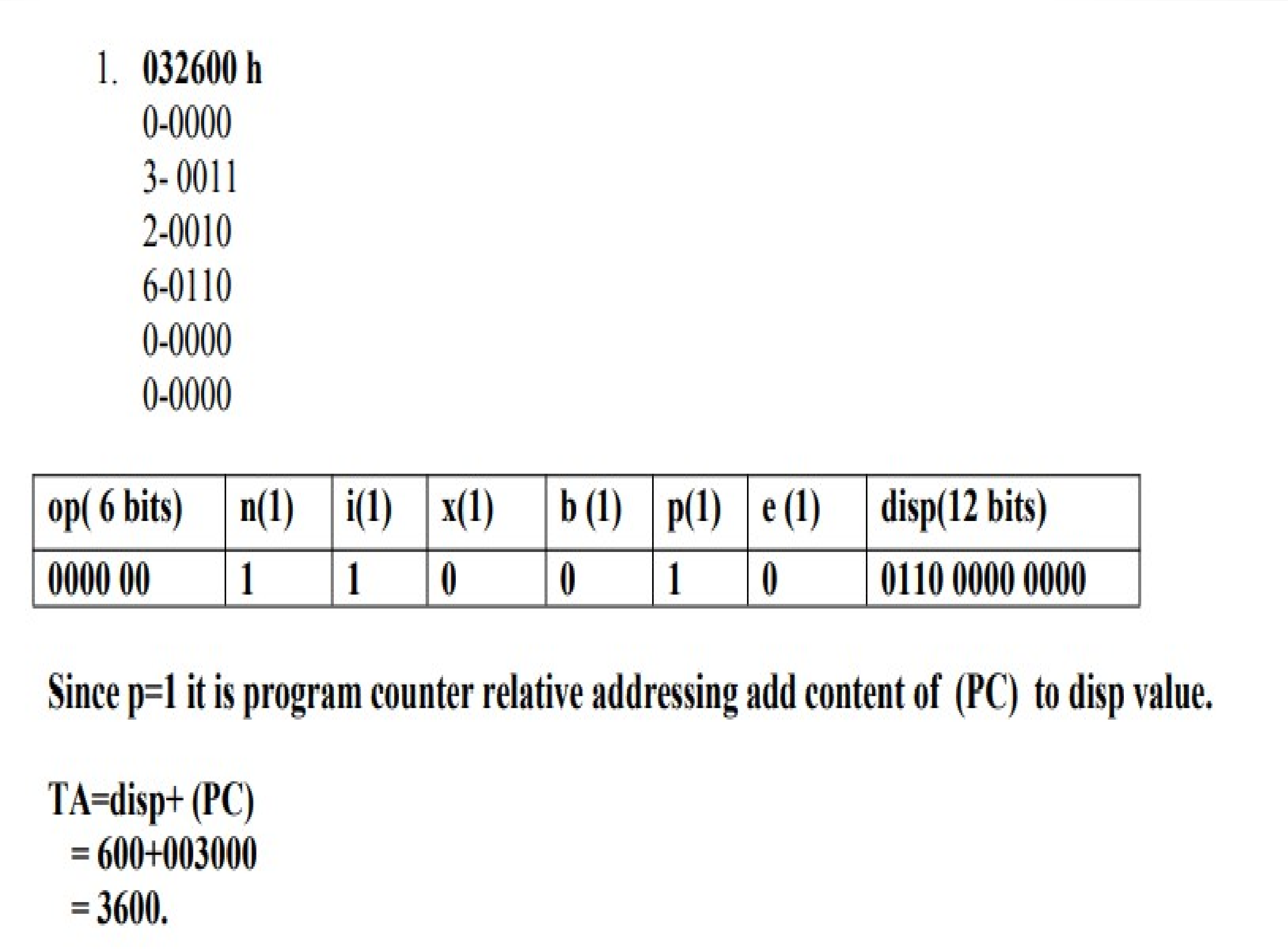
Given

- **(B)= 006000, (PC)=003000, (X)=000090**



Number	0	1	2	3	4	5	6	7
Binary	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal	0	1	2	3	4	5	6	7

Number	8	9	10	11	12	13	14	15
Binary	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal	8	9	A	B	C	D	E	F



