Programmable Calculators and Models

Melvin Truong

First Edition, 2012

ISBN 978-81-323-3206-0

Published by:
Research World
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1

Programmable Calculator

Programmable calculators are calculators capable of being programmed much like a computer.

Since the early 1990s, most of these flexible handheld units belong to the class of graphing calculators. Before the mass-manufacture of inexpensive dot-matrix LCD displays, however, programmable calculators usually featured a one-line numeric or alphanumeric display.

Calculator programming

Programmable calculators allow the user to write and store programs in the calculator in order to solve difficult problems or automate an elaborate procedure.

Programming capability appears most commonly (although not exclusively) in graphing calculators, as the larger screen allows multiple lines of source code to be viewed simultaneously (i.e., without having to scroll to the next/previous display line). Originally, calculator programming had to be done in the calculator's own command language, but as calculator hackers discovered ways to bypass the main interface of the calculators and write assembly language programs, calculator companies (particularly Texas Instruments) began to support native-mode programming on their calculator hardware, first revealing the hooks used to enable such code to operate, and later explicitly building in facilities to handle such programs directly from the user interface.

Many programs written for calculators can be found on the internet. Users can download the programs to a personal computer, and then upload them to the calculator using a specialized link cable, infrared wireless link or through a memory card,. Sometimes these programs can also be run through emulators on the PC.

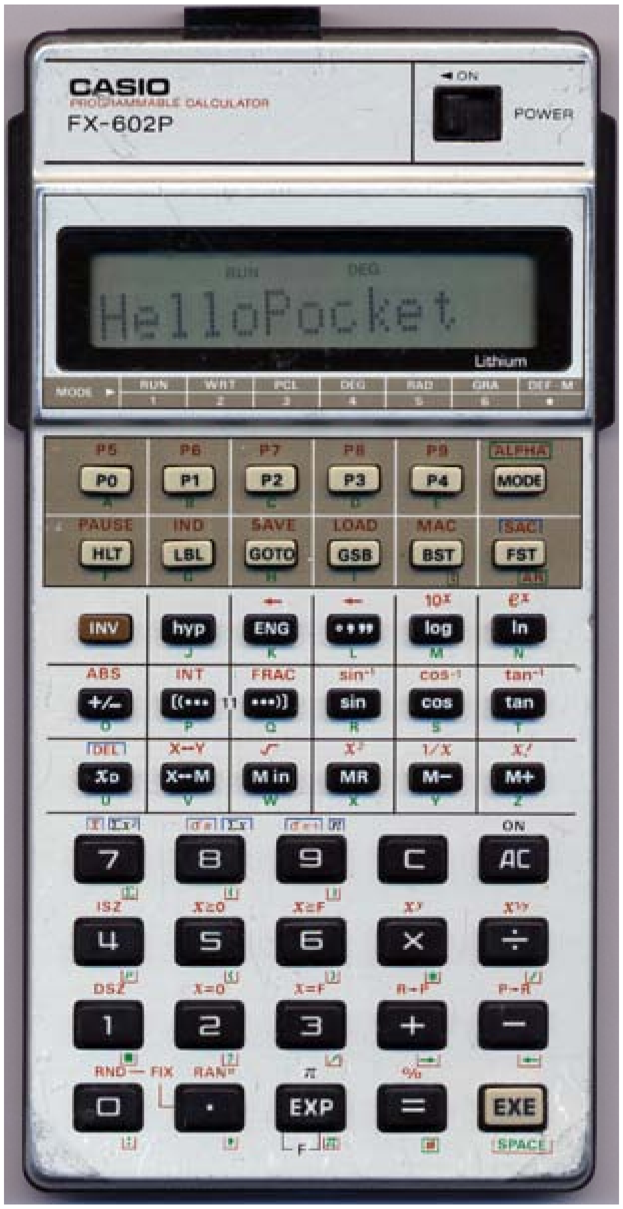
One possibility arising from the above is writing interpreters, compilers, and translator programmes for additional languages for programming the machines; BBC Basic has already been ported to the TI 83 and 84 series and other on-board languages and programming tools discussed by many include Fortran, awk, Pascal, Rexx, perl, Common Lisp, Python, tcl, and various Unix shells.

Commonly available programs for calculators include everything from math/science related problem solvers to video games, as well as so-called demos. Much of this code is user-created freeware or even open source, though commercial software, particularly for educational and science/engineering markets, is also available.

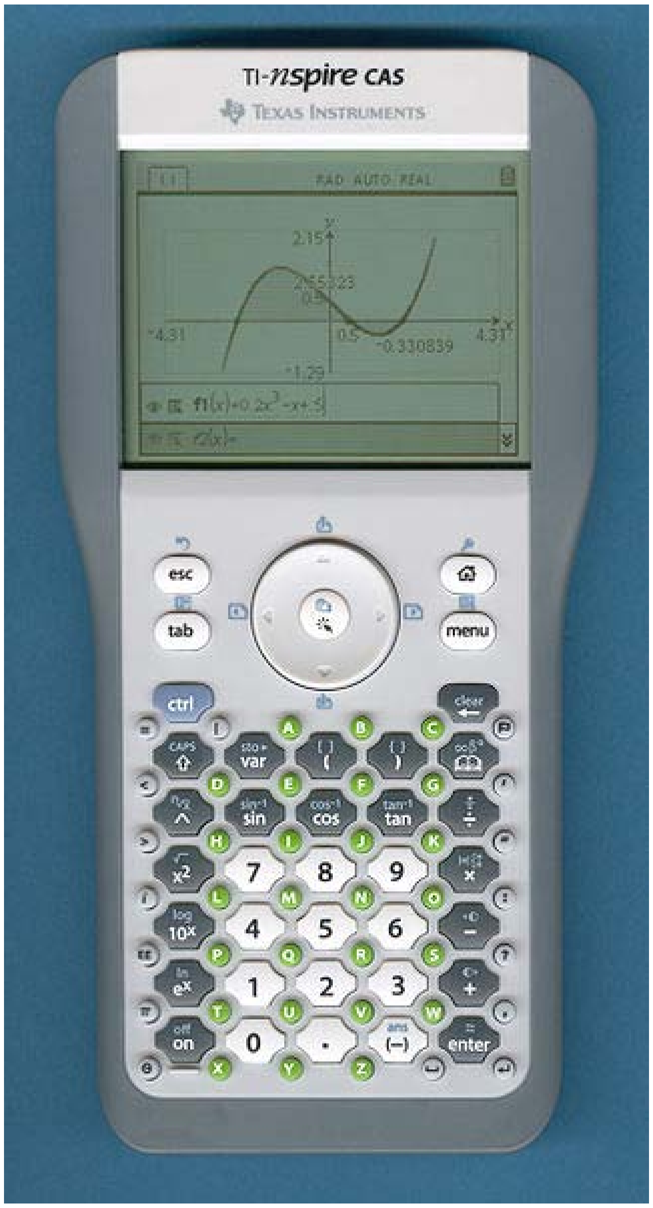
A TI-59 showing one card in the holder on the front of the calculator and another being inserted into the card reader in the side.

HP-41CX with magnetic card reader and thermal printer

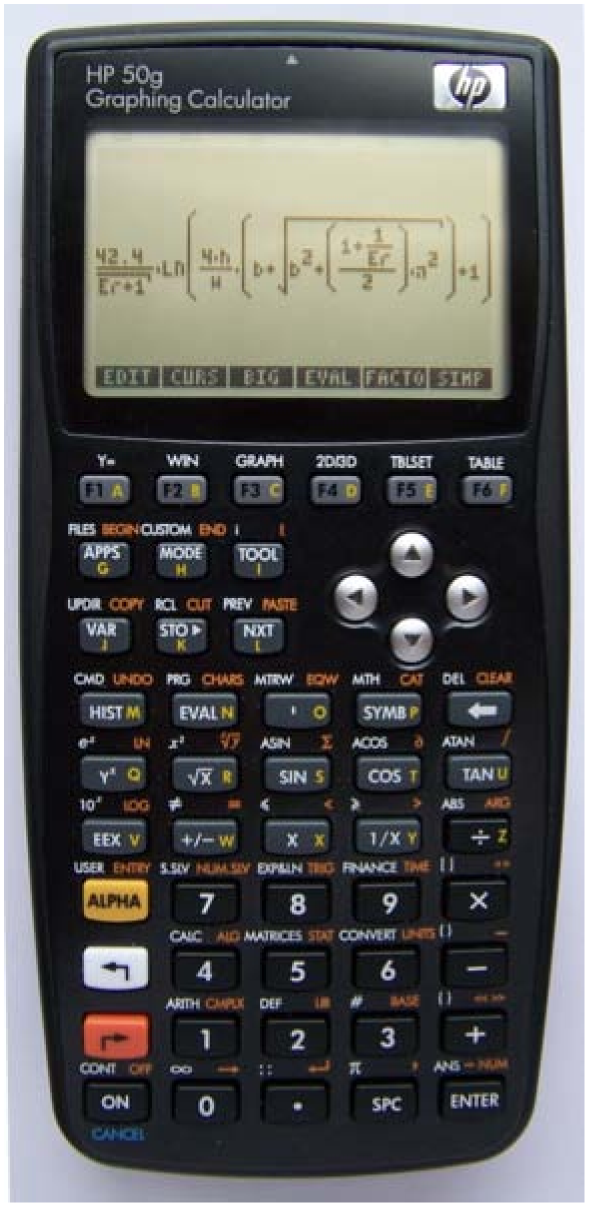
A complete range of programmable calculators were developed in former USSR. Some of them (like pictured above MK-52), were used even in space missions.



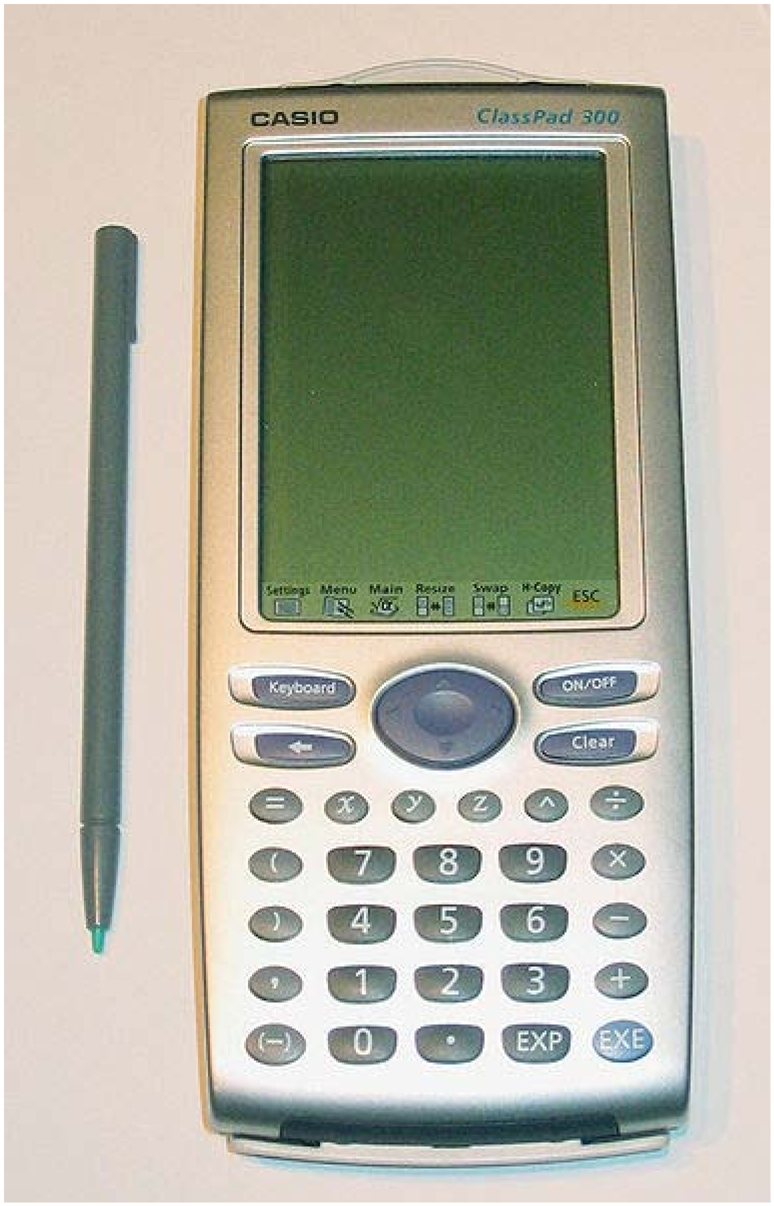
A 28 year old FX-602P in working condition



A TI-Nspire CAS Calculator



HP50g graphing calculator, with the Equation Editor being used



Casio Class Pad 300 touch screen calculator

Programming languages

Keystroke programming

In the early days most programmable calculators used a very simplified programming language, often based either on recording actual keystrokes or bytecode if the keystrokes were merged. Calculators supporting such programming were Turing-complete if they supported both conditional statements and indirect addressing of memory. Notable examples of Turing complete calculators were Casio FX-602P series, the HP-41 and the TI-59. Keystroke programming is still used in mid-range calculators like the HP 35s.

BASIC

BASIC is a widespread programming language commonly adapted to desktop computers and pocket computers. The most common languages now used in high range calculators are proprietary BASIC-style dialects as used by CASIO and TI (TI-BASIC). Those BASIC dialects are optimised for calculator use combining the advantages of BASIC and keystroke programming. Those dialects have little in common with mainstream BASIC .

A complete port of BBC Basic to the TI-83 subfamily of calculators is now available. It is put on the calculator via a cable or IrDA connexion with a computer.

RPL

RPL is a special Forth-like programming language used by Hewlett Packard in its high range devices. The first device with RPL calculator was the HP-28C released in 1987.

Assembler

Machine language programming was often discouraged on early calculator models; however, dedicated platform hackers discovered ways to bypass the built-in interpreters on some models and program the calculator directly in assembly language, a technique that was first discovered and utilized on the TI-85 due to a programming flaw in a mode-switching key. By the time the TI-83 came out, TI and HP had realized the need to address the support needs of homebrew programmers, and started to make assembly language libraries and documentation available for prospective developers. Software, particularly games, could now be nearly as fast and as graphical as their Game Boy counterparts, and TI in particular would later formalize assembly programming into support for packaged applications for future calculators such as the TI-83 Plus and TI-89; HP includes some onboard support for assembler programming on the HP-50g, its current top-of-the-line calculator model.

Programmes and toolkits to allow on-board assembly-like programming (often Intel 80*86 even if the actual processor in the calculator is something completely different like a Zilog or Motorola chip) are in the beta stage in at least two implementations—the native Basic variant can be enhanced by user-defined functions and procedures as well as assembly and C modules developed on a computer and uploaded to the calculator which allow for writing and running "pseudo assembly" programmes just as one would the Basic type ones. Other languages like Rexx, awk, Perl, and Windows NT and some Unix shells can also be implemented in this fashion on many calculators of this type.

Other Languages

The gcc development suite is available for several models of HP and TI calculators, meaning that C, C++, Fortran 77, and inline assembly language can be used to develop a programme on the computer side and then upload it to the calculator.

Projects in development by third parties include on-board and/or computer-side converters, interpreters, code generators, macro assemblers, or compilers for Fortran, other Basic variants, awk, C, Cobol, Rexx, Perl, Python, Tcl, Pascal Delphi, and operating system shells like DOS/Win95 batch, OS/2 batch, WinNT/2000 shell, Unix shells, and DCL.

Persistent memory

One important feature of programmable calculators is the availability of some form of persistent memory. Without persistent memory, programs have to be re-entered whenever power is lost, making the device cumbersome. Persistent memory can be within the calculator or as a separate device. Programmable calculators could use more than one technique.

