

TransEra HTBasic

---

# Training Manual

---

for TransEra HTBasic for Windows

***TransEra***



Copyright © 1988-2000 by TransEra Corporation, Orem, Utah.

Pre-release, September 2000.

Printed in the United States of America. All Rights reserved.

Information in this manual is subject to change without notice and does not represent a commitment on the part of the vendor. Contents of this publication may not be reproduced in any form without express written permission of TransEra Corporation.

The software described in this manual is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement.

High Tech Basic, HTBasic, HTBWin and TransEra are trademarks of TransEra Corporation. HP, HTBasic, and Hewlett-Packard are registered trademarks of Hewlett Packard Company. Windows, Windows 95/98 and Windows NT/2000 are trademarks of Microsoft Corporation. All other brand and product names are trademarks or registered trademarks of their respective companies.

Manual Part Number: HEA-0090



# Contents

---

Chapter 1	Introduction.....	1
Chapter 2	Software Overview.....	3
Chapter 3	Introduction to the Programming Language.....	11
Chapter 4	Starting HTBasic.....	21
Chapter 5	GUI Description.....	27
Chapter 6	Programming & Problem Solving.....	53
Chapter 7	Windows Editor.....	61
Chapter 8	The HTBasic Editor.....	75
Chapter 9	Program Control.....	83
Chapter 10	Mass Storage.....	91
Chapter 11	Data Types & Numeric Computations.....	101
Chapter 12	Program Structure and Flow.....	113
Chapter 13	String Variables.....	125
Chapter 14	Arrays.....	131
Chapter 15	Advanced Array Operations.....	143
Chapter 16	Printing.....	155
Chapter 17	Subroutines, Subprograms, CSUBS and Functions...	167
Chapter 18	Event-Initiated Branching.....	179
Chapter 19	Structured Techniques.....	189
Chapter 20	Data Storage and Retrieval.....	201
Chapter 21	A Closer Look at Files.....	213

Chapter 22	Error Handling.....	225
Chapter 23	Debugging .....	233
Chapter 24	Graphics .....	257
Chapter 25	HTBasic Plus.....	269
Chapter 26	DLL Toolkit.....	302
Chapter 27	DLL Toolkit Example .....	311

# 1 Introduction

---

## Goal

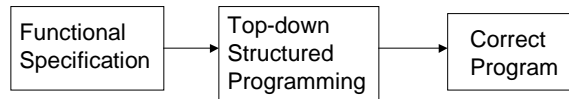
- To define course objectives

## Objectives

- Define a “correct program”
- Identify the three functional components of computer hardware

## Course Objective

To learn to program with HTBasic, utilizing top-down, structured techniques.



Definition: A Correct Program is one that is error free and does exactly what it is intended to do.

## Course Introduction

The goal of this class is to learn how to write “correct” programs in TransEra’s HTBasic computer language. By a “correct” program, one means the program is as error free as designed and the program does exactly what it was intended to do. Reliability is the essential element of computer systems. To achieve reliability, computer programs should be made correct and precise before we try to optimize their performance.

In this course we will study in depth the major aspects of HTBasic as well as a top-down segmented structured programming methodology that will assist us in writing “correct” programs with HTBasic or any other programming language.

## 2 Software Overview

---

### Goal

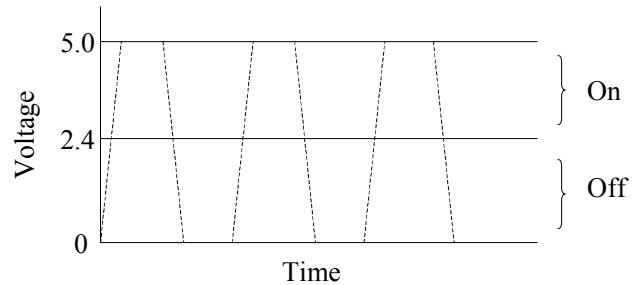
- To establish the role of software as a driving force in the development of the computer
- To present a history of the development of higher level languages