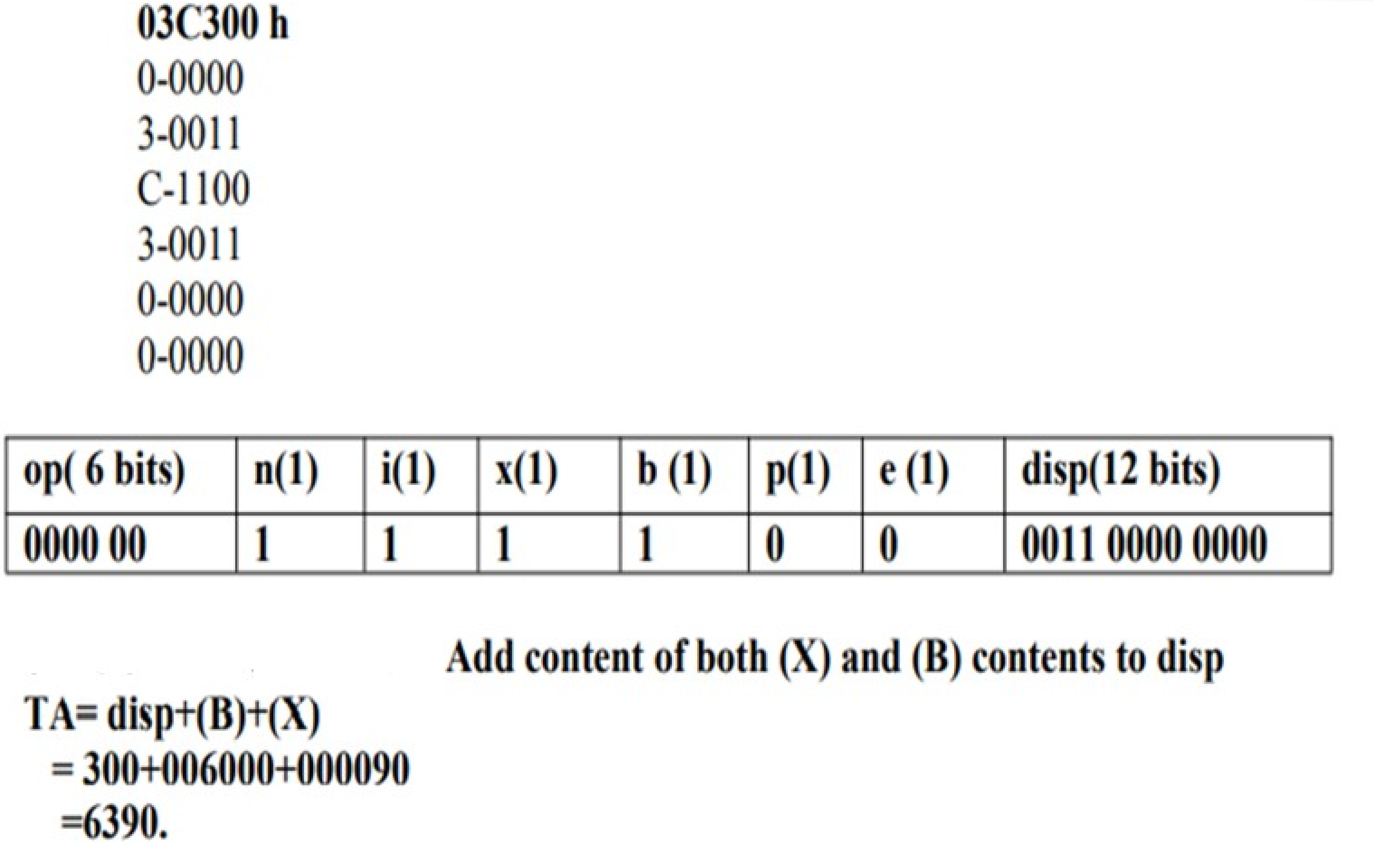
1. **032600 h**
   0-0000
   3- 0011
   2-0010
   6-0110
   0-0000
   0-0000

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	1	1	0	0	1	0	0110 0000 0000

Since p=1 it is program counter relative addressing add content of (PC) to disp value.

TA=disp+ (PC)
= 600+003000
= 3600.

2.



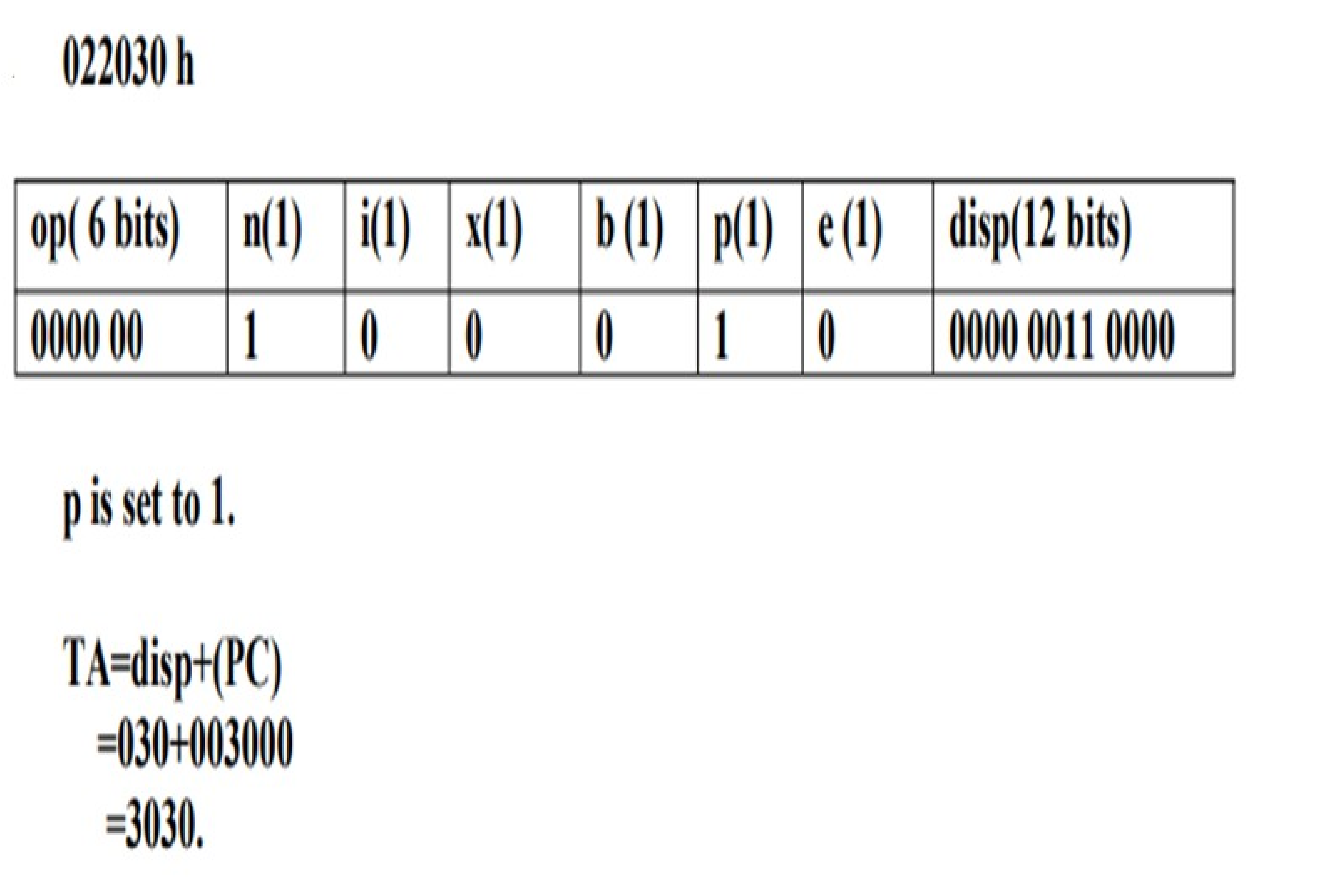
03C300 h
0-0000
3-0011
C-1100
3-0011
0-0000
0-0000