Magnetic card reader / writer

Magnetic card readers were among the first persistent memory options available. The entered programs are stored on magnetic strips. Those were easy to transport, and the reader/writer was compact in size. However, the reader/writer as well as the magnetic strips were quite expensive. The last and most notable devices to use magnetic strips were the HP-41C and TI-59.

Continuous memory

Continuous memory does not lose its content when the calculator is switched off. With continuous memory the user can, for example, change batteries without losing the entered programs.

Cassette tape

Compact Cassettes offered a simple cheap alternative to magnetic cards. Usually an interface module such as the Casio FA-1 was used to connect the calculator to a standard compact cassette recorder .

Sharp and HP also sold dedicated micro or mini cassette recorders which connected directly to the calculator. These set-ups while being more practical and reliable were also more expensive.

Semi-continuous memory

As memory demands rose, it became more difficult to create true continuous memory and developers sought alternatives. With semi-continuous memory memory content was only preserved if specific battery changing rules were observed. The most common rules were:

1. A special backup battery would ensure that the memory was not lost while the main batteries were changed.

2. Batteries had to be changed in a relatively short time. For example with the HP 35s batteries have to be changed in less than 2 minutes.
3. At least two main batteries were used and could only be changed one at a time.

PC-connection

Programs and data are transferred to a Personal computer for storage. The transfer is done by the following connection methods (chronological order of appearance) RS-232, IrDA and USB. This method has the advantage of being very cost efficient and is usually faster than cassette interface. These advantages are offset by the need of a Personal computer. An early example of PC connection is the Casio FX-603P in conjunction with the Casio FA-6 interface. In this set-up transfer was done in Plain text so the program and data could not only be stored but also edited with a standard Text editor.

Programmable calculators and pocket computers

Throughout the 80's and the beginning of the 90's programmable calculators stood in competition with pocket computers, with high end calculators sharing many similarities. For example both device types were programmable in unstructured BASIC and with few exceptions featured QWERTY keyboards. However there were also some differences:

- BASIC programmable calculators often featured an additional "calculator like" keyboard and a special calculator mode in which the system behaved like a Scientific calculator.
- Pocket computers often offered additional programming languages as option. The Casio PB-2000 for example offered ANSI-C, BASIC, Assembler and Lisp.

Companies often had both device types in their product portfolio. Casio for example sold some BASIC programmable calculators as part of their "fx-" calculator series and pocket computer the dedicated "pb-" series while Sharp on marketed all BASIC programmable devices as pocket computer.

Chapter 2

RPL (Programming Language)

	RPL
Paradigm	stack, structured
Appeared in	1984
Designed by	Hewlett-Packard
Dialects	System RPL, User RPL
Influenced by	Forth, infix notation
OS	HP calculators

The **RPL programming language** (RPL meaning **ROM-based procedural language** following Hewlett-Packard or, alternatively, **Reverse Polish LISP**) is a handheld calculator system and application programming language used on Hewlett-Packard's engineering graphing RPN calculators of the HP-28, HP-48, HP-49 and HP-50 series, but it is also usable on non-RPN calculators, such as the HP-39 series.

RPL is a structured programming language based on RPN but equally capable of processing algebraic expressions and formulae, implemented as a threaded interpreter. RPL has many similarities to Forth, both languages being stack-based, and of course the list-based LISP. Contrary to previous HP RPN calculators, which had a fixed four-level stack, the stack used by RPL is only limited by available calculator RAM.

RPL originated from HP's Corvallis, Oregon development facility in 1984 as a replacement for the previous practice of implementing the operating systems of calculators in assembly language. According to a quote by Dr. William Wickes, one of the original RPL developers, "the development team never calls it anything but (the initials) RPL."

Variants

The internal low- to medium-level variant of RPL, called **System RPL** (or **SysRPL**) is used on some earlier HP calculators as well as the aforementioned ones, as part of their operating system implementation language. This variant of RPL is not accessible to the calculator user without the use of external tools. It is possible to cause a serious crash while coding in SysRPL, so caution must be used while using it. The high-level **User RPL** (or **UserRPL**) version of the language is available on said graphing calculators for developing textual as well as graphical application programs. All UserRPL programs are internally represented as SysRPL programs, but use only a safe subset of the available SysRPL commands. The error checking that is a part of UserRPL commands, however, makes UserRPL programs noticeably slower than equivalent SysRPL programs. The UserRPL command SYSEVAL tells the calculator to process designated parts of a UserRPL program as SysRPL code.

Control blocks

RPL control blocks are not strictly postfix. Although there are some notable exceptions, the control block structures appear as they would in a standard infix language. The calculator manages this by allowing the implementation of these blocks to skip ahead in the program stream as necessary.

Conditional statements

IF/THEN/ELSE/END

RPL supports basic conditional testing through the IF/THEN/ELSE structure. The basic syntax of this block is:

```
IF condition THEN if-true [ELSE if-false] END
```

The following example tests to see if the number at the bottom of the stack is "1" and, if so, replaces it with "Equal to one":

```
« IF 1 == THEN "Equal to one" END »
```

The IF construct evaluates the condition then tests the bottom of the stack for the result. As a result RPL can optionally support FORTH-style IF blocks, allowing the condition to be determined before the block. By leaving the condition empty, the IF statement will not make any changes to the stack during the condition execution and will use the existing result at the bottom of the stack for the test:

```
« 1 == IF THEN "Equal to one" END »
```

IFT/IFTE

Postfix conditional testing may be accomplished by using the IFT ("if-then") and IFTE ("if-then-else") functions.

IFT and IFTE pop two or three commands off the stack, respectively. The topmost value is evaluated as a boolean and, if true, the second topmost value is pushed back on the stack. IFTE allows a third "else" value that will be pushed back on the stack if the boolean is false.

The following example uses the IFT function to pop an object from the bottom of the stack and, if it is equal to 1, replaces it with "One":

```
« 1 == "One" IFT »
```

The following example uses the IFTE function to pop an object from the bottom of the stack and, if it is equal to 1, replaces it with "One". If it does not equal 1, it replaces it with the string "Not one":

```
« 1 == "One" "Not one" IFTE »
```

IFT and IFTE will evaluate a program block given as one of its arguments, allowing a more compact form of conditional logic than an IF/THEN/ELSE/END structure. The following example pops an object from the bottom of the stack, and replaces it with "One", "Less", or "More", depending on whether it is equal to, less than, or greater than 1.

```
«
  DUP 1 ==
  « DROP "One" »
  « 1 < "Less" "More" IFTE »
  IFTE
»
```

CASE/THEN/END

To support more complex conditional logic, RPL provides the CASE/THEN/END structure for handling multiple exclusive tests. Only one of the branches within the CASE statement will be executed. The basic syntax of this block is:

```
CASE
 condition_1 THEN if-condition_1 END
  ...
 condition_n THEN if-condition_n END
 if-none
END
```

The following code illustrates the use of a CASE/THEN/END block. Given a letter at the bottom of the stack, it replaces it with its string equivalent or "Unknown letter":

```
«
  CASE
     DUP "A" == THEN "Alpha" END
     DUP "B" == THEN "Beta" END
     DUP "G" == THEN "Gamma" END
     "Unknown letter"
  END
  SWAP DROP  @ Get rid of the original letter
»
```

This code is identical to the following nested IF/THEN/ELSE/END block equivalent:

```
«
   IF DUP "A" ==
   THEN
      "Alpha"
   ELSE
      IF DUP "B" == THEN
         "Beta"
      ELSE
         IF DUP "G" == THEN
            "Gamma"
         ELSE
            "Unknown letter"
         END
      END
   END
   SWAP DROP  @ Get rid of the original letter
»
```

Looping statements

FOR/NEXT

RPL provides a FOR/NEXT statement for looping from one index to another. The index for the loop is stored in a temporary local variable that can be accessed in the loop. The syntax of the FOR/NEXT block is:

```
index_from index_to FOR variable_name loop_statement NEXT
```

The following example uses the FOR loop to sum the numbers from 1 to 10. The index variable of the FOR loop is "I":

```
«
   0       @ Start with zero on the stack
   1 10    @ Loop from 1 to 10
   FOR I   @ "I" is the local variable
      I +  @ Add "I" to the running total
   NEXT    @ Repeat...
»
```

START/NEXT

The START/NEXT block is used for a simple block that runs from a start index to an end index. Unlike the FOR/NEXT loop, the looping variable is not available. The syntax of the START/NEXT block is:

```
index_from index_to START loop_statement NEXT
```

FOR/STEP and START/STEP

Both FOR/NEXT and START/NEXT support a user-defined step increment. By replacing the terminating NEXT keyword with an increment and the STEP keyword, the loop variable will be incremented or decremented by a different value than the default of +1. For instance, the following loop steps back from 10 to 2 by decrementing the loop index by 2:

```
« 10 2 START -2 STEP »
```

WHILE/REPEAT/END

The WHILE/REPEAT/END block in RPL supports an indefinite loop with the condition test at the start of the loop. The syntax of the WHILE/REPEAT/END block is:

```
WHILE condition REPEAT loop_statement END
```

DO/UNTIL/END

The DO/UNTIL/END block in RPL supports an indefinite loop with the condition test at the end of the loop. The syntax of the DO/UNTIL/END block is:

```
DO loop_statement UNTIL condition END
```

Chapter 3

Assembly Language

An **assembly language** is a low-level programming language for computers, microprocessors, microcontrollers, and other programmable devices. It implements a symbolic representation of the machine codes and other constants needed to program a given CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on mnemonics that symbolize processing steps (instructions), processor registers, memory locations, and other language features. An assembly language is thus specific to a certain physical (or virtual) computer architecture. This is in contrast to most high-level programming languages, which, ideally, are portable.

A utility program called an *assembler* is used to translate assembly language statements into the target computer's machine code. The assembler performs a more or less isomorphic translation (a one-to-one mapping) from mnemonic statements into machine instructions and data. This is in contrast with high-level languages, in which a single statement generally results in many machine instructions.

Many sophisticated assemblers offer additional mechanisms to facilitate program development, control the assembly process, and aid debugging. In particular, most modern assemblers include a macro facility (described below), and are called *macro assemblers*.

Key concepts

Assembler

Typically a modern **assembler** creates object code by translating assembly instruction mnemonics into opcodes, and by resolving symbolic names for memory locations and other entities. The use of symbolic references is a key feature of assemblers, saving tedious calculations and manual address updates after program modifications. Most assemblers also include macro facilities for performing textual substitution—e.g., to generate common short sequences of instructions as inline, instead of *called* subroutines.

Assemblers are generally simpler to write than compilers for high-level languages, and have been available since the 1950s. Modern assemblers, especially for RISC architectures, such as SPARC or POWER, as well as x86 and x86-64, optimize Instruction scheduling to exploit the CPU pipeline efficiently.

Number of passes

There are two types of assemblers based on how many passes through the source are needed to produce the executable program.

- One-pass assemblers go through the source code once and assume that all symbols will be defined before any instruction that references them.
- Two-pass assemblers create a table with all symbols and their values in the first pass, then use the table in a second pass to generate code. The assembler must at least be able to determine the length of each instruction on the first pass so that the addresses of symbols can be calculated.

The advantage of a one-pass assembler is speed, which is not as important as it once was with advances in computer speed and abilities. The advantage of the two-pass assembler is that symbols can be defined anywhere in program source code. This lets programs be defined in more logical and meaningful ways, making two-pass assembler programs easier to read and maintain.

High-level assemblers

More sophisticated high-level assemblers provide language abstractions such as:

- Advanced control structures
- High-level procedure/function declarations and invocations
- High-level abstract data types, including structures/records, unions, classes, and sets
- Sophisticated macro processing (although available on ordinary assemblers since late 1950s for IBM 700 series and since 1960's for IBM/360, amongst other machines)
- Object-oriented programming features such as classes, objects, abstraction, polymorphism, and inheritance

Use of the term

Note that, in normal professional usage, the term *assembler* is often used ambiguously: It can refer to an assembly language itself, as well as to an assembler utility. Thus: "CP/CMS was written in S/360 assembler" as well as "ASM-H was a widely-used S/370 assembler."

Assembly language

A program written in assembly language consists of a series of mnemonic statements and meta-statements (known variously as directives, pseudo-instructions and pseudo-ops), comments and data. These are translated by an assembler to a stream of executable instructions that can be loaded into memory and executed. Assemblers can also be used to produce blocks of data, from formatted and commented source code, to be used by other code.

Take for example, the instruction that tells an x86/IA-32 processor to move an immediate 8-bit value into a register. The binary code for this instruction is 10110 followed by a 3-bit identifier for which register to use. The identifier for the *AL* register is 000, so the following machine code loads the *AL* register with the data 01100001.

```
10110000 01100001
```

This binary computer code can be made more human-readable by expressing it in hexadecimal as follows

```
B0 61
```

Here, `B0` means 'Move a copy of the following value into *AL*', and `61` is a hexadecimal representation of the value 01100001, which is 97 in decimal. Intel assembly language provides the mnemonic MOV (an abbreviation of *move*) for instructions such as this, so the machine code above can be written as follows in assembly language, complete with an explanatory comment if required, after the semicolon. This is much easier to read and to remember.

```
MOV AL, 61h       ; Load AL with 97 decimal (61 hex)
```

At one time, many assembly language mnemonics were three letter abbreviations, such as JMP for *jump*, INC for *increment*, etc. Modern processors have a much larger instruction set and many mnemonics are now longer, for example FPATAN for "*floating point partial arctangent*" and BOUND for "*check array index against bounds*". Many assembly language statements consist of an opcode mnemonic followed by a comma-separated list of data, arguments or parameters.

The same mnemonic MOV refers to a family of related opcodes to do with loading, copying and moving data, whether these are immediate values, values in registers, or memory locations pointed to by values in registers. The opcode 10110000 (`B0`) copies an 8-bit value into the *AL* register, while 10110001 (`B1`) moves it into *CL* and 10110010 (`B2`) does so into *DL*. Assembly language examples for these follow.

```
MOV AL, 1h        ; Load AL with immediate value 1
MOV CL, 2h        ; Load CL with immediate value 2
MOV DL, 3h        ; Load DL with immediate value 3
```

The syntax of MOV can also be more complex as the following examples show.

```
MOV EAX, [EBX]        ; Move the 4 bytes in memory at the address
contained in EBX into EAX
MOV [ESI+EAX], CL ; Move the contents of CL into the byte at address
ESI+EAX
```

In each case, the MOV mnemonic is translated directly into an opcode in the ranges 88-8E, A0-A3, B0-B8, C6 or C7 by an assembler, and the programmer does not have to know or remember which.

Transforming assembly language into machine code is the job of an assembler, and the reverse can at least partially be achieved by a disassembler. Unlike high-level languages, there is usually a one-to-one correspondence between simple assembly statements and machine language instructions. However, in some cases, an assembler may provide *pseudoinstructions* (essentially macros) which expand into several machine language instructions to provide commonly needed functionality. For example, for a machine that lacks a "branch if greater or equal" instruction, an assembler may provide a pseudoinstruction that expands to the machine's "set if less than" and "branch if zero (on the result of the set instruction)". Most full-featured assemblers also provide a rich macro language (discussed below) which is used by vendors and programmers to generate more complex code and data sequences.

Each computer architecture and processor architecture usually has its own machine language. On this level, each instruction is simple enough to be executed using a relatively small number of electronic circuits. Computers differ by the number and type of operations they support. For example, a new 64-bit machine would have different circuitry from a 32-bit machine. They may also have different sizes and numbers of registers, and different representations of data types in storage. While most general-purpose computers are able to carry out essentially the same functionality, the ways they do so differ; the corresponding assembly languages reflect these differences.

Multiple sets of mnemonics or assembly-language syntax may exist for a single instruction set, typically instantiated in different assembler programs. In these cases, the most popular one is usually that supplied by the manufacturer and used in its documentation.