### Objectives

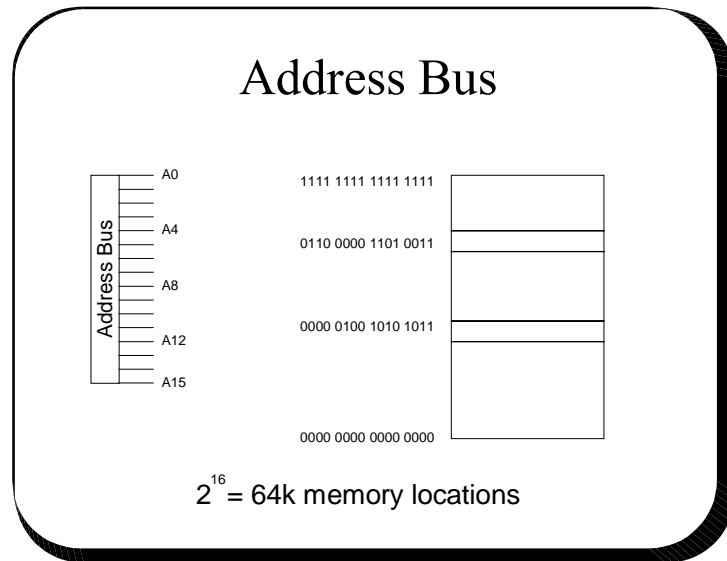
At the end of completion of this module the student will be able to:

- Present the connection between voltage signals and the information they represent
- Present arithmetical methods used by the computer
- Discuss the roles of languages and their development
- Present a brief history of BASIC & HTBasic

## Industry Standard Signal



The myriad components of a typical computer system communicate by sending electrical signals over the different system buses and through the I/O ports. These signals represent coded instructions or data and are created with varying voltage levels. One industry standard signal (TTL) consists of using a voltage level between +2.4 volts and +5.0 volts to indicate “on” while any voltage less than +2.4 volts represents “off”.



How these varying voltage signals are actually implemented can be seen by taking a look at an address bus that is composed of sixteen parallel lines or wires. Number these wires 0 through 15 from right to left. Each line of the bus can represent one of two possible states: off or on depending on the voltage level applied to the line according to the industry standard. Hence, the bus constitutes a binary system with each wire representing one state or “bit”. Symbolically, the bit is either equal to 1, the “on” state, or it is equal to 0, the “off” state. In this manner the sixteen bus lines or bits are used to represent a memory address. Thus, with a total of 16 bits, a total of 2 raised to the 16<sup>th</sup> power or 65,536 unique memory locations can be represented using 16 individual bus lines.

For example, if all of the bits are zero represented by voltage levels below +2.4 volts the address is written:

0000 0000 0000 0000

Turning bits 0 and 8 on, the address becomes

0000 0001 0000 0001

Similarly, if the data bus is eight bits wide, then it has 2 to the 8<sup>th</sup> or 256 possible combinations of 0 and 1, which is then used to represent numbers, alphabetical characters, and special characters. Although character codes can vary from one hardware vendor to another, the majority of vendors implement the industry standard known as ASCII which was developed by the American Society for Communications Interface and Interchange.

## Binary Numbers and Arithmetic

Binary Arithmetic					
Numbers					
<u>Decimal</u>	<u>Binary</u>	<u>Addition</u>		<u>Subtraction</u>	
0	0000	3	0011	3	0011
1	0001				
2	0010	+2	0010	-2	0010
3	0011	5	0101	1	0001
4	0100				
5	0101				
6	0110				
7	0111				
8	1000				
9	1001				

Now consider how simple addition and subtraction are carried out by the computer. Decimal numbers are represented by binary numbers in the following manner. For simplicity's sake consider an eight bit data bus. Then we can form the following table of numbers:

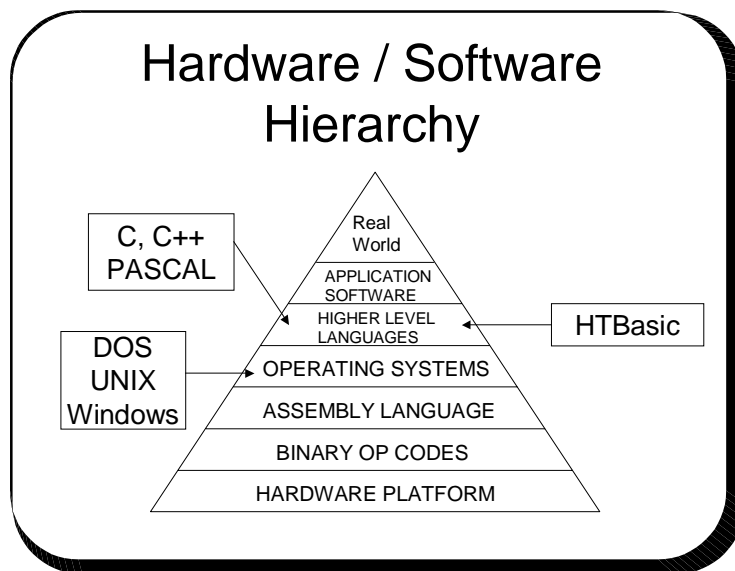
## Assembly Language

In the real world, computers require without fail human written programs of very explicit instructions in order to function in any capacity. These programs must be entered into the computer's memory in a binary bit pattern that is formatted to the architecture of the hardware. The instruction decoder will translate the bit pattern into a series of executable procedures for the CPU as we saw in an earlier example.

The binary bit pattern is called machine language. Programming the computer directly in machine language obviously would be an exceedingly laborious and error prone task. Therefore, microprocessor manufacturers introduced an elementary programming language consisting of a collection of mnemonics representing microprocessor instructions. This collection of mnemonics is called assembly language.

The assembly language is a set of unique mnemonics assigned to represent the instruction set, which is designed into the hardware as the chip is developed. The programmer's job is greatly simplified by using descriptive mnemonics instead of the awkward binary code. The use of assembly code also introduced an intermediate step in the generation of a program. Since the microprocessor only understands binary code, the programmer's original assembly language program is converted or translated to binary format by a special computer program called an "assembler".

## Hardware/Software Hierarchy



Despite the productivity gains made possible with the use of assembly language over binary code, many problems remained. First of all, assembly language is completely machine dependent. Thus, an assembly language program written for one type of processor was not transferable to another type of processor. Second, although programming in assembly language was an improvement over programming in binary code, it still required prodigious efforts and attention to detail. The need for faster program development, quality, portability, and increased user friendliness

drove the development of higher level languages which avoided cryptic mnemonics and which were much closer to natural language. From this effort, the higher level programming languages of Fortran, Cobol, and BASIC evolved.

Every computer has an operating system, which is a special program that micromanages the computer's hardware. The operating system is responsible for scheduling and allocating computer resources in an efficient and timely manner. Windows is the common operating system on the PC.

## **A Bit of BASIC History**

Unlike spoken languages such as English and Chinese, BASIC does have a known birth date: May 1, 1964. That's when two faculty members and a handful of students at Dartmouth College first saw correct answers emerge using a new language they had created for their timesharing computer system.

Before BASIC, most programming used batch processing: a student or engineer would write a program, punch out a card for each instruction, then hand over the pack of cards to a computer center. Then, hours (or days) later, the cards were returned, along with a printout showing error codes. It was tedious and time-consuming, usually requiring several "runs" to get the program right.

Those pioneers at Dartmouth created an alternative. They called it BASIC, for Beginner's All-purpose Symbolic Instruction Code. From the outset, it was clear that the word "Beginner's" was a misnomer, since BASIC quickly found favor with a full spectrum of the scientific community.

For one thing, it was easier to use: statements were easy-to-remember words like GOTO, PRINT, and IF...THEN. There was one instruction per program line, and each line began with a keyword. Another advantage of BASIC was its instant feedback: when a user on the timesharing system typed a program line, it was checked immediately for syntax. Error messages popped up right away if the statement was wrong, so correction could be immediate.

Finally, BASIC was hardware-independent. A user didn't have to know anything about file structure to save a program, for instance.

Those early BASIC programmers had a tiny vocabulary to work with - just over a dozen commands and statements in all. By contrast, today's HTBasic boasts over 400 keywords.