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	1	1	1	1	0	0	0011 0000 0000

Add content of both (X) and (B) contents to disp

TA= disp+(B)+(X)
= 300+006000+000090
=6390.

3.



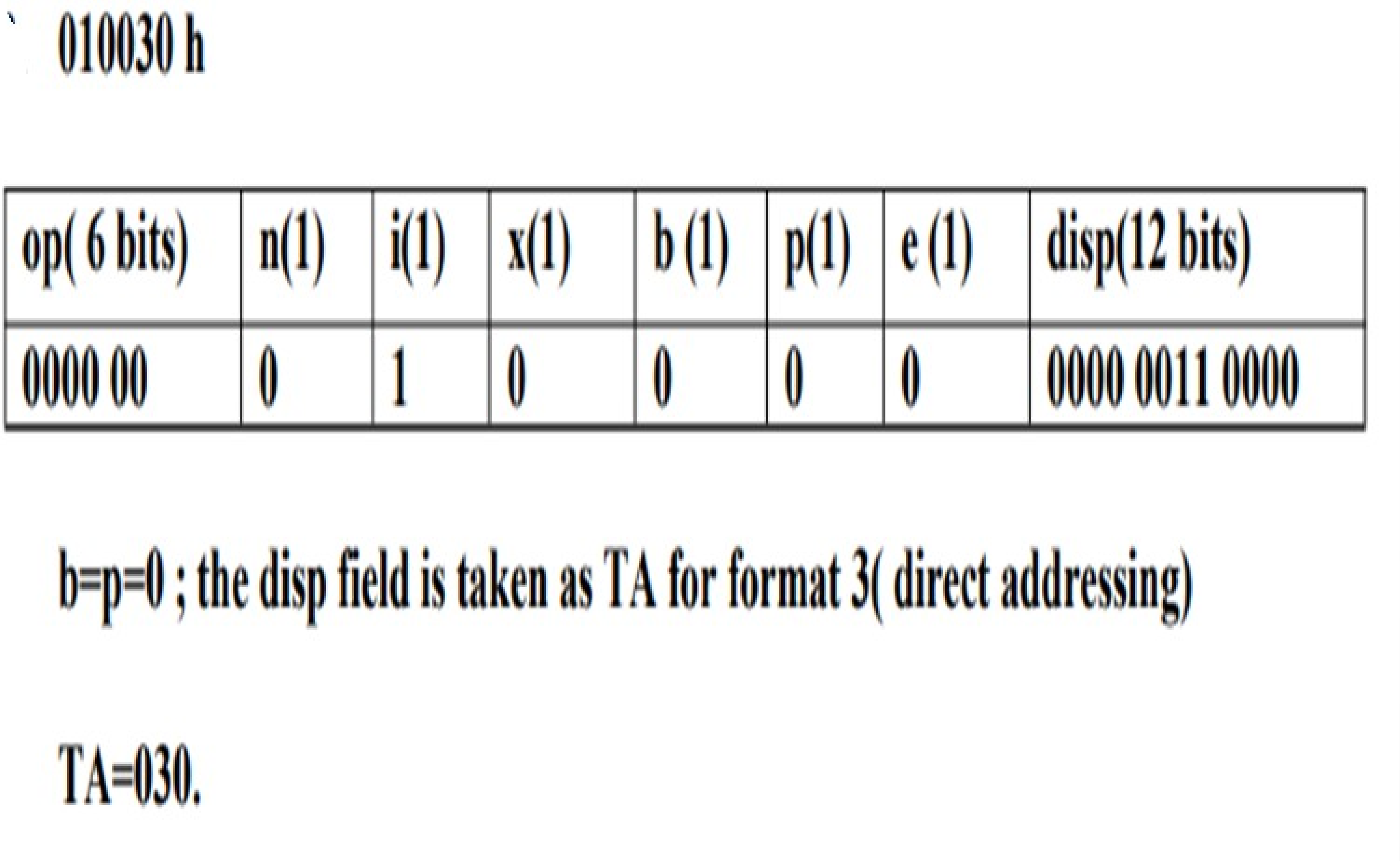
022030 h

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	1	0	0	0	1	0	0000 0011 0000

p is set to 1.