Language design

Basic elements

There is a large degree of diversity in the way the authors of assemblers categorize statements and in the nomenclature that they use. In particular, some describe anything other than a machine mnemonic or extended mnemonic as a pseudo-operation (pseudo-op). A typical assembly language consists of 3 types of instruction statements that are used to define program operations:

- Opcode mnemonics
- Data sections
- Assembly directives

Opcode mnemonics and extended mnemonics

Instructions (statements) in assembly language are generally very simple, unlike those in high-level language. Generally, a mnemonic is a symbolic name for a single executable machine language instruction (an opcode), and there is at least one opcode mnemonic defined for each machine language instruction. Each instruction typically consists of an *operation* or *opcode* plus zero or more *operands*. Most instructions refer to a single value, or a pair of values. Operands can be immediate (typically one byte values, coded in the instruction itself), registers specified in the instruction, implied or the addresses of data located elsewhere in storage. This is determined by the underlying processor architecture: the assembler merely reflects how this architecture works. *Extended mnemonics* are often used to specify a combination of an opcode with a specific operand, e.g., the System/360 assemblers use **B** as an extended mnemonic for **BC** with a mask of 15 and **NOP** for **BC** with a mask of 0.

Extended mnemonics are often used to support specialized uses of instructions, often for purposes not obvious from the instruction name. For example, many CPU's do not have an explicit NOP instruction, but do have instructions that can be used for the purpose. In 8086 CPUs the instruction *xchg ax,ax* is used for *nop*, with *nop* being a pseudo-opcode to encode the instruction *xchg ax,ax*. Some disassemblers recognize this and will decode the *xchg ax,ax* instruction as *nop*. Similarly, IBM assemblers for System/360 and System/360 use the extended mnemonics *NOP* and *NOPR* for *BC* and *BCR* with zero masks.

Some assemblers also support simple built-in macro-instructions that generate two or more machine instructions. For instance, with some Z80 assemblers the instruction *ld hl,bc* is recognized to generate *ld l,c* followed by *ld h,b*. These are sometimes known as *pseudo-opcodes*.

Data sections

There are instructions used to define data elements to hold data and variables. They define the type of data, the length and the alignment of data. These instructions can also define whether the data is available to outside programs (programs assembled separately) or only to the program in which the data section is defined. Some assemblers classify these as pseudo-ops.

Assembly directives

Assembly directives, also called pseudo opcodes, pseudo-operations or pseudo-ops, are instructions that are executed by an assembler at assembly time, not by a CPU at run time. They can make the assembly of the program dependent on parameters input by a programmer, so that one program can be assembled different ways, perhaps for different applications. They also can be used to manipulate presentation of a program to make it easier to read and maintain.

(For example, directives would be used to reserve storage areas and optionally their initial contents.) The names of directives often start with a dot to distinguish them from machine instructions.

Symbolic assemblers let programmers associate arbitrary names (*labels* or *symbols*) with memory locations. Usually, every constant and variable is given a name so instructions can reference those locations by name, thus promoting self-documenting code. In executable code, the name of each subroutine is associated with its entry point, so any calls to a subroutine can use its name. Inside subroutines, GOTO destinations are given labels. Some assemblers support *local symbols* which are lexically distinct from normal symbols (e.g., the use of "10$" as a GOTO destination).

Some assemblers provide flexible symbol management, letting programmers manage different namespaces, automatically calculate offsets within data structures, and assign labels that refer to literal values or the result of simple computations performed by the assembler. Labels can also be used to initialize constants and variables with relocatable addresses.

Assembly languages, like most other computer languages, allow comments to be added to assembly source code that are ignored by the assembler. Good use of comments is even more important with assembly code than with higher-level languages, as the meaning and purpose of a sequence of instructions is harder to decipher from the code itself.

Wise use of these facilities can greatly simplify the problems of coding and maintaining low-level code. *Raw* assembly source code as generated by compilers or disassemblers—code without any comments, meaningful symbols, or data definitions—is quite difficult to read when changes must be made.

Macros

Many assemblers support *predefined macros*, and others support *programmer-defined* (and repeatedly re-definable) macros involving sequences of text lines in which variables and constants are embedded. This sequence of text lines may include opcodes or directives. Once a macro has been defined its name may be used in place of a mnemonic. When the assembler processes such a statement, it replaces the statement with the text lines associated with that macro, then processes them as if they existed in the source code file (including, in some assemblers, expansion of any macros existing in the replacement text).

Since macros can have 'short' names but expand to several or indeed many lines of code, they can be used to make assembly language programs appear to be far shorter, requiring fewer lines of source code, as with higher level languages. They can also be used to add higher levels of structure to assembly programs, optionally introduce embedded debugging code via parameters and other similar features.

Many assemblers have built-in (or *predefined*) macros for system calls and other special code sequences, such as the generation and storage of data realized through advanced bitwise and boolean operations used in gaming, software security, data management, and cryptography.

Macro assemblers often allow macros to take parameters. Some assemblers include quite sophisticated macro languages, incorporating such high-level language elements as optional parameters, symbolic variables, conditionals, string manipulation, and arithmetic operations, all usable during the execution of a given macro, and allowing macros to save context or exchange information. Thus a macro might generate a large number of assembly language instructions or data definitions, based on the macro arguments. This could be used to generate record-style data structures or "unrolled" loops, for example, or could generate entire algorithms based on complex parameters. An organization using assembly language that has been heavily extended using such a macro suite can be considered to be working in a higher-level language, since such programmers are not working with a computer's lowest-level conceptual elements.

Macros were used to customize large scale software systems for specific customers in the mainframe era and were also used by customer personnel to satisfy their employers' needs by making specific versions of manufacturer operating systems. This was done, for example, by systems programmers working with IBM's Conversational Monitor System / Virtual Machine (CMS/VM) and with IBM's "real time transaction processing" add-ons, CICS, Customer Information Control System, and ACP/TPF, the airline/financial system that began in the 1970s and still runs many large computer reservations systems (CRS) and credit card systems today.

It was also possible to use solely the macro processing abilities of an assembler to generate code written in completely different languages, for example, to generate a version of a program in COBOL using a pure macro assembler program containing lines of COBOL code inside assembly time operators instructing the assembler to generate arbitrary code.

This was because, as was realized in the 1960s, the concept of "macro processing" is independent of the concept of "assembly", the former being in modern terms more word processing, text processing, than generating object code. The concept of macro processing appeared, and appears, in the C programming language, which supports "preprocessor instructions" to set variables, and make conditional tests on their values. Note that unlike certain previous macro processors inside assemblers, the C preprocessor was not Turing-complete because it lacked the ability to either loop or "go to", the latter allowing programs to loop.

Despite the power of macro processing, it fell into disuse in many high level languages (a major exception being C/C++) while remaining a perennial for assemblers. This was because many programmers were rather confused by macro parameter substitution and did not disambiguate macro processing from assembly and execution.

Macro parameter substitution is strictly by name: at macro processing time, the value of a parameter is textually substituted for its name. The most famous class of bugs resulting was the use of a parameter that itself was an expression and not a simple name when the macro writer expected a name. In the macro: `foo: macro a load a*b` the intention was that the caller would provide the name of a variable, and the "global" variable or constant b would be used to multiply "a". If foo is called with the parameter `a-c`, the macro expansion of `load a-c*b` occurs. To avoid any possible ambiguity, users of macro processors can parenthesize formal parameters inside macro definitions, or callers can parenthesize the input parameters.

PL/I and C/C++ feature macros, but this facility can only manipulate text. On the other hand, homoiconic languages, such as Lisp, Prolog, and Forth, retain the power of assembly language macros because they are able to manipulate their own code as data.

Support for structured programming

Some assemblers have incorporated structured programming elements to encode execution flow. The earliest example of this approach was in the Concept-14 macro set, originally proposed by Dr. H.D. Mills (March, 1970), and implemented by Marvin Kessler at IBM's Federal Systems Division, which extended the S/360 macro assembler with IF/ELSE/ENDIF and similar control flow blocks. This was a way to reduce or eliminate the use of GOTO operations in assembly code, one of the main factors causing spaghetti code in assembly language. This approach was widely accepted in the early 80s (the latter days of large-scale assembly language use).

A curious design was A-natural, a "stream-oriented" assembler for 8080/Z80 processors from Whitesmiths Ltd. (developers of the Unix-like Idris operating system, and what was reported to be the first commercial C compiler). The language was classified as an assembler, because it worked with raw machine elements such as opcodes, registers, and memory references; but it incorporated an expression syntax to indicate execution order. Parentheses and other special symbols, along with block-oriented structured programming constructs, controlled the sequence of the generated instructions. A-natural was built as the object language of a C compiler, rather than for hand-coding, but its logical syntax won some fans.

There has been little apparent demand for more sophisticated assemblers since the decline of large-scale assembly language development. In spite of that, they are still being developed and applied in cases where resource constraints or peculiarities in the target system's architecture prevent the effective use of higher-level languages.

Use of assembly language

Historical perspective

Assembly languages were first developed in the 1950s, when they were referred to as second generation programming languages. For example, SOAP (Symbolic Optimal Assembly Program) was a 1957 assembly language for the IBM 650 computer. Assembly languages eliminated much of the error-prone and time-consuming first-generation programming needed with the earliest computers, freeing programmers from tedium such as remembering numeric codes and calculating addresses. They were once widely used for all sorts of programming. However, by the 1980s (1990s on microcomputers), their use had largely been supplanted by high-level languages, in the search for improved programming productivity. Today, although assembly language is almost always handled and generated by compilers, it is still used for direct hardware manipulation, access to specialized processor instructions, or to address critical performance issues. Typical uses are device drivers, low-level embedded systems, and real-time systems.

Historically, a large number of programs have been written entirely in assembly language. Operating systems were almost exclusively written in assembly language until the widespread acceptance of C in the 1970s and early 1980s. Many commercial applications were written in assembly language as well, including a large amount of the IBM mainframe software written by large corporations. COBOL and FORTRAN eventually displaced much of this work, although a number of large organizations retained assembly-language application infrastructures well into the 90s.

Most early microcomputers relied on hand-coded assembly language, including most operating systems and large applications. This was because these systems had severe resource constraints, imposed idiosyncratic memory and display architectures, and provided limited, buggy system services. Perhaps more important was the lack of first-class high-level language compilers suitable for microcomputer use. A psychological factor may have also played a role: the first generation of microcomputer programmers retained a hobbyist, "wires and pliers" attitude.

In a more commercial context, the biggest reasons for using assembly language were minimal bloat (size), minimal overhead, greater speed, and reliability.

Typical examples of large assembly language programs from this time are IBM PC DOS operating systems and early applications such as the spreadsheet program Lotus 1-2-3, and almost all popular games for the Atari 800 family of home computers. Even into the 1990s, most console video games were written in assembly, including most games for the Mega Drive/Genesis and the Super Nintendo Entertainment System. According to some industry insiders, the assembly language was the best computer language to use to get the best performance out of the Sega Saturn, a console that was notoriously challenging to develop and program games for . The popular arcade game NBA Jam (1993) is another example. On the Commodore 64, Amiga, Atari ST, as well as ZX Spectrum home computers, assembler has long been the primary development language. This was in large part because BASIC dialects on these systems offered insufficient execution speed, as well as insufficient facilities to take full advantage of the available hardware on these systems. Some systems, most notably Amiga, even have IDEs with highly advanced debugging and macro facilities, such as the freeware ASM-One assembler, comparable to that of Microsoft Visual Studio facilities (ASM-One predates Microsoft Visual Studio).

The Assembler for the VIC-20 was written by Don French and published by *French Silk*. At 1639 bytes in length, its author believes it is the smallest symbolic assembler ever written. The assembler supported the usual symbolic addressing and the definition of character strings or hex strings. It also allowed address expressions which could be combined with addition, subtraction, multiplication, division, logical AND, logical OR, and exponentiation operators.

Current usage

There have always been debates over the usefulness and performance of assembly language relative to high-level languages. Assembly language has specific niche uses where it is important; see below. But in general, modern optimizing compilers are claimed to render high-level languages into code that can run as fast as hand-written assembly, despite the counter-examples that can be found. The complexity of modern processors and memory sub-system makes effective optimization increasingly difficult for compilers, as well as assembly programmers. Moreover, and to the dismay of efficiency lovers, increasing processor performance has meant that most CPUs sit idle most of the time, with delays caused by predictable bottlenecks such as I/O operations and paging. This has made raw code execution speed a non-issue for many programmers.

There are some situations in which practitioners might choose to use assembly language, such as when:

- a stand-alone binary executable is required, i.e. one that must execute without recourse to the run-time components or libraries associated with a high-level language; this is perhaps the most common situation. These are embedded programs that store only a small amount of memory and the device is intended to do single purpose tasks. Such examples consist of telephones, automobile fuel and ignition systems, air-conditioning control systems, security systems, and sensors.
- interacting directly with the hardware, for example in device drivers and interrupt handlers.
- using processor-specific instructions not exploited by or available to the compiler. A common example is the bitwise rotation instruction at the core of many encryption algorithms.
- creating vectorized functions for programs in higher-level languages such as C. In the higher-level language this is sometimes aided by compiler intrinsic functions which map directly to SIMD mnemonics, but nevertheless result in a one-to-one assembly conversion specific for the given vector processor.
- extreme optimization is required, e.g., in an inner loop in a processor-intensive algorithm. Game programmers take advantage of the abilities of hardware

features in systems, enabling games to run faster. Also large scientific simulations require highly optimized algorithms, e.g. linear algebra with BLAS or discrete cosine transformation (e.g. SIMD assembly version from x264)

- a system with severe resource constraints (e.g., an embedded system) must be hand-coded to maximize the use of limited resources; but this is becoming less common as processor price decreases and performance improves.
- no high-level language exists, on a new or specialized processor, for example.
- writing real-time programs that need precise timing and responses, such as simulations, flight navigation systems, and medical equipment. For example, in a fly-by-wire system, telemetry must be interpreted and acted upon within strict time constraints. Such systems must eliminate sources of unpredictable delays, which may be created by (some) interpreted languages, automatic garbage collection, paging operations, or preemptive multitasking. However, some higher-level languages incorporate run-time components and operating system interfaces that can introduce such delays. Choosing assembly or lower-level languages for such systems gives programmers greater visibility and control over processing details.
- complete control over the environment is required, in extremely high security situations where nothing can be taken for granted.
- writing computer viruses, bootloaders, certain device drivers, or other items very close to the hardware or low-level operating system.
- writing instruction set simulators for monitoring, tracing and debugging where additional overhead is kept to a minimum
- reverse-engineering existing binaries that may or may not have originally been written in a high-level language, for example when cracking copy protection of proprietary software.
- reverse engineering and modifying video games (also termed ROM hacking), which is possible via several methods. The most widely employed is altering program code at the assembly language level.
- writing self modifying code, to which assembly language lends itself well.
- writing games and other software for graphing calculators.
- writing compiler software that generates assembly code, and the writers should therefore be expert assembly language programmers themselves.
- writing cryptographic algorithms that must always take strictly the same time to execute, preventing timing attacks.

Nevertheless, assembly language is still taught in most computer science and electronic engineering programs. Although few programmers today regularly work with assembly language as a tool, the underlying concepts remain very important. Such fundamental topics as binary arithmetic, memory allocation, stack processing, character set encoding, interrupt processing, and compiler design would be hard to study in detail without a grasp of how a computer operates at the hardware level. Since a computer's behavior is fundamentally defined by its instruction set, the logical way to learn such concepts is to study an assembly language. Most modern computers have similar instruction sets. Therefore, studying a single assembly language is sufficient to learn: I) the basic concepts; II) to recognize situations where the use of assembly language might be appropriate; and III) to see how efficient executable code can be created from high-level languages.

Typical applications

Hard-coded assembly language is typically used in a system's boot ROM (BIOS on IBM-compatible PC systems). This low-level code is used, among other things, to initialize and test the system hardware prior to booting the OS, and is stored in ROM. Once a certain level of hardware initialization has taken place, execution transfers to other code, typically written in higher level languages; but the code running immediately after power is applied is usually written in assembly language. The same is true of most boot loaders.

Many compilers render high-level languages into assembly first before fully compiling, allowing the assembly code to be viewed for debugging and optimization purposes. Relatively low-level languages, such as C, often provide special syntax to embed assembly language directly in the source code. Programs using such facilities, such as the Linux kernel, can then construct abstractions using different assembly language on each hardware platform. The system's portable code can then use these processor-specific components through a uniform interface.

Assembly language is also valuable in reverse engineering, since many programs are distributed only in machine code form, and machine code is usually easy to translate into assembly language and carefully examine in this form, but very difficult to translate into a higher-level language. Tools such as the Interactive Disassembler make extensive use of disassembly for such a purpose.

One niche that makes use of assembly language is the demoscene. Certain competitions require contestants to restrict their creations to a very small size (e.g. 256B, 1KB, 4KB or 64 KB), and assembly language is the language of choice to achieve this goal. When resources, especially CPU processing-constrained systems, like the earlier Amiga models, and the Commodore 64, are a concern, assembler coding is a must. Optimized assembler code is written "by hand" and instructions are sequenced manually by programmers in an attempt to minimize the number of CPU cycles used. The CPU constraints are so great that every CPU cycle counts. However, using such methods has enabled systems like the Commodore 64 to produce real-time 3D graphics with advanced effects, a feat which might be considered unlikely or even impossible for a system with a 1.02MHz processor.

Related terminology

- **Assembly language** or **assembler language** is commonly called **assembly**, **assembler**, **ASM**, or **symbolic machine code**. A generation of IBM mainframe programmers called it **BAL** for *Basic Assembly Language*.

  Note: Calling the language **assembler** is of course potentially confusing and ambiguous, since this is also the name of the utility program that translates assembly language statements into machine code. Some may regard this as imprecision or error. However, this usage has been common among professionals and in the literature for decades. Similarly, some early computers called their *assembler* their **assembly program**.)

- The computational step where an assembler is run, including all macro processing, is termed **assembly time**.
- The use of the word **assembly** dates from the early years of computers (*cf.* short code, speedcode).
- A **cross assembler** is functionally just an assembler. This term is used to stress that the assembler is run on a different computer than the target system, the system on which the resulting code is run. Because nowadays assemblers are written portably in a high level language like C, this is largely irrelevant. Cross assembling may be necessary if the target system lacks the capacity to run an assembler itself. This is typically the case for small embedded systems. The most important distinguishing feature of a cross assembler is that it provides for or interfaces to facilities to transport the code to the target processor, e.g. to reside in flash or EPROM. It generates a binary image, or Intel HEX file rather than an object file.
- An **assembler directive** or *pseudo-opcode* is a command given to an assembler. These directives may do anything from telling the assembler to include other source files, to telling it to allocate memory for constant data.

List of assemblers for different computer architectures

The following page has a list of different assemblers for the different computer architectures, along with any associated information for that specific assembler:

- List of assemblers

Further details

For any given personal computer, mainframe, embedded system, and game console, both past and present, at least one — possibly dozens — of assemblers have been written.

On Unix systems, the assembler is traditionally called as, although it is not a single body of code, being typically written anew for each port. A number of Unix variants use GAS.

Within processor groups, each assembler has its own dialect. Sometimes, some assemblers can read another assembler's dialect, for example, TASM can read old MASM code, but not the reverse. FASM and NASM have similar syntax, but each support different macros that could make them difficult to translate to each other. The basics are all the same, but the advanced features will differ.

Also, assembly can sometimes be portable across different operating systems on the same type of CPU. Calling conventions between operating systems often differ slightly or not at all, and with care it is possible to gain some portability in assembly language, usually by linking with a C library that does not change between operating systems. An instruction set simulator (which would ideally be written in an assembler language) can, in theory, process the object code/ binary of *any* assembler to achieve portability even across platforms (with an overhead no greater than a typical bytecode interpreter). This is essentially what microcode achieves when a hardware platform changes internally.

For example, many things in libc depend on the preprocessor to do OS-specific, C-specific things to the program before compiling. In fact, some functions and symbols are not even guaranteed to exist outside of the preprocessor. Worse, the size and field order of structs, as well as the size of certain typedefs such as off_t, are entirely unavailable in assembly language without help from a configure script, and differ even between versions of Linux, making it impossible to portably call functions in libc other than ones that only take simple integers and pointers as parameters. To address this issue, FASMLIB project provides a portable assembly library for Win32 and Linux platforms, but it is yet very incomplete.

Some higher level computer languages, such as C and Borland Pascal, support inline assembly where relatively brief sections of assembly code can be embedded into the high level language code. The Forth language commonly contains an assembler used in CODE words.

Many people use an emulator to debug their assembly-language programs.

Example listing of assembly language source code

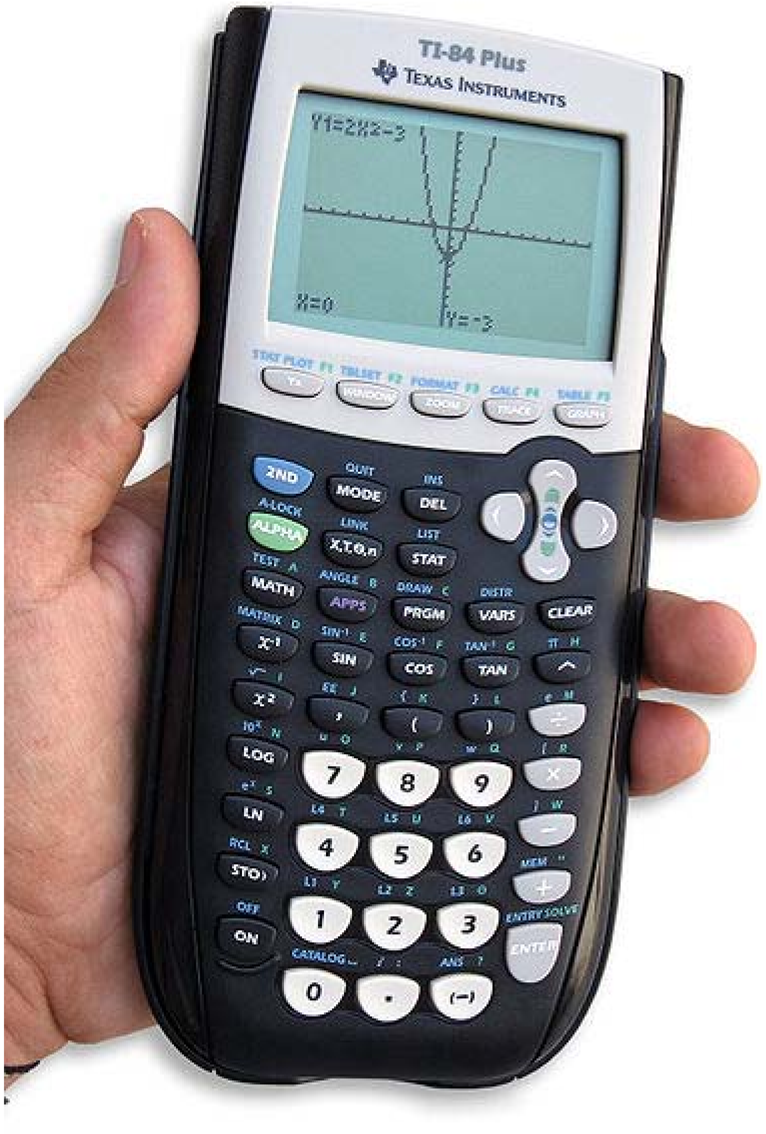
Address	Label	Instruction (AT&T syntax)	Object code
		`.begin`	
		`.org 2048`	
	`a_start`	`.equ 3000`	
2048		`ld length,%`	
2064		`be done`	00000010 10000000 00000000 00000110
2068		`addcc %r1,-4,%r1`	10000010 10000000 01111111 11111100
2072		`addcc %r1,%r2,%r4`	10001000 10000000 01000000 00000010
2076		`ld %r4,%r5`	11001010 00000001 00000000 00000000
2080		`ba loop`	00010000 10111111 11111111 11111011
2084		`addcc %r3,%r5,%r3`	10000110 10000000 11000000 00000101
2088	`done:`	`jmpl %r15+4,%r0`	10000001 11000011 11100000 00000100
2092	`length:`	`20`	00000000 00000000 00000000 00010100

```
2096   address: a_start                  00000000 00000000 00001011
                                         10111000
                .org a_start
3000   a:
```

Example of a selection of instructions (for a virtual computer) with the corresponding address in memory where each instruction will be placed. These addresses are not static, see memory management. Accompanying each instruction is the generated (by the assembler) object code that coincides with the virtual computer's architecture (or ISA).

Chapter 4

Graphing Calculator



A typical graphing calculator by Texas Instruments

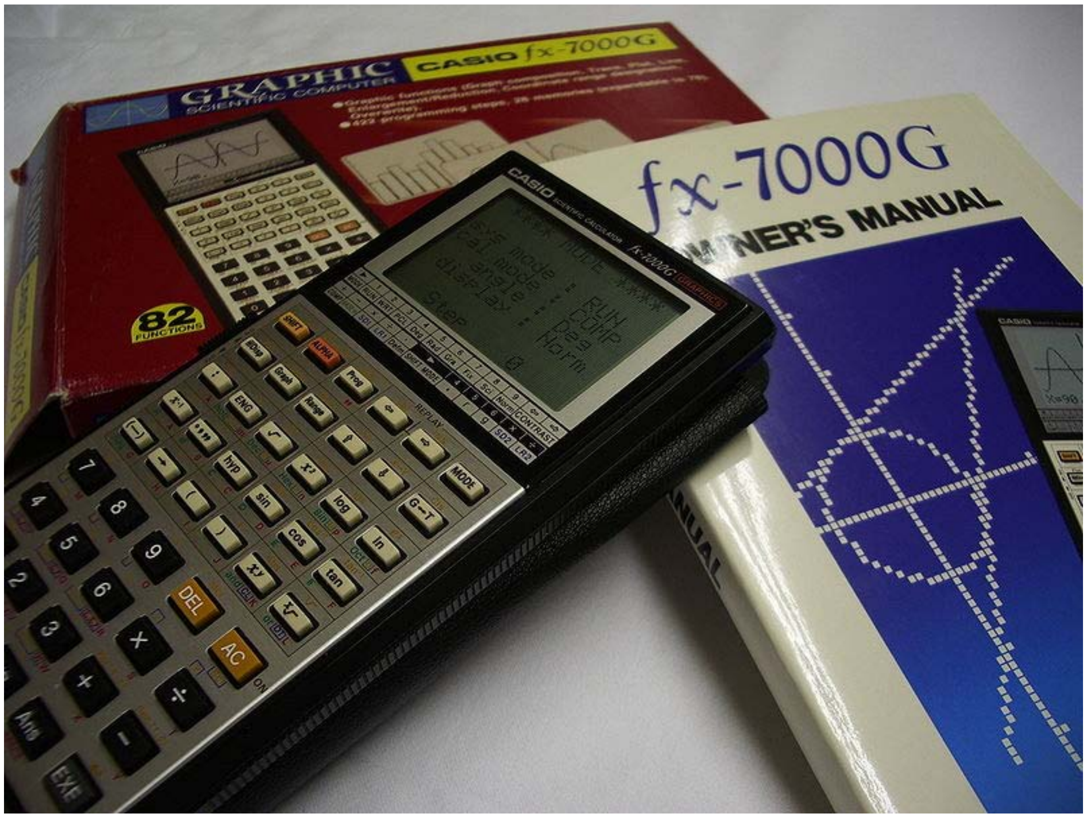
A **graphing calculator** (also **graphics calculator**) typically refers to a class of handheld calculators that are capable of plotting graphs, solving simultaneous equations, and performing numerous other tasks with variables. Most popular graphing calculators are also programmable, allowing the user to create customized programs, typically for scientific/engineering and education applications. Due to their large displays intended for graphing, they can also accommodate several lines of text and calculations at a time. Some graphing calculators also have colour displays, and others even include 3D graphing.

Many graphing calculators can be attached to devices like electronic thermometers, pH gauges, weather instruments, decibel and light meters, accelerometers, and other sensors and therefore function as data loggers.

Since graphing calculators are readily user-programmable, such calculators are also widely used for gaming purposes, with a sizable body of user-created game software on most popular platforms.

There is also computer software available to emulate or perform the functions of a graphing calculator. One such example is Grapher for Mac OS X and is a basic software graphics calculator.

History



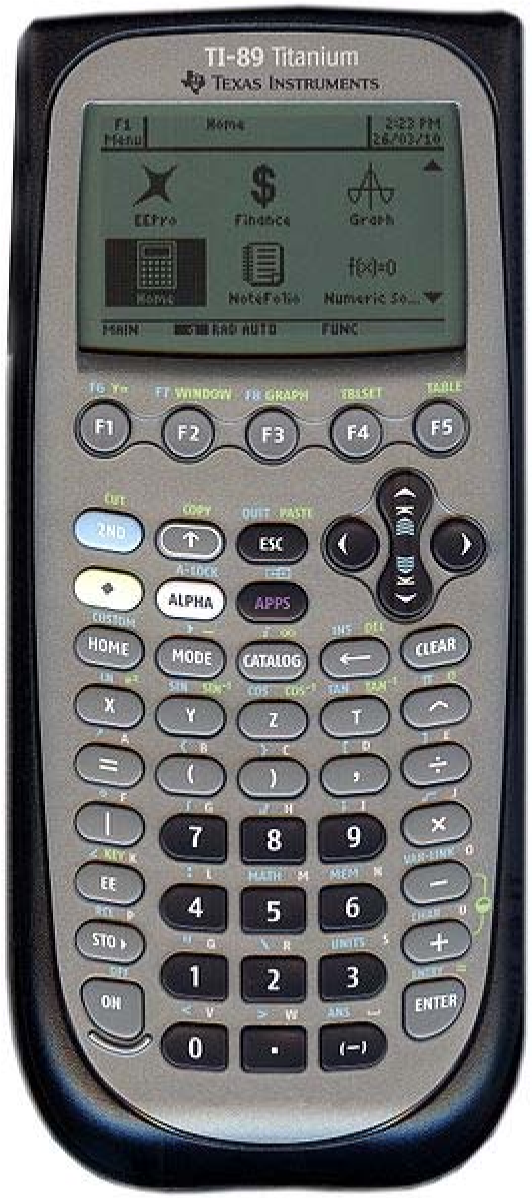
Casio fx-7000G; The world's first graphing calculator

Casio produced the world's first graphics calculator, the fx-7000G, in 1985. After Casio, Hewlett Packard followed shortly in the form of the HP-28C. This was followed by the HP-28S (1988), HP-48SX (1990), HP-48S (1991), and many other models. Recent models like the current HP 50g (2006), feature a computer algebra system (CAS) capable of manipulating symbolic expressions and analytic solving. The HP-28 and -48 range were primarily meant for the professional science/engineering markets; the HP-38/39/40 were sold in the high school/college educational market; while the HP-49 series cater to both educational and professional customers of all levels. The HP series of graphing calculators is best known for its Reverse Polish notation interface, although the HP-49 introduced a standard expression entry interface as well.

Texas Instruments has produced graphing calculators since 1990, the oldest of which was the TI-81. Some of the newer calculators are similar, with the addition of more memory, faster processors, and USB connection such as the TI-82, TI-83 series, and TI-84 series. Other models, designed to be appropriate for students 10–14 years of age, are the TI-80 and TI-73. Other TI graphing calculators have been designed to be appropriate for calculus, namely the TI-85, TI-86, TI-89 series, and TI-92 series (TI-92, TI-92 Plus, and Voyage 200). TI offers a CAS on the TI-89, TI-Nspire CAS and TI-92 series models with the TI-92 series featuring a QWERTY keypad. TI calculators are targeted specifically to the educational market, but are also widely available to the general public.