TA=disp+(PC)
=030+003000
=3030.

- 4.



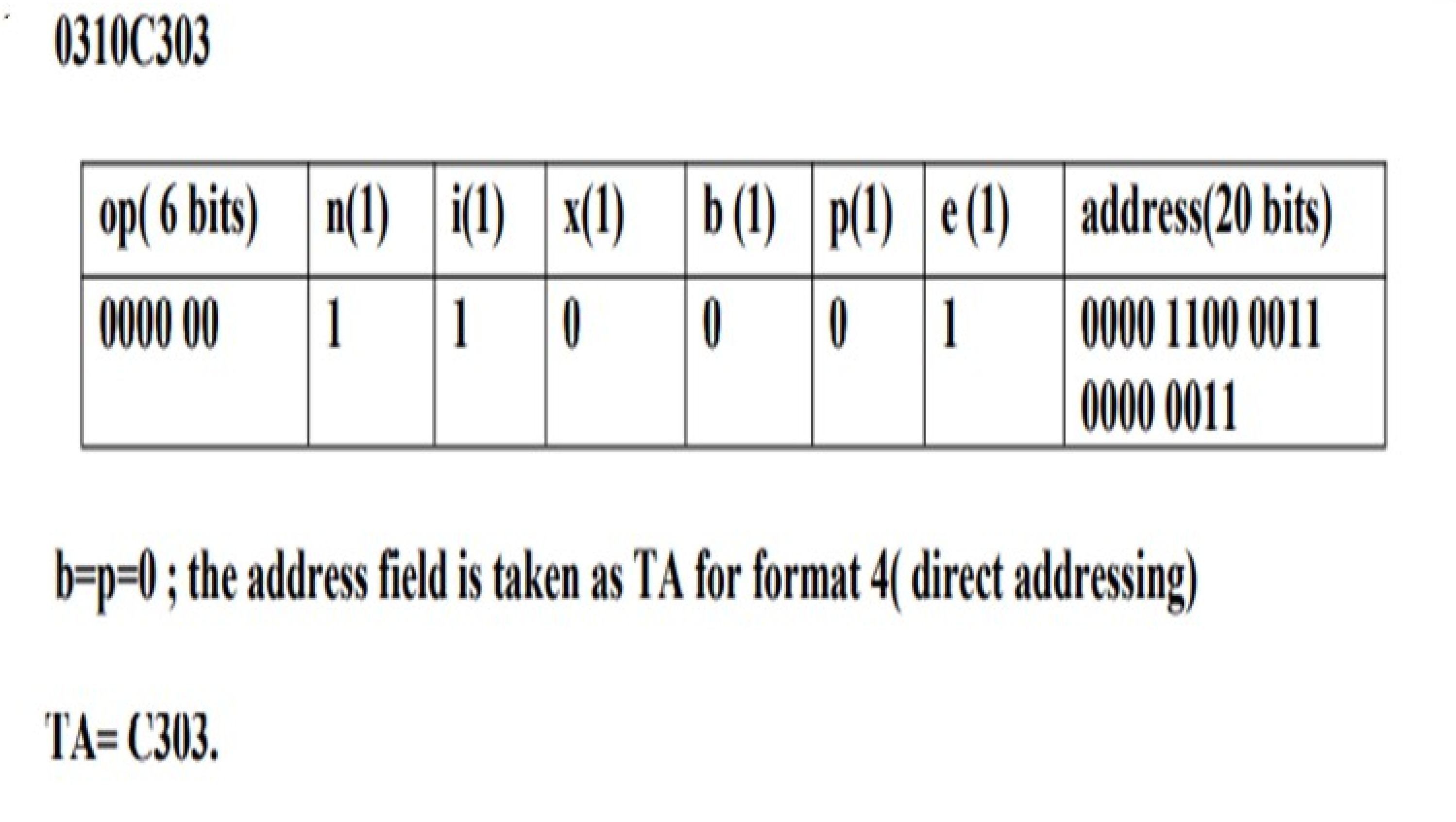
010030 h

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	0	1	0	0	0	0	0000 0011 0000

b=p=0 ; the disp field is taken as TA for format 3(direct addressing)

TA=030.

6.