Graphing calculators are also manufactured by Sharp but they do not have the online communities, user-websites and collections of programs like the other brands.

Graphing calculators in schools



TI-89 Titanium, capable of doing Symbolic Manipulation, Computer Algebra System (CAS)

- **North America** – high school mathematics teachers allow and even encourage their students to use graphing calculators in class. In some cases (especially in calculus courses) they are *required*. Some of them are banned in certain classes such as chemistry or physics due to their capacity to contain full periodic tables.
- **United Kingdom** – a graphics calculator is required for most A-level maths courses, the use of such devices is both taught and tested. However, for GCSE maths exams, a limited number of calculator models are allowed, none of which are capable of graphic operations (although they are capable of scientific and statistical operations).

- **Finland and Slovenia** – and certain other countries, it is forbidden to use calculators with symbolic calculation (CAS) or 3D graphics features in the matriculation exam.
- **Norway** – calculators with wireless communication capabilities, such as IR links, have been banned at some technical universities.
- **College Board of the United States** – permits the use of most graphing or CAS calculators that do not have a QWERTY-style keyboard for parts of its AP and SAT exams, but IB schools do not permit the use of calculators with computer algebra systems on its exams.
- **Australia** – policies vary from state to state.
- **Victoria, Australia** – the VCE specifies approved calculators as applicable for its mathematics exams. For Further Mathematics an approved graphics calculator (for example TI-83/84, Casio 9860, HP-39G) or CAS (for example TI-89, Classpad 300, HP-40G) can be used. Mathematical Methods and Mathematical Methods CAS have a common technology free examination consisting of short answer and some extended answer questions. They also each have a technology assumed access examination consisting of extended response and multiple choice questions: a graphics calculator is the assumed technology for Mathematical Methods and a CAS for Mathematical Methods CAS. These two exams have substantial material in common but also some distinctive questions. Specialist Mathematics has a technology free examination and a technology assumed access examination where either an approved graphics calculator or CAS may be used. Calculator memories are not required to be cleared. In subjects like Physics and Chemistry, students are only allowed a standard scientific calculator.
- **Western Australia** – all tertiary entrance examinations in Mathematics involve a calculator section which assume the student has a graphics calculator; CAS enabled calculators are also permitted. In subjects such as Physics, Chemistry and Accounting only non programmable calculators are permitted.
- **New South Wales** – graphics calculators are allowed for the General Mathematics Higher School Certificate exam, but disallowed in the higher level Mathematics courses.
- **New Zealand** – calculators identified as having high-level algebraic manipulation capability are prohibited in NCEA examinations unless specifically allowed by a standard or subject prescription. This includes calculators such as the TI-89 series .
- **Turkey** – any type of calculator whatsoever is prohibited in all primary and high schools except the IB and American schools.
- **Singapore** – graphing calculators are used in junior colleges; it is required in the Mathematics paper of the GCE 'A' Levels, and most schools use the TI-84 Plus or TI-84 Plus Silver Edition.
- **Netherlands** – high school students are obliged to use graphing calculators during tests and exams in their final three years. Most students use the TI-83 Plus or TI-84 Plus, but other graphing calculators are allowed, including the Casio CFX-9860G and HP-39G.

Programming

Most graphing calculators, as well as some non-graphing scientific and programmer's calculators (e.g. the Radio Shack PC-7 and other machines in that series, which use a full-blown Basic variant) can be programmed to automate complex and frequently used series of calculations and those inaccessible from the keyboard.

The actual programming can often be done on a computer then later uploaded to the calculators. The most common tools for this include the PC link cable and software for the given calculator, configurable text editors like TextPad, and specialised programming tools such as the below-mentioned implementation of various languages on the computer side and Hackman, a hex editor which also includes syntax highlighting and other tools for assembly programming on more than a dozen processor types as well as a disassembler, macro and scripting facilities, programmers' calculator application, source code management and code snippet library functions and the ability to edit various types of media like clusters and File Allocation Tables on a disc.

Earlier calculators stored programmes on magnetic cards and the like; increased memory capacity has made storage on the calculator the most common implementation. More of the newer machines also can use memory cards as well.

Many calculators, such as earlier TI graphing and scientific calculators. will tokenise the code for a programme or function, using ISO 8859 type character codes for the statements and other programming elements. The TI-92 Plus and many HP calculators read the code much like computers do and they have functions such as Chr$, Chr, Char, Asc, and the like in Basic (sometimes renamed) in addition to using somewhat modified or unmodified versions of 7-bit, 8-bit or 9-bit ISO 8859-derived character sets and other character sets running from of values of 0 to 127 (07F hex), 255 (0FF hex), or 511 (1FF hex) -- and many of them have a tool similar to the Character Map on Windows.

The official sites of the manufacturers and of other people like professors & teachers, students, statisticians, scientists, and organisations like university business and computer science departments, SourceForge, and the 27. November Spreadsheet Macro Programming Club are also useful. A broad array of third-party software including 3-D function graphing tools, web browsers, chat, email and NNTP clients, telnet/SSH, spreadsheets, word processors, sound & graphics tools, network tools, and programming tools can be located on the internet.

A cable and/or IrDA transceiver connecting the calculator to a computer make the process easier and expands other possibilities such as on-board spreadsheet, database, graphics, and word processing programmes. The second option is being able to code the programmes on board the calculator itself. This option is facilitated by the inclusion of full-screen text editors and other programming tools in the default feature set of the calculator or as optional items. Some calculators have QWERTY keyboards and others can be attached to an external keyboard which can be close to the size of a regular 102-

key computer keyboard. Programming is a major use for the software and cables used to connect calculators to computers, other calculators &c.

The most common programming languages used for calculators are similar to keystroke-macro languages and variants of Basic. The latter can have a large feature set -- approaching that of Basic as found in computers -- including character and string manipulation, advanced conditional and branching statements, sound, graphics, and more including, of course, the huge spectrum of mathematical, string, bit-manipulation, number base, I/O, and graphics functions built into the machine.

Languages for programming calculators fall into all of the main groups, i.e. machine code, low-level, mid-level, high-level languages for systems and application programming, scripting, macro, and glue languages, procedural, functional, imperative &. Object-Oriented Programming can be achieved in some cases.

Nearly all calculators capable to being connected to a computer can be programmed in assembly language and machine code. The most common assembly and machine languages are for the purpose-designed, TMS9900, Zilog Z-80, and various Motorola chips (e.g. the modified 68000) which serve as the Central Processing Units of the machines. At least one machine in development may have a conventional 80*86 series, RISC, or purpose-built Intel chip. All of the above chips are modified to some extent from their use elsewhere. Some manufacturers do not document and even mildly discourage the assembly language programming of their machines because they must programmed in this way by putting together the programme on the PC and then forcing it into the calculator by various improvised methods, but more current models are set up for greater ease of assembly programming and more official documentation is available.

Other on-board programming languages include purpose-made languages, variants of Eiffel, Forth, and Lisp, and Command Script facilities which are similar in function to batch/shell programming and other glue languages on computers but generally not as full featured. Ports of other languages like BBC Basic and development of on-board interpreters for Fortran, Rexx, Awk, Perl, Unix shells (ksh, sh, bash, csh, zsh, tcsh &c.), other shells (DOS/Win95, OS/2, and WinNT/2000 shells as well as the related 4Dos, 4NT and 4OS2 as well as DCL), Cobol, C, Python, Tcl, Pascal, Delphi, ALGOL, and other languages are at various levels of development.

Some calculators, especially those with other PDA-like functions have actual operating systems including the TI proprietary OS for its more recent machines, MS-DOS, Windows CE, and rarely Windows NT 4.0 Embedded et seq, and Linux. Experiments with the TI-89, 92, 92+ and Voyager machines show the possibility of installing some variants of other systems such as a chopped-down variant of CP/M, an operating system which has been used for portable devices in the past.

Tools which allow for programming the calculators in C/C++ and possibly Fortran and assembly language are used on the computer side, such as HPgcc, TIgcc and others. Flash memory is another means of conveyance of information to and from the calculator.

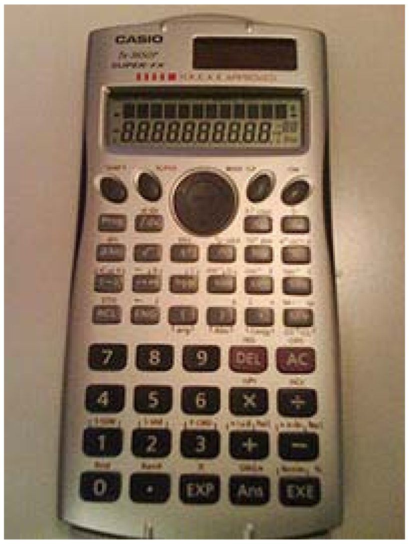
The on-board Basic variants in TI graphing calculators and the languages available on HP 48 type calculators can be used for rapid prototyping by developers, professors, and students, often when a computer is not close at hand.

Most graphing calculators have on-board spreadsheets which usually integrate with Microsoft Excel on the computer side. At this time, spreadsheets with macro and other automation facilities on the calculator side are not on the market. In some cases, the list, matrix, and data grid facilities can be combined with the native programming language of the calculator to have the effect of a macro and scripting enabled spreadsheet.

Chapter 5

Casio fx-3650P

Casio fx-3650P



Type	Scientific calculator
Manufacturer	Casio
Introduced	2002
	Calculator
Entry mode	D.A.L.
Precision	±1 at the 10th d.p.
Display Type	2 line display 1st line-Dot Matrix

	2nd line-LCD
Programming	
Programming language(s)	Keystroke
Other	
Power consumption	Solar cell and G13 Type (LR44) button battery two way power

Casio fx-3650P is a programmable scientific calculator manufactured by Casio Computer Co., Ltd. It can store 12 digits for the mantissa and 2 digits for the exponent together with the expression each time when the "EXE" button is pressed. Also, the calculator can use the previous result to do calculations by pressing "Ans".

Modes

The calculator is available in 6 modes:

- Basic arithmetic calculations
- Complex number calculations
- Standard deviation calculations
- Regression calculations
- Base-*n* calculations
- Programs

Basic arithmetic calculations

- Arithmetic calculations
- Fraction operations
    - Fraction calculations
    - Decimal↔fraction conversions
    - Mixed fraction↔improper fraction conversions
- Percentage calculations
- Degrees, minutes, seconds calculations
- Rounding (Must be used with *Fix* decimal display mode)
- Trigonometric functions
- Hyperbolic function
- Logarithm
- Natural logarithm
- Antilogarithm
- Differential calculus

- Integral calculus

Complex number calculations

In this mode, if the result have both real and imaginary part, an "R↔I" symbol will appear at the top-right corner.

- Absolute value and argument calculations
- Rectangular↔polar form display
- Conjugate calculations

Standard deviation calculations

This mode is for statistical calculation. For some input data, sum of squares of values (Σx^2), sum of values (Σx), number of data (n), sample standard deviation (xσn-1) and population standard deviation (xσn) can be calculated.

Regression calculations

This mode is for statistical calculation and can be divided further into

- Linear regression: y=A+Bx
- Logarithmic regression: y=A+B*ln x
- Exponential regression: ln y=ln A+Bx
- Power regression: $y=A+x^B$
- Inverse regression: y=A+B/x
- Quadratic regression: $y=A+Bx+Cx^2$

For some input ordered pairs, one of the below can be calculated. (The availability differs from modes.)

Σx^2, Σx, n, Σy^2, Σy, Σxy, $\bar{x}$, xσn, xσn-1, $\bar{y}$, yσn, yσn-1, Regression coefficient A, Regression coefficient B, Correlation coefficient r, $\hat{x}$, $\hat{y}$, Σx^3, $\Sigma x^2 y$, Σx^4, Regression coefficient C, $\hat{x}_1$and $\hat{x}_2$

Program

The calculator can hold up to 4 programs with total capacity of 360 bt Program commands:

- ? - Operator input command, used when user's input is required. Usually used with →(*variable*)
- → - Assign to variable command, to assign the value before it to the variable after it. Always used as (*value*)→(*variable*).
- : - Multi-statement separator, separate program statements.
- ◢ - Output command, output the value.

- $\Rightarrow$- Conditional jump, jump when conditions are met.
- = - Relational operator
- ≠ - Relational operator
- > - Relational operator
- ≧- Relational operator.
- Goto - Unconditional jump, jump to label.
- Lbl - Label, jump destination.

Conditional jumps

Conditional jumps should be used in the syntax:
condition$\Rightarrow$*statement 1*:*statement 2*
When *condition* is true, statement 1 is executed, then statement 2 is executed. If *condition* is false, statement 1 is skipped and statement 2 is executed.

E.g.: ...A=0$\Rightarrow$A+1→B:C+5→D:... If A=0, both A+1→B and C+5→D is executed. If A≠0, only C+5→D is executed.

Unconditional jumps

Unconditional jumps uses Goto and Lbl to operate.

When Goto *n* (where n is an integer in 0-9) is executed, the program will jump to Lbl *n*. Loops can be created with unconditional jumps.

System check

The calculator will perform system check when shift, 7 and ON is pressed together. The system check have 3 parts.

Part 1: light check - All lights will be on when the system check is initiated, press shift then all lights will be off. Press shift to proceed. The next 2 screens have similar functions.

Part 2: display check - The dot matrix screen will display "24   PRG13" and the LCD screen will display "0000000000 ^{00}". Press shift to proceed. Then the display become "BBBBBBBBBB" and "1111111111 ^{11}". Then "CCCCCCCCCC" and "222222222 ^{22}" and so on up till "JJJJJJJJJJ""9999999999 ^{99}". Press shift to proceed to part 3.

Part 3: key check - Press shift, the LCD will display 1. Press alpha, up, down, left, right in order. The display will increase by 1 each time you press a button. Next, press MODE, prog, $\int$dx, x^{-1}, x^3, and so on up till Ans. When EXE is pressed, the display become "24   OK""13"(The display differs from version, the above is for version 4 of the calculator which is the latest version). Press ON to end the system check.

Glitch

Warning: Using the following glitch may reset the calculator.

There is a glitch in the calculator.

Enter 1M+:1÷0 as a program. 1M+:sin^{-1}2 is also OK. Refer to the following table in this section.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	1	2	3	4	5	6	7	8	9
1	.	E				Reset All			Mode clear	=
2		%	%+	%-	M+	M-	→A	→B	→C	→D
3	Σx^2	Σx	n	Σy^2	Σy	Σxy	Σx^3	$\Sigma x^2 y$	Σx^4	→D
4	r	$\bar{x}$	xσn	xσn-1	$\bar{y}$	yσn	yσn-1	A	B	C
5	G	T	M	K	-1	0	1	2	A	B
6						Fix (col)				
7						Sci (col)				
8	10	e	log	ln	$\sqrt{}$	$\sqrt[3]{}$	-	(	arg	Abs
9	sin	$\sin^{-1}$	sinh	$\sinh^{-2}$	cos	$\cos^{-1}$	cosh	$\cosh^{-1}$	tan	$\tan^{-1}$

Note that (col) means the column header. For example Fix (col) at column 0 is Fix 0.

The table is the code table for the glitch. For example, the code 85 refers to A and 76 refers to Fix 7.

1. Run the program
2. It will display "Math ERROR". press AC.
3. Enter a number and press EXE
4. Press up and some characters will display.

To translate the entered number to the characters:

1. Put the left-most digit after 0 and refer to the table. That's the first character
2. For the next 9 digits, group the numbers in two. Refer to the table and that shows the next 5 characters.(Note: If the number of digits is odd, a 0 is added next to the last digit)
3. Then add some zeros. The number of zeros = 6-(number of characters generated by the above step).
4. Lastly, Refer to the table for (number of total digits)-1, if it is a single-digit number, put a 0 BEFORE it. That's the last digit.