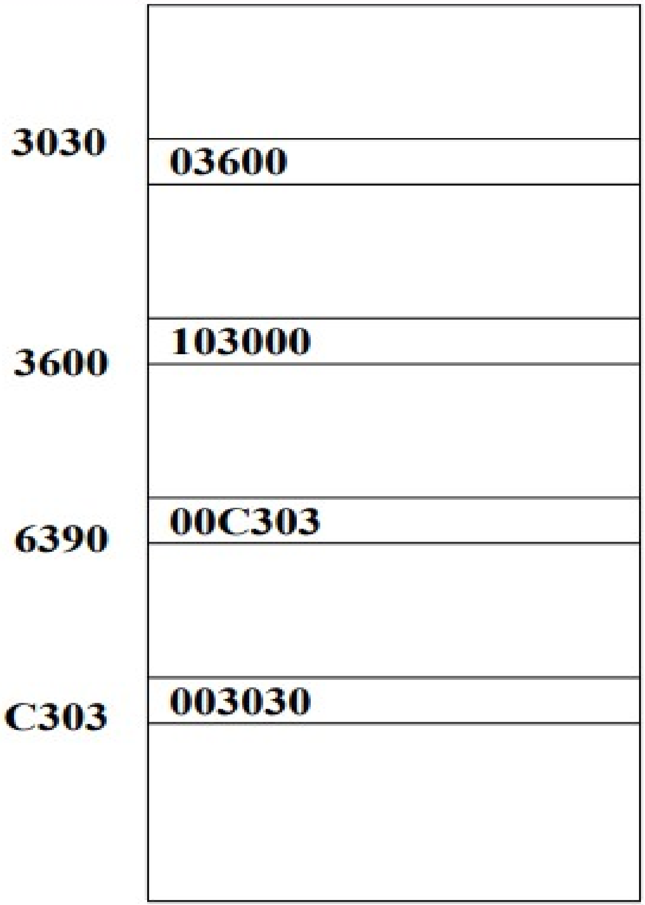
0310C303

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	address(20 bits)
0000 00	1	1	0	0	0	1	0000 1100 0011 0000 0011

b=p=0 ; the address field is taken as TA for format 4(direct addressing)

TA= C303.