Chapter 6

Casio FX-502P Series and Casio FX-602P Series

Casio FX-502P series

CASIO FX-501P / FX-502P

A 30 year old Casio FX-501P in working condition displaying the number π

Type	Programmable Scientific
Manufacturer	Casio
Introduced	1978

	Calculator
Entry mode	Infix
Precision	12 digits mantissa, ±99 exponent
Display Type	LCD Seven-segment display
Display Size	10 + 3 Digits
	Programming
Programming language(s)	Keystroke (fully merged, Turing complete)
Memory Register	11 (FX-501P) 22 (FX-502P)
Program Steps	128 (FX-501P) 256 (FX-502P)
	Interfaces
Ports	one vendor specific
	Other
Power supply	2×"G13" or 2x"LR44"
Power consumption	0.0008W
Weight	141g, 5 oz
Dimensions	15.24x7,6x1.2 cm, 6"×3"×½"

The **FX-501P** and **FX-502P** were programmable calculators, manufactured by CASIO from 1978. They were the predecessors of the Casio FX-601P and Casio FX-602P.

Arithmetic

The FX-502P series use the algebraic logic as was state of the art at the time.

Display

The **FX-501P** and **FX-502P** featured a single line 7-segment liquid crystal display with 10 digits as main display. An additional 3 digits 7-segment display used to display exponents and program steps when entering or debugging programs and 10 status indicators. The display was covered with a yellow filter, supposedly to prevent ultra-violet radiation damage.

Programming

The programming model employed was key stroke programming by which each key pressed was recorded and later played back. On record multiple key presses where merged into in a single programming step. All operations fitted into one program step.

The **FX-501P** could store 128 steps, with 11 memory registers. The **FX-502P** had double capacity with 256 steps and 22 memory registers.

Conditional and Unconditional jumps as well as subroutines where supported. The FX-502P series supported 10 labels for programs and subroutines called P0 .. P9. Each program or subroutine could have up to 10 local labels called LBL0 .. LBL9 for jumps and branches.

The **FX-501P** and **FX-502P** supported indirect addressing both for memory access and jumps and therefore the programming model could be considered Turing complete.

Since the **FX-501P** and **FX-502P** only employed a Seven-segment display each program step was represented by a special 2 digit codes made op of the digits **0** .. **9** and the character **C**, **E**, **F** and **P**. The Calculator came with a special overlay so the user need not memorise the mapping between code and actual command.

Programming example

Here is a sample program that computes the factorial of an integer number from 2 to 69. For 5!, you'll type `5 P0` and get the result, 120. The whole program is only 9 bytes long.

```
Key-code     Display-code  Comment

P0           P0            You'll call the program with the P0 key
Min 0        C6 00         stores the value in register M0
1            01            starts with 1
LBL 0        F0 00         label for the loop
*            E1            multiply
MR 0         C7 00         by n
DSZ GOTO 0   FF 01 F1 00   decrements  register M0 and back to LBL0
until M0=0
=            E5            end of loop, the machine has calculated
1*n*(n-1)*...2*1=n!
```

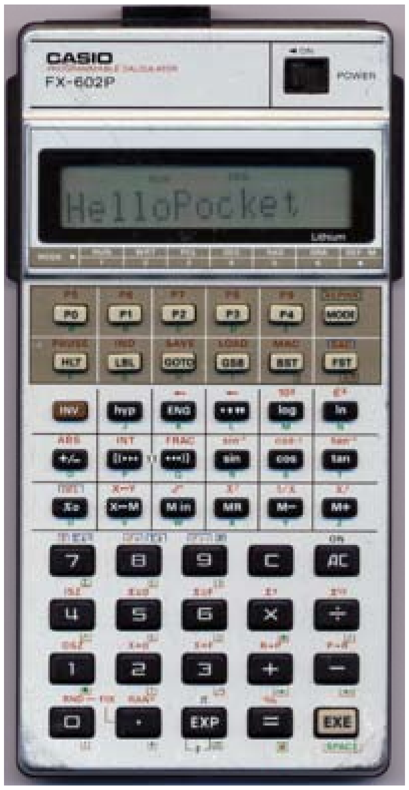
This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59

Interface

The FX-501P and FX-502P used the FA-1 to store program and data to Compact Cassette using the Kansas City standard. The FA-1 also enabled the calculators to generate musical notes.

Casio FX-602P series

CASIO FX-602P series



A 28 year old FX-602P in working condition

Type	Programmable Scientific
Manufacturer	Casio
Introduced	1981
Discontinued	1989
Calculator	
Entry mode	Infix
Precision	12 digits mantissa, ±99 exponent
Display Type	LCD Dot-matrix
Display Size	11 Characters + 3 digits

	Programming
Programming language(s)	Keystroke (fully merged, Turing complete)
Memory Register	11 (FX-601P) 22 .. 88 (FX-602P)
Program Steps	128 (FX-601P) 32 .. 512 (FX-602P)
	Interfaces
Ports	one vendor specific
	Other
Power supply	2×"CR-2032" Lithium
Weight	141g, 5 oz
Dimensions	15.24x7,6x1.2 cm, 6"×3"×½"

The **FX-601P** and **FX-602P** were programmable calculators, manufactured by CASIO from 1981. It was the successor model to the Casio FX-502P series and was itself succeeded in 1990 by the Casio FX-603P.

Display

The **FX-601P series** featured a single line dot matrix display with 11 characters as main display. An additional 3 digits 7-segment display used to display exponents as well as program steps when entering or debugging programs. There were 11 status indicators.

Programming

The programming model employed *key stroke* programming by which each key pressed was recorded and later played back. On record, multiple key presses where merged into in a single programming step. Only a few operations needed two bytes. Synthetic programming was possible but not very common

The **FX-601P** could store 128 fully merged steps and data could be stored in 11 memory register. The memory of the **FX-602P** could be partitioned between from 32 to 512 fully merged steps and data could be stored in 22 to 88 memory register. The default set-up was 22 register and 512 steps. From there one could trade 8 steps for one additional register or 80 steps for 11 register with the 11th register begin a so called "F" register.

Like its predecessor the FX-602P series supported 10 labels for programs and subroutines called P0 .. P9. Each program or subroutine could have up to 10 local labels called LBL0 .. LBL9 for jumps and branches.

The **FX-601P** and **FX-602P** supported indirect addressing both for memory access and jumps and therefore programming model could be considered Turing complete.

Both the FX-601P and FX-602P could load and execute programs from the predecessors.

Programming example

This program computes the factorial of an integer number from 2 to 69. For 5!, the user enters `5 P0` to produce the result, 120. The program occupies 9 bytes of memory.

```
Key-code        Comment
P0              Call the program with the P0 key
Min00           stores the value in register 0
1               starts with 1
LBL0            label for the loop
*               multiply
MR00            by n
DSZ GOTO0       décrements M00 and back to LBL0 until M00=0
=               end of loop, the machine has calculated 1*n*(n-
1)*...2*1=n!
```

This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59

Interface

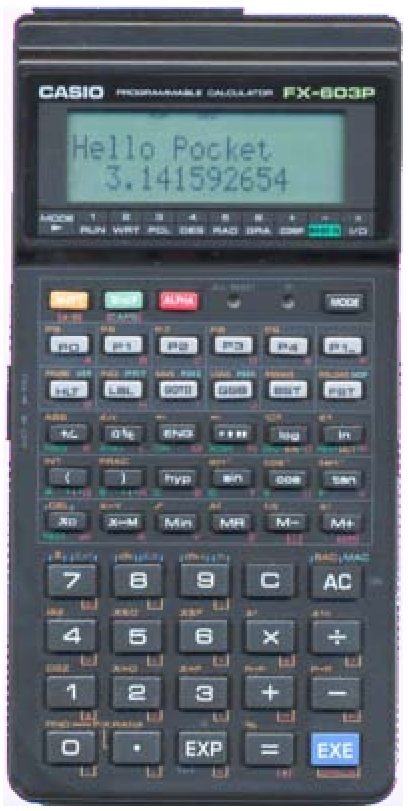
The FX-601P and FX-602P used the same FA-1 interface as used by the FX-502P line of calculators or alternatively the newer FA-2 interface which was also used by Casio FX-702P. Both interfaces featured a Kansas City standard Compact Cassette interface. The FA-2 featured an additional printer port for the FP-10 Thermal printer.

Chapter 7

Casio FX-603P and Elektronika B3-34

Casio FX-603P

CASIO FX-603P



A FX-603P in working condition

Type	Programmable Scientific
Manufacturer	Casio

Introduced	1990
	Calculator
Entry mode	Infix
Precision	12 digits mantissa, ±99 exponent
Display Type	LCD Dot-matrix
Display Size	2×16 Character
	Programming
Programming language(s)	Keystroke (fully merged, Turing complete)
Memory Register	110
Program Steps	6'144
	Interfaces
Ports	one vendor specific
	Other
Power supply	2×"CR-2032" Lithium + 1×"CR-2032" Lithium
Weight	141g, 5 oz
Dimensions	15.24x7,6x1.2 cm, 6"×3"×½"

The **FX-603P** was a programmable calculator, manufactured by CASIO from 1990. It was the successor model to the Casio FX-602P.

Display

The FX-603P featured a two line dot matrix display with 16 characters each as main display. An additional 4 digits 7-segment display used to display the program step when entering or debugging programs and 20 status indicators.

Programming

The programming model employed was key stroke programming by which each key pressed was recorded and later played back. On record multiple key presses where merged into in a single programming step. There were only a very few operations which needed two bytes.

The FX-603P could store 6'144 steps. Data could be stored in 110 memory register. The FX-603P series supported 20 labels for programs and subroutines called P0 .. P19 - twice the amount of the predecessor models. Each program or subroutine could have up to 10 local labels called LBL0 .. LBL9 for jumps and branches.

The **FX-603P** supported indirect addressing both for memory access and jumps and therefore programming model could be considered Turing complete.

The FX-603P was upward compatible to the FX-602P could load FX-602P programs from Compact Cassette.

Programming example

Here is a sample program that computes the factorial of an integer number from 2 to 69. For 5!, you'll type `5 P0` and get the result, 120. With the additional alpha output the program is 14 byte long.

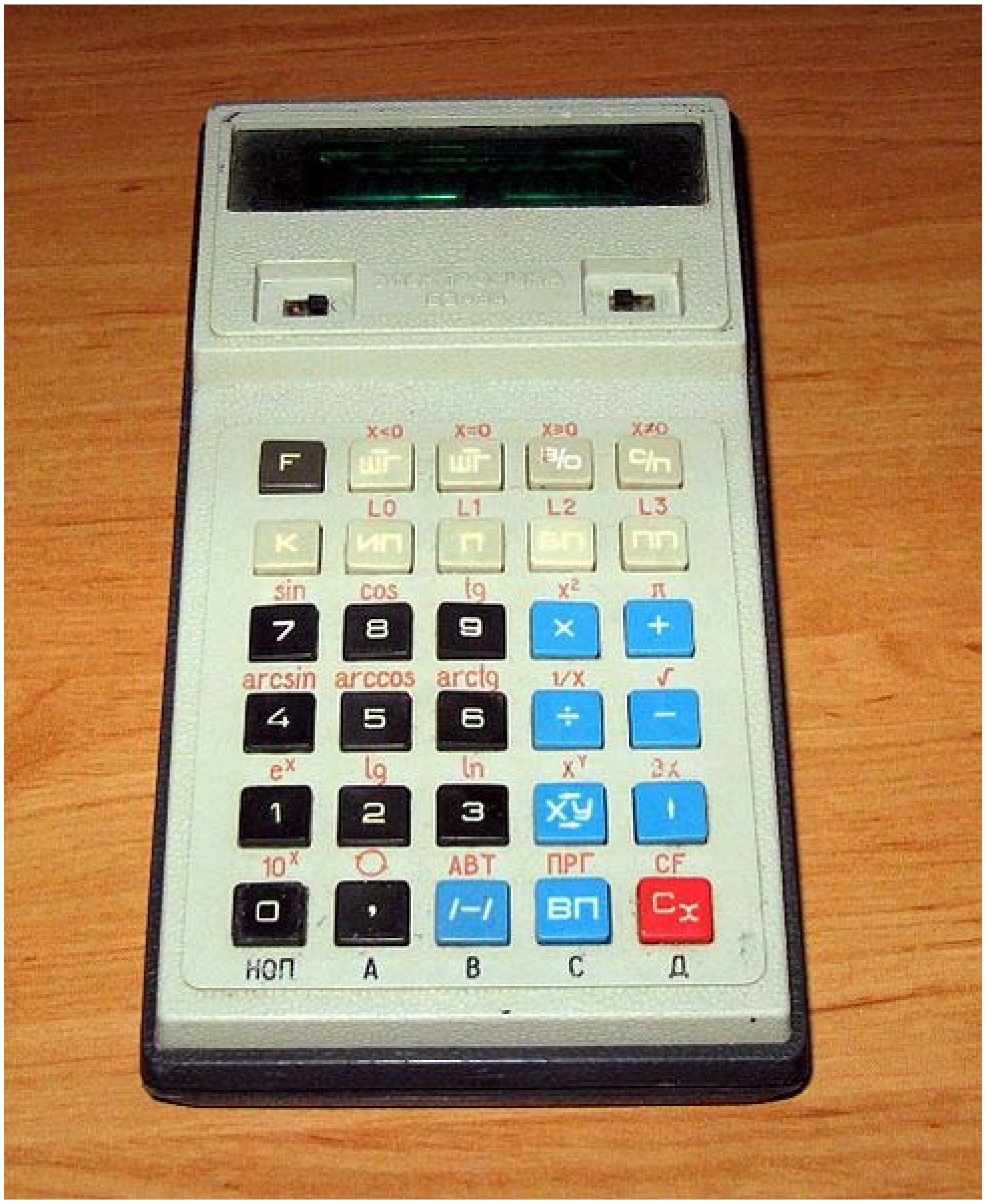
```
Key-code         Comment

P0              You'll call the program with the P0 key
"#!="           output "n!=" in first row of display. Result will be
shown in 2nd line
Min00           stores the value in register 0
1               starts with 1
LBL0            label for the loop
*               multiply
MR00            by n
DSZ GOTO0       décrements M00 and back to LBL0 until M00=0
=               end of loop, the machine has calculated 1*n*(n-
1)*...2*1=n!
```

This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59

Interface

The FX-603P used the same FA-6 interface as used by the Casio FX-850P / Casio FX-880P line of calculators. This interface features a Kansas City standard Compact Cassette interface, a Centronics printer port and a RS-232 interface.

Elektronika B3-34



An Elektronika B3-34.

Elektronika B3-34 was a very popular Soviet programmable calculator. It was released in 1980 and was sold for 85 rubles.

B3-34 used Reverse Polish notation and had 98 bytes of instruction memory, 4 stack user registers and 14 addressable registers. Each register could store up to 8 mantissa digits and two exponent digits in the range from 1e-99 to 1e+99.

The first Soviet programmable stationary calculator ISKRA 123, powered by the power grid, was released at the beginning of the 1970s. The first programmable battery-powered pocket calculator Elektronika "B3-21" was developed by the end of 1977 and released at the beginning of 1978. Its successor, B3-34, wasn't backward compatible with B3-21. The instruction set, hardware architecture and the microcode of the B3-34 defined the standard of the later Soviet programmable hand-held and office-deck calculators: MK-61, MK-52, MK-54, MK-56.

Later, at the end of 1980s, much more powerful calculators appeared on the Soviet market. For example, the calculator or hand-held computer MK-90, which had a graphic LCD display and an internal BASIC interpreter, was essentially a pocket-sized variety of the PDP-11. Due to their high price and the growing popularity of much more powerful personal computers, such as ZX Spectrum, these powerful calculators never gain popularity among the general Soviet population. Therefore, the B3-34-derived calculators are remembered by many as their "first computer".