Calculate the value Loaded into Register A

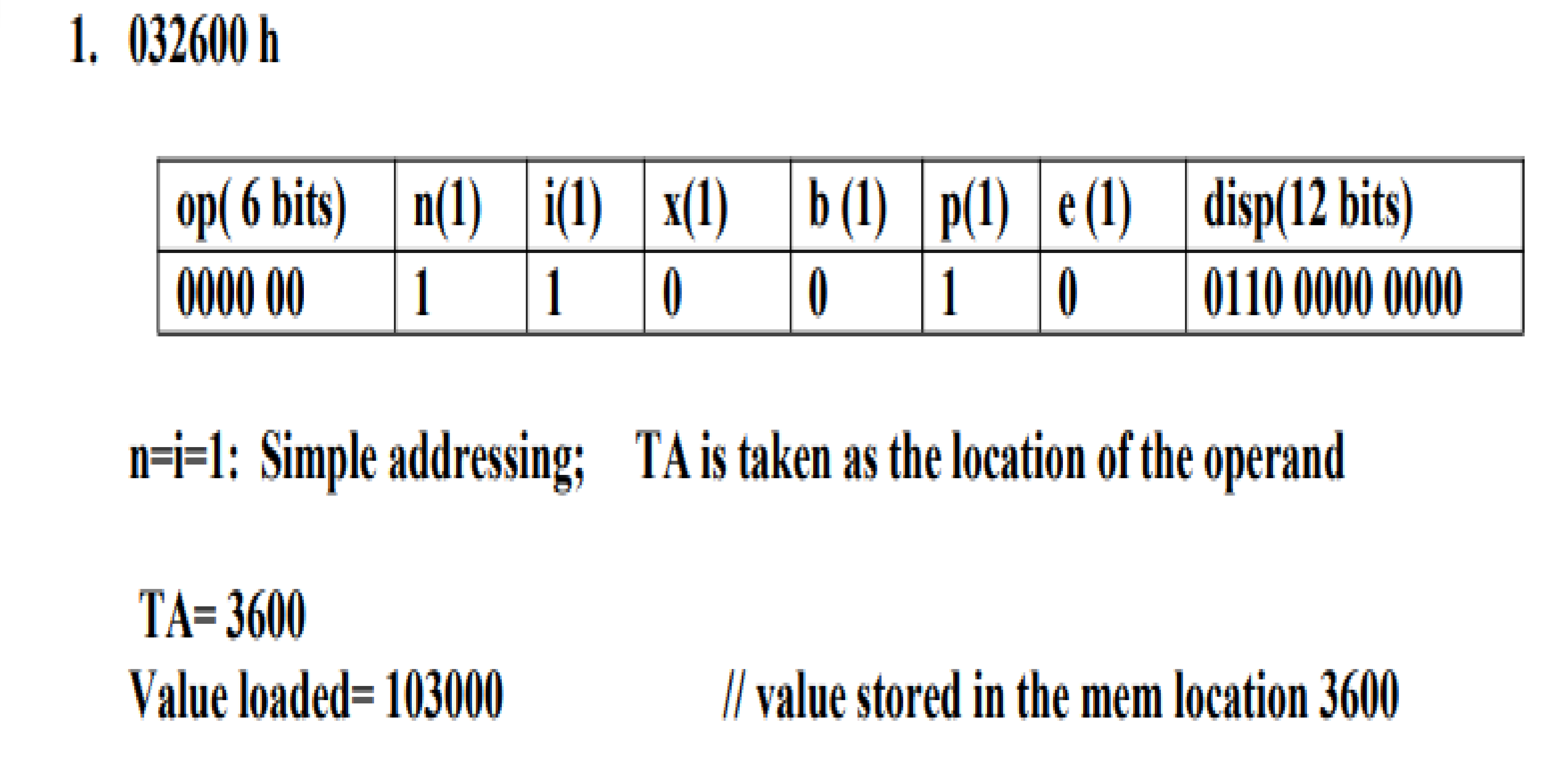


3030 03600

3600 103000

6390 00C303

C303 003030



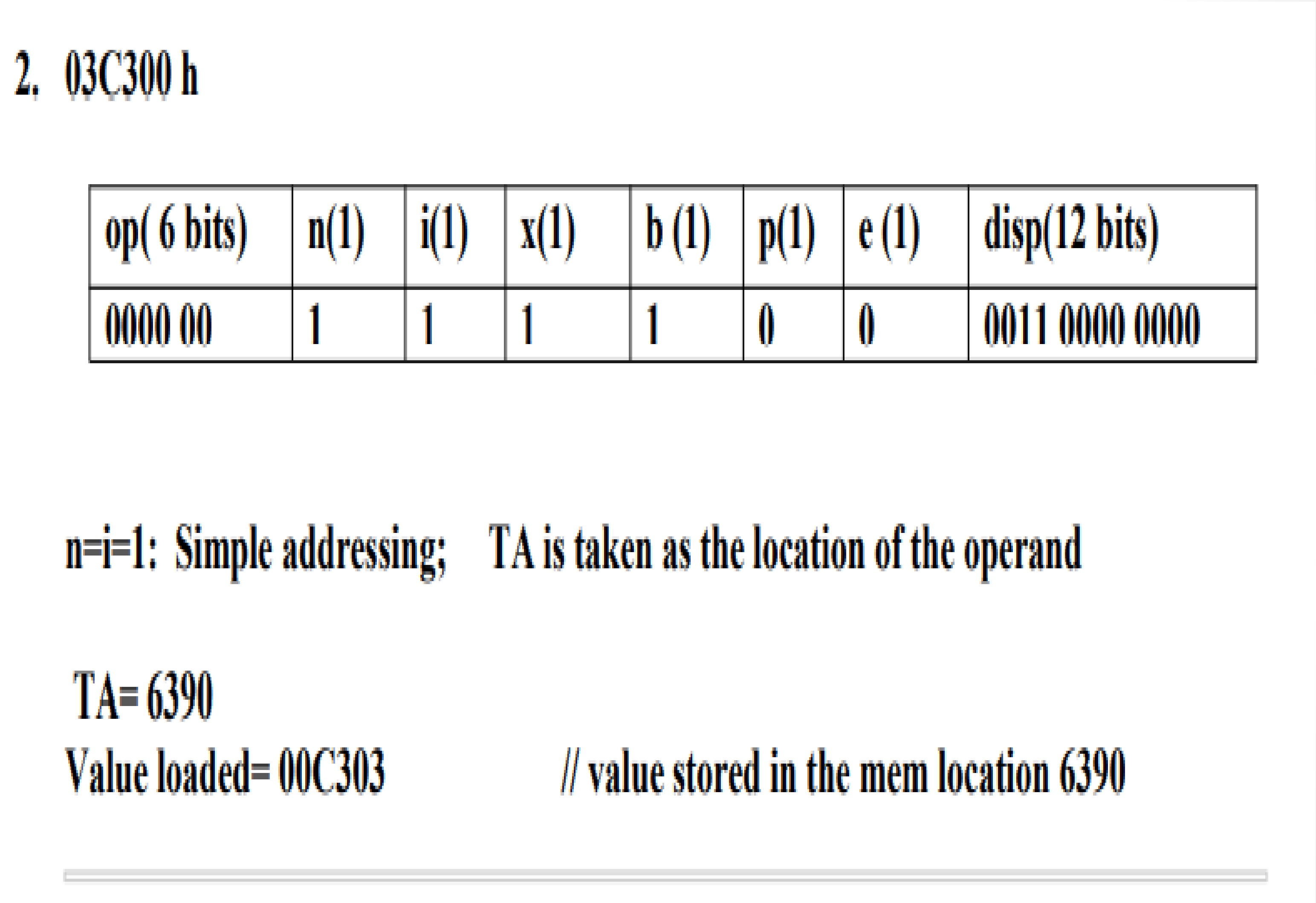
1. 032600 h

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	1	1	0	0	1	0	0110 0000 0000

n=i=1: Simple addressing; TA is taken as the location of the operand

TA= 3600

Value loaded= 103000 // value stored in the mem location 3600



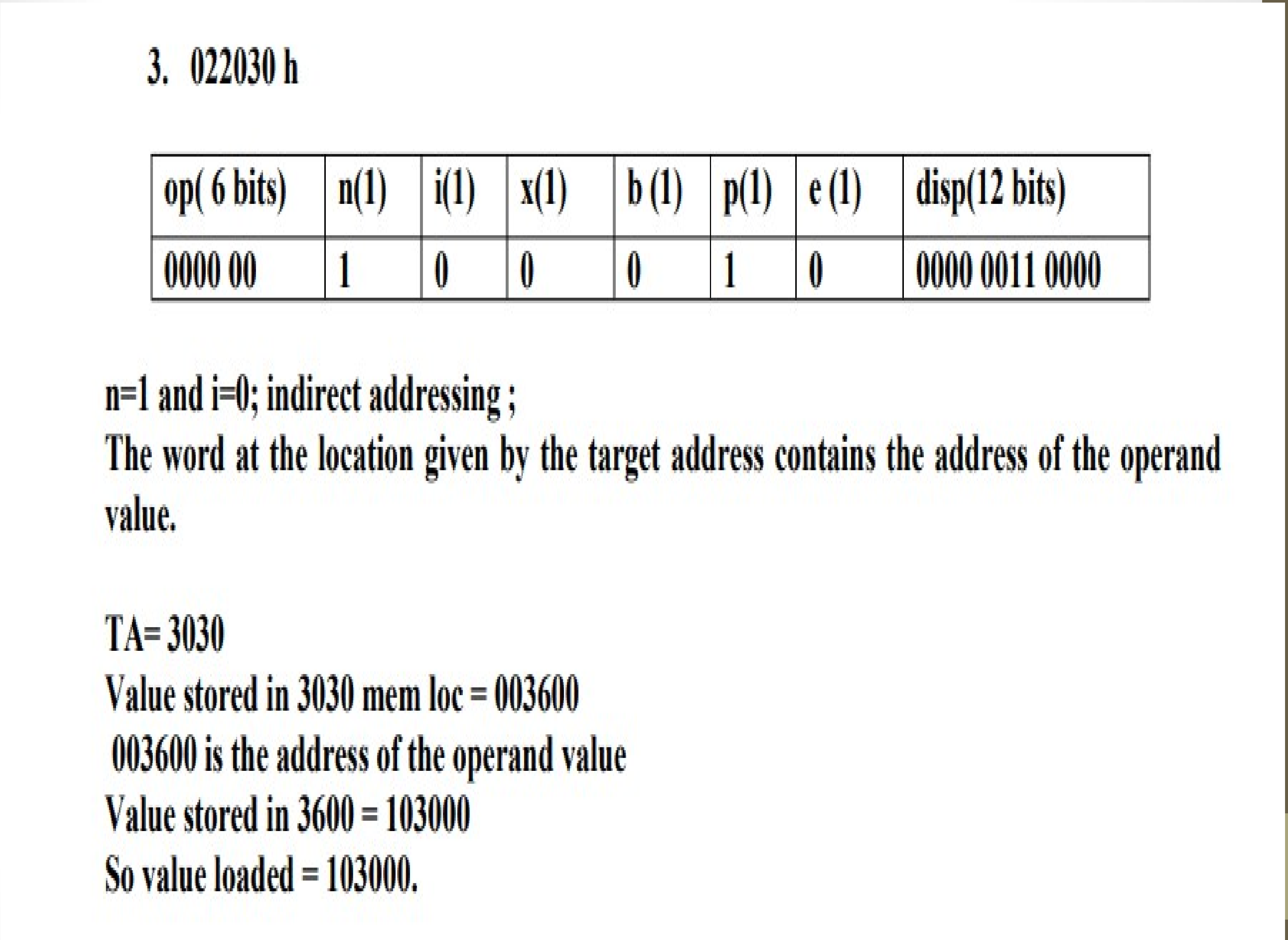
2. 03C300 h

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	1	1	1	1	0	0	0011 0000 0000