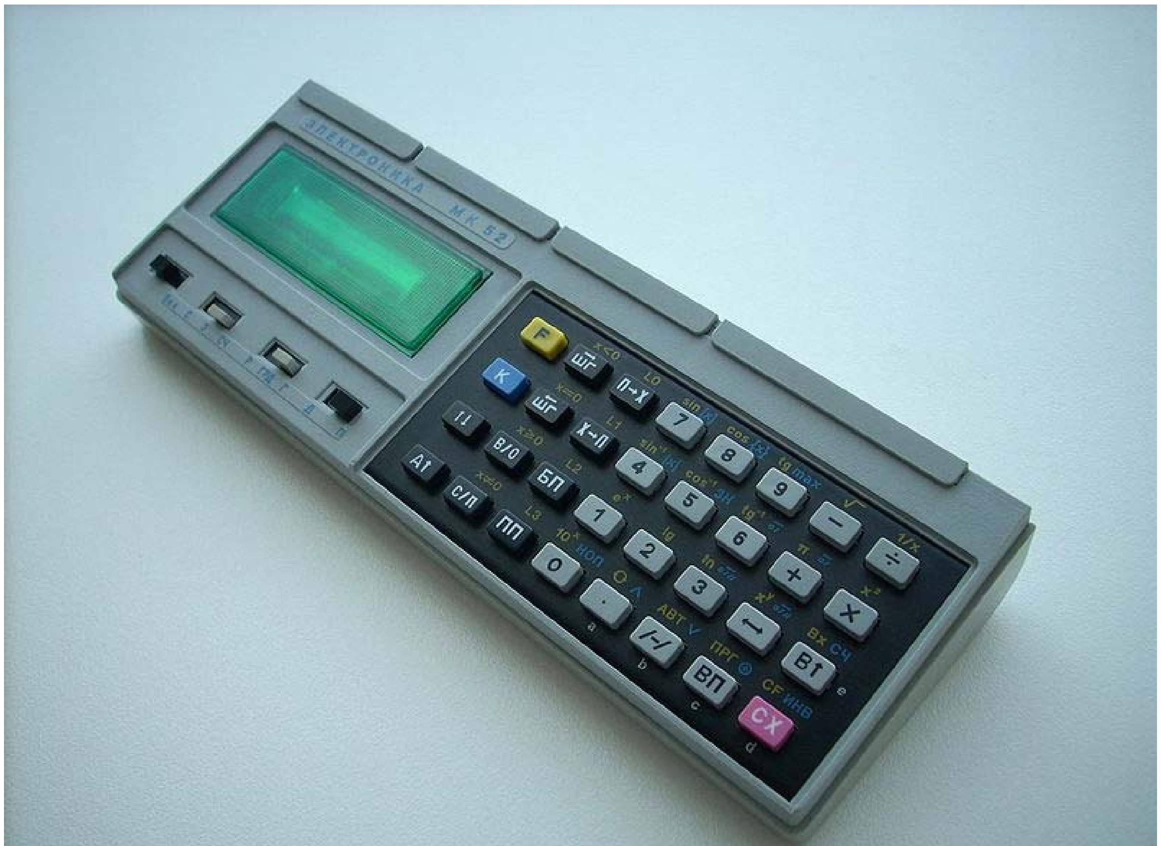
Despite very limited capability, people managed to write all kinds of programs for B3-34 and its later successors, including adventure games and libraries of sophisticated calculus-related functions for engineers. Hundreds, perhaps thousands, of programs were written for these machines, from practical scientific and business software, which were used in real-life offices and labs, to fun games for children. The Elektronika MK-52 calculator (using the extended B3-34 command set, and featuring internal EEPROM memory for storing programs and external interface for EEPROM cards and other periphery) was used in soviet spacecraft program (for Soyuz TM-7 flight) as a backup of the board computer.

This series of calculators was also noted for a large number of highly counter-intuitive mysterious undocumented features, not unlike the "synthetic programming" of the American HP-41, which were exploited by applying normal arithmetic operations to error messages, jumping to non-existent addresses and other techniques. A clever step away from the documented path would often cause some highly unusual things. For example, operations over the hexadecimal number 0xF, which looked like a decimal point on the dark screen, could cause a number of bizarre effects, from complete freeze to self-modification of the program, temporary appearance of otherwise invisible undocumented registers and sometimes totally non-deterministic behavior. A number of respected monthly publications, including the popular science magazine "Nauka i Zhizn" ("Science and Life"), featured special columns, dedicated to optimization techniques for calculator programmers and updates on undocumented features for hackers, which grew into a whole esoteric science with many branches, known as "eggogology" ("еггогология"). The error messages on those calculators were intended to appear as a the English word "Error", which to the Russians looked like a meaningless "ЕГГОГ" (*EGGOG*). B3-34 and its derivatives helped many Soviet programmers to develop their skills, because programming and debugging required ability to read and write machine code and optimize literally every byte of the program. The microcode of those calculators remains only partially published and some of their "dark secrets" are still a mystery and are still being researched by some enthusiasts.

Like the HP-41 series in the Western countries, the B3-34 and its successors became a legend among some Soviet programmers and computer hobbyists. A numbers of websites provide the source of hundreds of programs for these calculators, technical documentation, lists of undocumented features and amazing stories about them. Some Soviet hackers managed to modify B3-34 into digital multimeters, control interfaces for model railroads, added tape storage devices and other peripherals. Modern Russian calculators MK-161 and MK-152, designed and manufactured in Novosibirsk since 2007, are partially backward compatible with B3-34 and are also based on Reverse Polish notation. However, they are only compatible on function level and don't reproduce the original undocumented features.

Chapter 8

Elektronika MK-52



The **Elektronika MK-52** (Russian: Электро́ника МК-52) is RPN-programmable calculator which was manufactured in the Soviet Union during the years 1983 to 1992.

The functionality of the MK-52 is identical to that of the MK-61, except the MK-52 has an internal non-volatile EEPROM memory module, for permanent data storage, diagnostic slot, and slot for ROM modules. Programming language and functionality of MK-52 and MK-61 are extensions of the MK-54, the B3-34 and B3-21 Elektronika calculators. It is the only known calculator to have internal storage in the form of an EEPROM module. All Soviet calculators are renowned for having a very large number of undocumented functions.

The MK-52 has 105 steps of volatile program memory, an internal EEPROM module (with 512 bytes of memory) and 15 memory registers. It functions using either four AA-size battery cells or a wall plug. It has a relatively dim, ten-digit (8 digit mantissa, 2 digit exponent) green vacuum fluorescent display. The MK-52 has an expansion port to which various ROM (Read-only memory) modules may be attached. Its system clock speed is approximately 455 kHz (derived from a ceramic resonator), its weight is approximately 0.4 kilograms and its original selling price was 115 Roubles.

The MK-52 was used as a backup to the onboard computers of the Soyuz spacecraft on the Soyuz TM-7 mission to the Mir space station.

Basic operations

Should it be required, one can refer to the Russian to English translation of the MK-52's keyboard.

Note that throughout this page, square brackets represent actual keys, for example, [+] represents an 'addition' key.

The MK-52 has two main operating modes; 'automatic mode' and 'programming mode'. General calculations and operations are performed in automatic mode; programs are input in programming mode. To switch between modes, one must press [F] [CHS] (looks like [/-/]) to switch to automatic mode and one must press [F] [EE] (looks like [Bn]) to switch to programming mode.

Basic operations in automatic mode are conducted in accordance with RPN (Reverse Polish Notation) logic. For example, to evaluate 2+3, the following keystrokes are required: [enter] (looks like [B^]) [+].

Programming

In simple programming, commands are typed into the MK-52 in programming mode and are then executed in order.

In programming mode, the screen displays information about the program in memory. For example, if '10 01 0E 03' is displayed, then this means that '0E' is stored at program step '00', '01' is stored at program step '01', '10' is stored at program step '02' and the machine is currently prompting for data to be input for program step '03'. Individual program operations are represented by two-digit operation codes in programming mode.

Saving to EEPROM

Note that before entering a program to volatile memory with the intention of saving this program to EEPROM memory, the EEPROM program space to be saved to must be cleared first, as performing the clearing operation clears the volatile memory as well as the selected area of the EEPROM memory.

Each program step requires 1 byte of memory and each register requires 7 bytes of memory.

When clearing, reading or writing to the EEPROM memory, the 'address' and 'range' are specified in the form of a six-digit number, preceded by a non-zero number (which is ignored) in automatic mode, i.e. '1aaaadd' means 'dd' bytes, starting at memory address 'aaaa'. A two-position data/program switch controls whether data (from the registers) or program memory is transferred; a three-position switch is used to select read, write and clear operations.

Example of operation

This example demonstrates the entry of a program (which simply adds 1 to the input number and displays the result) and the saving and loading of this program to/from the EEPROM module.

Step 1: Clear the memory

The program will be four steps long (as will be explained in step 2) and, hence, requires 4 bytes of EEPROM. The memory is cleared using the following procedure:

Switch the clear/write/read switch to 'clear' and ensure that the data/program switch is set to 'program'.

In automatic mode, enter '1000004' (4 bytes starting at address '0000').

Press [Addr] (a key that looks like [A^] in the bottom left).

Press [R/W] (a key that looks like [^v] above the bottom left key).

Step 2: Enter the program

Switch the clear/write/read switch to 'write'.

Enter programming mode.

Enter program. For the one described above, the following procedure may be used:

- initially the screen should display " 00", prompting for input.

[enter] (looks like [B^])

[+]

- the screen should now display "10 01 0E 03".

[R/S] (looks like [C/n])

- the screen should now display "50 10 01 04".

The program should now be in volatile memory.

Note that the program contains four steps. Each program step requires 1 byte of memory, hence the '04' at the end of the '1000004' command.

Enter automatic mode.

One may now press [RTN] (looks like [B/0]) to return to the start of the program.

One may now enter a number, then press [R/S] to run the program. The result (the input number plus 1) should then be displayed.

Step 3: Write to memory

Switch the clear/write/read switch to 'write'.

If it is not still displayed, then enter '1000004' in automatic mode.

Press [Addr] (a key that looks like [A^] in the bottom left).

Press [R/W] (a key that looks like [^v] above the bottom left key).

The program should now be written to the EEPROM module. You may now power off the machine in the knowledge that the program is stored safely.

Step 4: Read from memory.

Power on the machine.

Switch the clear/write/read switch to 'read'.

Enter '1000004' in automatic mode.

Press [Addr] (a key that looks like [A^] in the bottom left).

Press [R/W] (a key that looks like [^v] above the bottom left key).

The stored program should now be transferred to volatile memory and ready for use.

Similar procedures may be used to read and write register data (set the two position switch to data for these procedures).

Additional information

Bitwise/binary operations:

The MK-52 is fully capable of performing binary number operations. The following example demonstrates the OR logical operation between the binary numbers '111000' and '100001':

First, the numbers are made into groups of four digits, adding leading zeros if necessary, i.e. making '111000' into groups of four gives '0011' and '1000'.

The equivalent decimal values of each of these four-digit binary numbers are '3' and '8', which gives a hexadecimal number of '38', equivalent to the binary number '111000'. Similarly, '100001' is equivalent to '21' in hexadecimal.

Binary numbers are input into the machine as hexadecimal numbers prepended by an '8.'.

So, the numbers '8.38' and '8.21' are entered into the MK-52 and the OR operation is performed on them. The OR operation is achieved by pressing [K], then [CHS] (which looks like [/-/]).

The result displayed should be '8.39'. This translates to the two binary number groups '0011' and 1001 and, hence, the binary number '111001', which is the result of the OR operation performed on the two binary numbers '111000' and '100001'.

The following list details the MK-52's graphical representation of hexadecimal numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, L, C, T, E, (blank). Normal hexadecimal representation is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E.

Games:

There are a host of games available for the MK-52 (as can be found from one link below). The MK-52's undocumented functions tend to be heavily used in the various games of the machine due to their use in producing unusual calculations and specialised displays. A simple example of the modification of the display may be observed by the repeated squaring of, say, 1 x 10^50 (ignoring error messages).

Colours:

The MK-52 was available in a variety of colours. Known colours are: black/grey, turquoise/blue, white/grey and orange.

Schematics:

In what would be considered an unusual practice today (but was common for Soviet electronics), technical schematics were provided for the MK-52 when it was purchased, prompting user modification and repair of the machine.

EGGOG:

To the amusement of many, when an error is encountered on the machine, the display produces a message similar to the English 'Error'. The word, written in this fashion, cuts down on the number of display segments used to display the error message. The result is that, in Russian, this spelling is not pronounced 'error', but 'eggog'.

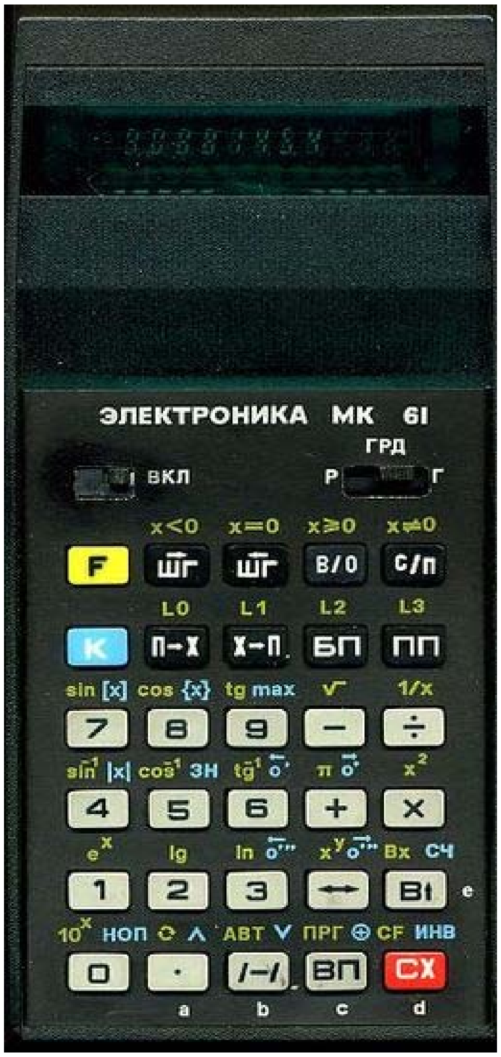
Known bugs/errors

There is currently only one known bug in the MK-52. That bug is that the MAX function gives a result of zero if one of the two arguments of the function is zero.