n=i=1: Simple addressing; TA is taken as the location of the operand

TA= 6390

Value loaded= 00C303 // value stored in the mem location 6390

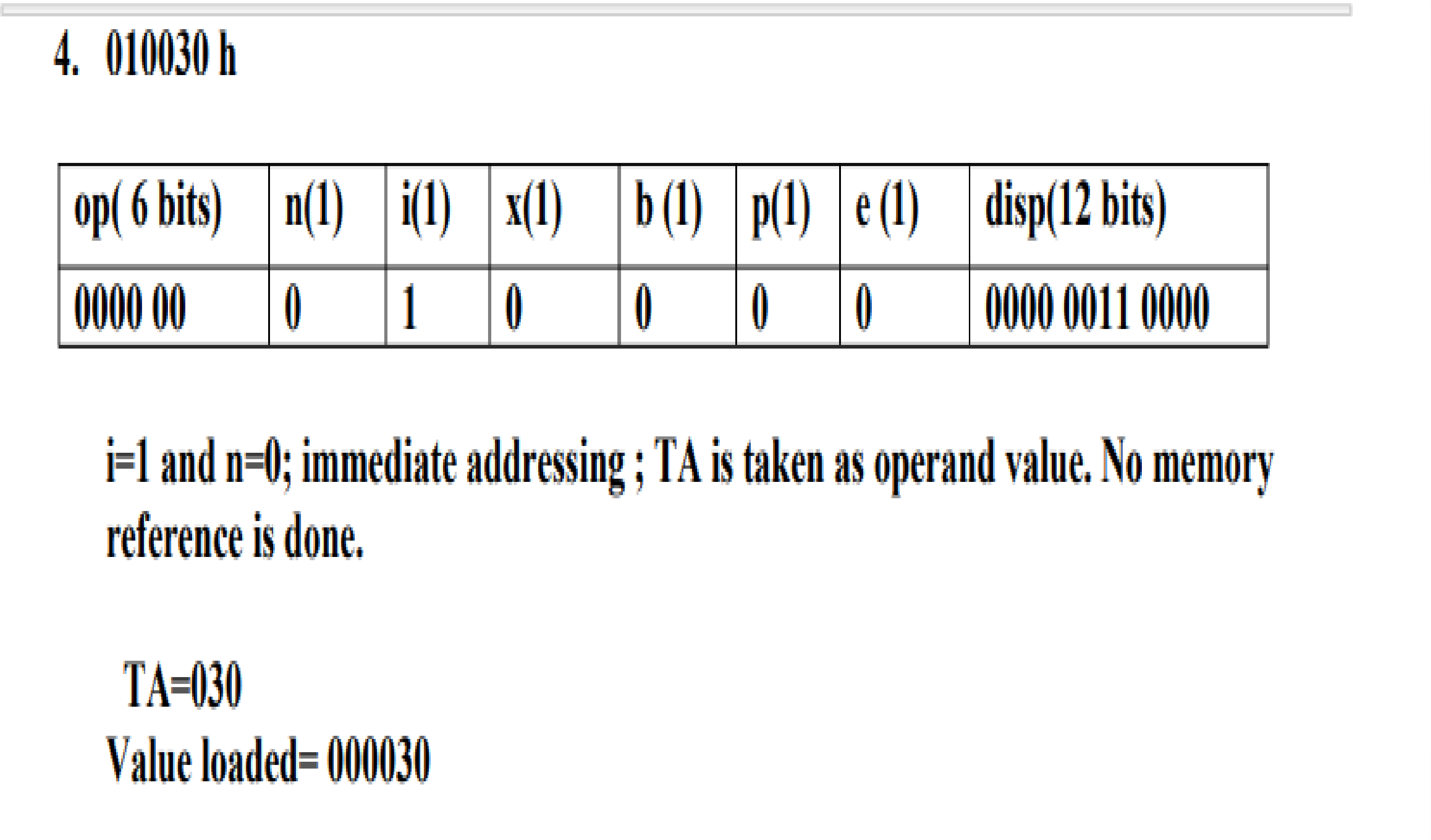


3. 022030 h

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	1	0	0	0	1	0	0000 0011 0000

n=1 and i=0; indirect addressing ;
The word at the location given by the target address contains the address of the operand value.

TA= 3030
Value stored in 3030 mem loc = 003600
003600 is the address of the operand value
Value stored in 3600 = 103000
So value loaded = 103000.

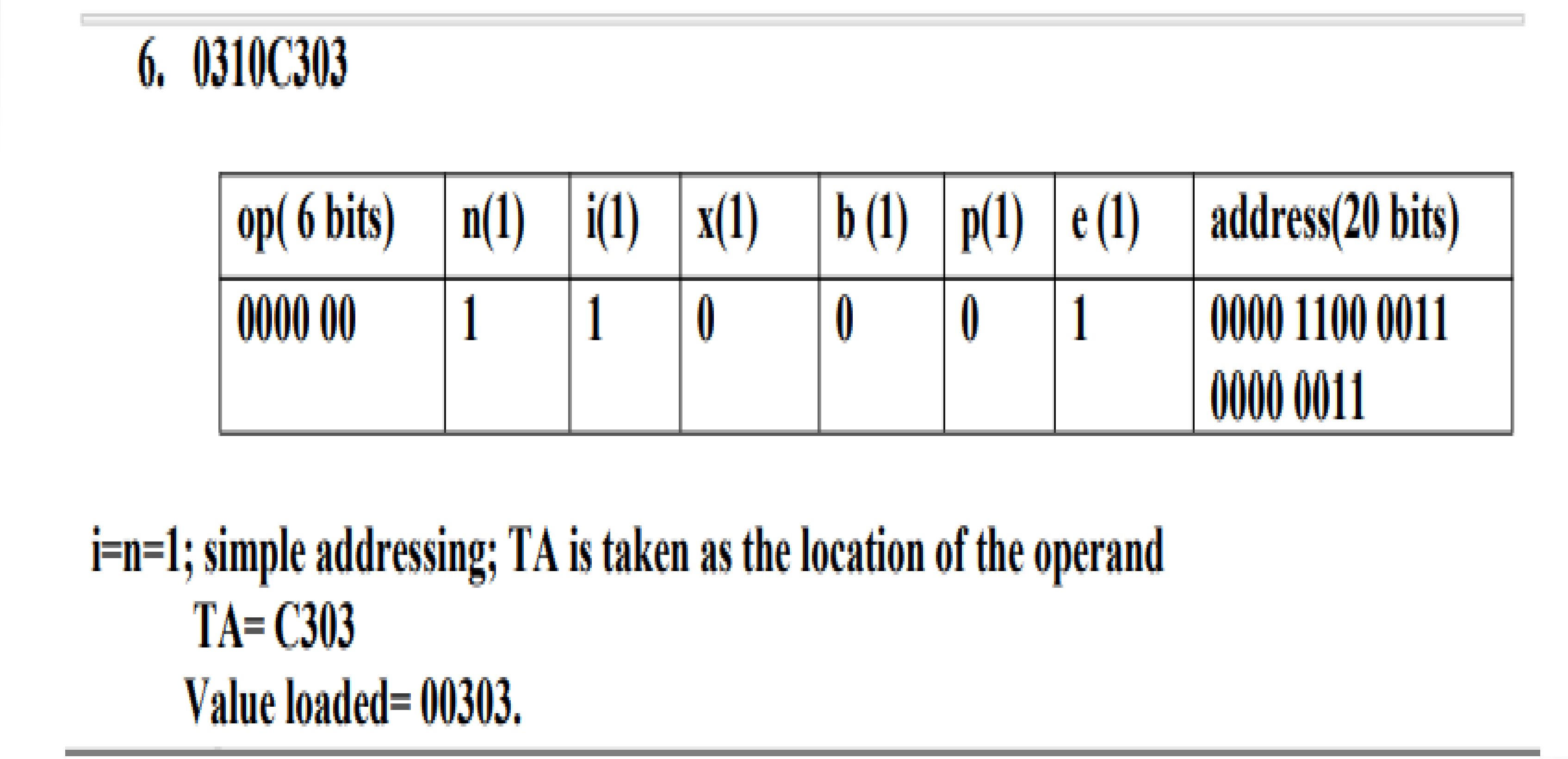


4. 010030 h

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	disp(12 bits)
0000 00	0	1	0	0	0	0	0000 0011 0000

i=1 and n=0; immediate addressing ; TA is taken as operand value. No memory reference is done.

TA=030
Value loaded= 000030



6. 0310C303

op(6 bits)	n(1)	i(1)	x(1)	b (1)	p(1)	e (1)	address(20 bits)
0000 00	1	1	0	0	0	1	0000 1100 0011 0000 0011

i=n=1; simple addressing; TA is taken as the location of the operand

TA= C303

Value loaded= 00303.