Chapter 9

Elektronika MK-61 and HP 35s

Elektronika MK-61



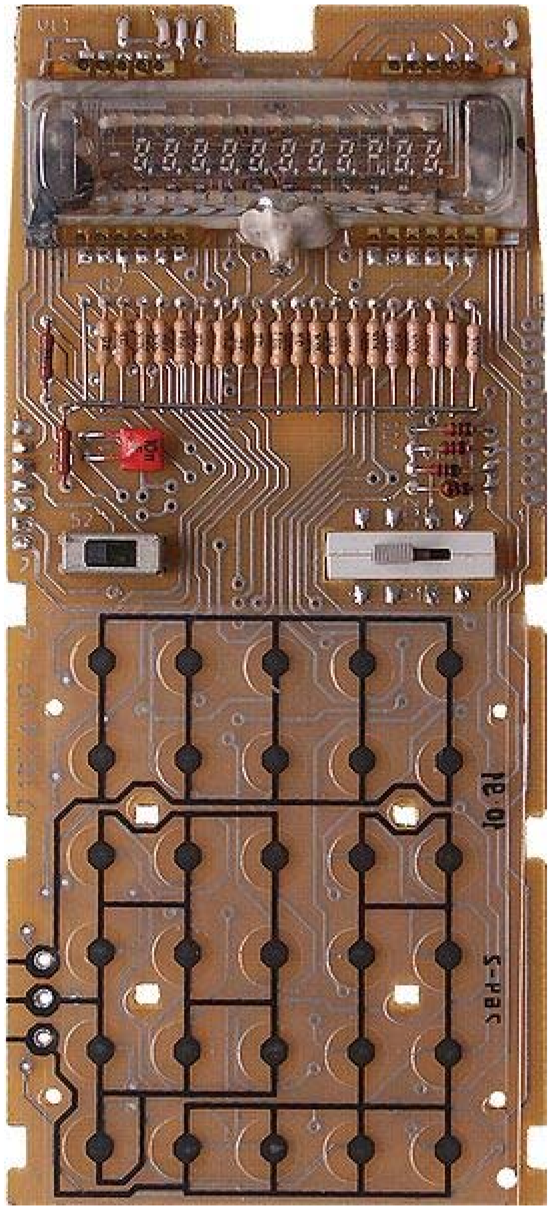
Elektronika MK-61

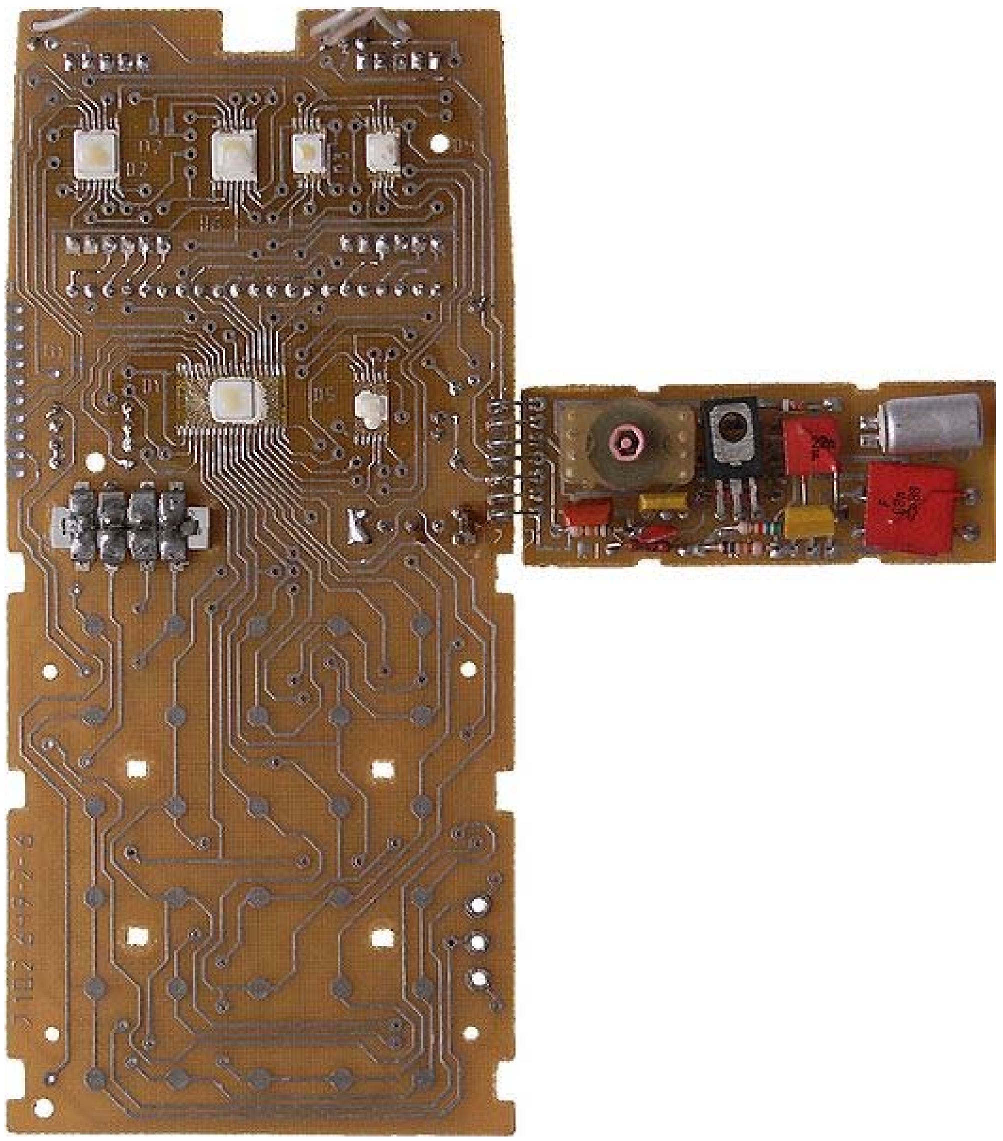
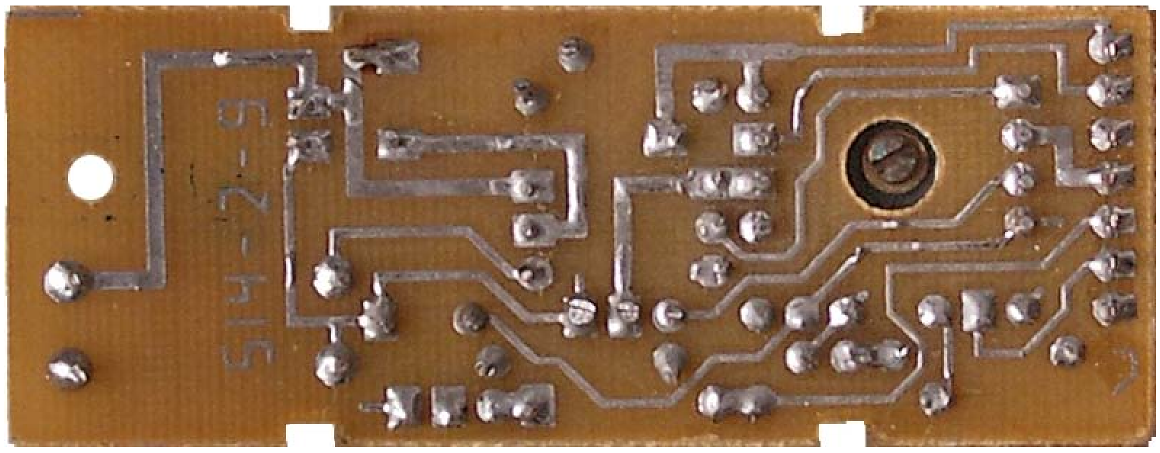
The **Elektronika MK-61** (Russian: Электро́ника МК-61) is a third-generation non-BASIC, RPN programmable calculator which was manufactured in the Soviet Union during the years 1983 to 1991. Its original selling price was 85 rubles.

The functionality of the MK-61 is identical to that of the MK-52, except the MK-52 has an internal non-volatile EEPROM memory module, for permanent data storage and also has the capability of using external EEPROM modules. The MK-61 is functionally very similar to the MK-54, the B3-34 and B3-21 Elektronika calculators, all of which are renowned for having a very large number of undocumented functions.

The MK-61 has 105 steps of volatile program memory and 15 memory registers. It functions using either three AA-size battery cells or a wall plug. It has a ten-digit (eight digit mantissa, two digit exponent) green vacuum fluorescent display.

The MK-61 was in production from 1985 to 1993.



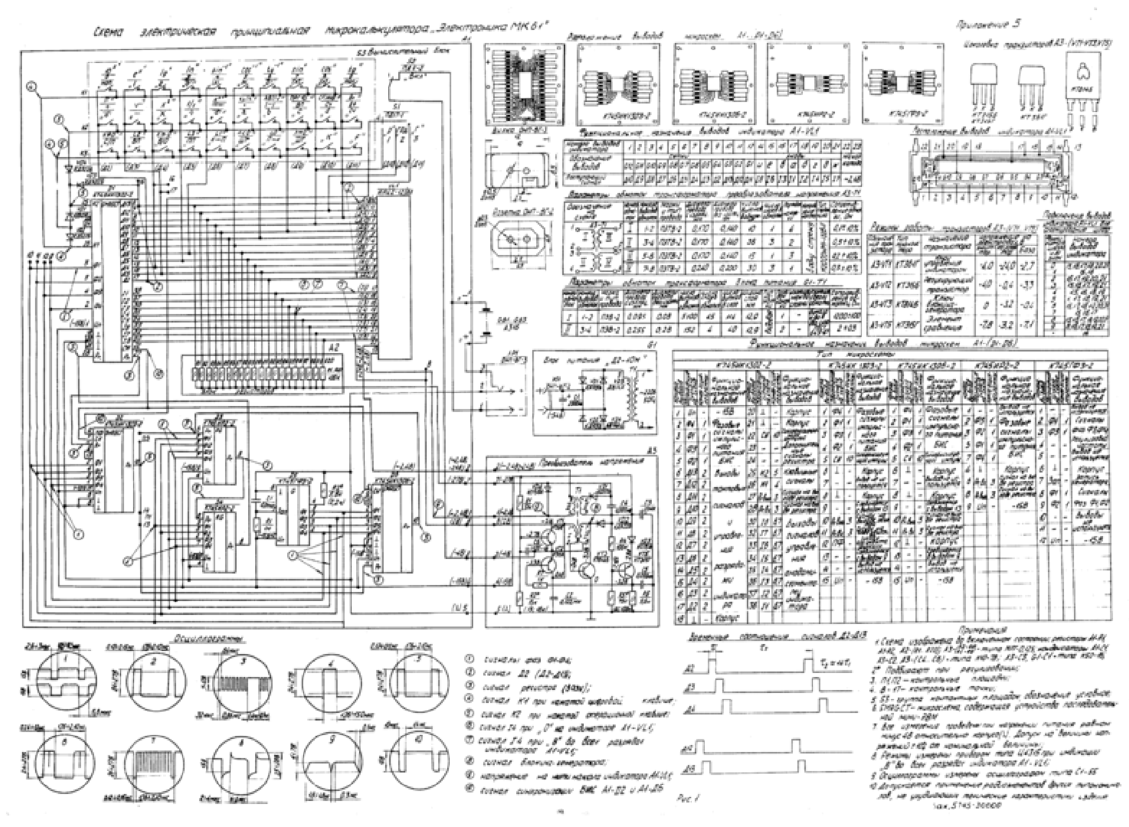
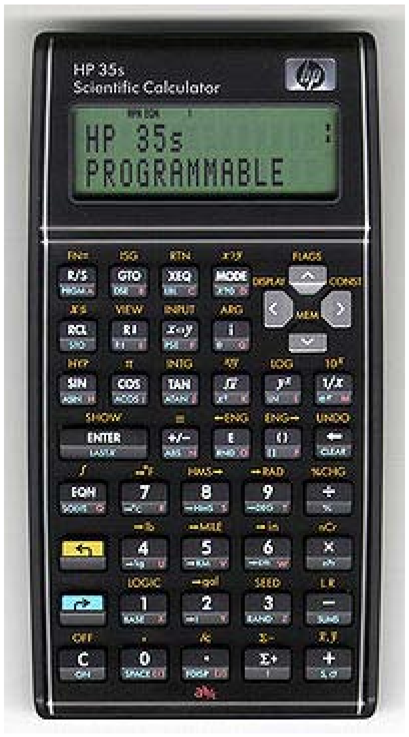


Схема электрическая принципиальная микрокалькулятора „Электроника МК 61"
Приложение 5
Осциллограммы
Примечания
Рис. 1

HP 35s

HP-35s


Front view of the HP-35s

Type	Programmable Scientific
Manufacturer	Hewlett-Packard
Introduced	2007
Design firm	Kinpo Electronics, Inc
Calculator	
Entry mode	RPN, Infix
Precision	15 digits ±499 exponent (internal)
Display Type	LCD Dot-matrix
Display Size	2 × 14 Character
CPU	
Processor	Sunplus Technology 8502
Programming	
Programming language(s)	RPN Keystroke (fully merged, Turing complete)
User Memory	30KB

Memory Register	26 .. 800
	Other
Power supply	2 × CR2032 batteries
Weight	125 g (4.4 oz)
Dimensions	15.8 × 8.2 × 1.82 cm (6.22 × 3.23 × 0.72 in)

The **HP 35s Scientific Calculator** is, as of 2007, the latest in **Hewlett-Packard's** long line of non-graphing scientific and programmable calculators. Although it is a successor to the HP 33s, it was introduced to commemorate the 35th anniversary of the HP-35, Hewlett-Packard's first pocket calculator (and the world's first pocket scientific calculator).

The HP 35s uses either Reverse Polish Notation (RPN) or Infix notation as input.

Other features of the HP 35s include:

- Two-line alphanumeric LCD display
- 26 memory registers
- Scientific and statistical functions
- Operation in decimal, binary, octal, hexadecimal
- Equation solver (a feature first seen on the HP-34C)
- Numerical integration (also first seen on the HP-34C)
- Support for input and display of fractions
- Complex number and vector calculations
- Unit conversions and table of physical constants
- Approximately 30 kilobytes of memory for programs and/or data

Although the HP 35s has far more functions, processing power, and memory than the original HP-35 which it commemorates, Hewlett-Packard has attempted to give the HP 35s the look of the original HP-35 and other HP calculators of that era. It also features the sloped-front keys for which HP calculators (although not the original HP-35) are well known.

The physical appearance and keyboard layout of the HP 35s is very different than that of its immediate predecessor, the HP 33s, but the two calculators are functionally almost identical. The primary differences are:

- The HP 35s allows both label and line number addressing in programs. The HP 33s had only label addressing. With only 26 labels, it was difficult to write programs making use of the entire 30KB of memory.
- The memory in the HP 35s is also usable for data storage, in the form of up to 801 numbered memory registers.
- Support for vector operations is new in the HP 35s.

- Indirect branching, which allows the contents of a memory register to be used as the target of a branching instruction (GTO or XEQ) is available in the HP 33s, but was omitted from the HP 35s.

Design and Manufacture

The HP 35s was designed by Hewlett-Packard in conjunction with Kinpo Electronics, Inc. The latter company manufactures this calculator for HP. The present implementation is mechanically well built and sturdy, with much the same key feel as classic HP calculators and keys that are rated for over a million keypresses.

Programming

Unlike the original HP-35, the HP 35s is keystroke programmable, meaning that it can remember and later execute sequences of keystrokes to solve particular problems of interest to the user. These keystroke programs, in addition to performing any operation normally available on the keyboard, can also make use of conditional and unconditional branching and looping instructions, allowing programs to perform repetitive operations and make decisions.

Even with indirect jumping removed, the HP 35s still supports indirect addressing, with which it is still possible to implement a Universal Turing machine; and therefore the programming model of the HP 35s can be considered Turing complete.

As in normal operation, programming can be done in either in RPN or in D.A.L.

RPN mode programs are usually smaller and faster, while D.A.L. is better suited for interactive use and easier to use for those who are not used to RPN .

Programming example

Here is a sample program that computes the factorial of an integer number from 2 to 69. There are two versions of the example: one for algebraic mode (LBL A) and one for RPN mode (LBL R). Even in this small example the algebraic version is 10 bytes (40%) larger.

```
Step  Op-code       Comment

A001  LBL A         Algebraic version started with XEQ A ENTER
A002  ALG           Switch to algebraic mode
A003  INPUT N       Read N from keyboard
A004  1►F           Store 1 in F
A005  N*F►F         Store N*F in F
A006  DSE N         decrements N and
A007  GOTO A005     back to step A005 until N=0
A008  VIEW F        View Result
A009  STOP          End of Program
                    LN=35 CK=3B3F
```

```
R001  LBL R         RPN version started with XEQ R ENTER
R002  RPN           Switch to RPN mode
R003  STO I         Store x (stack) into I (memory)
R004  1             starts with 1
R005  RCL* I        recall I  (memory) and multiply with x (stack)
R006  DSE I         decrements I and
R007  GOTO R005     back to R005 until I=0
R008  STOP          end of program - result is in x (stack)
                    LN=25 CK=A8A1
```

This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59

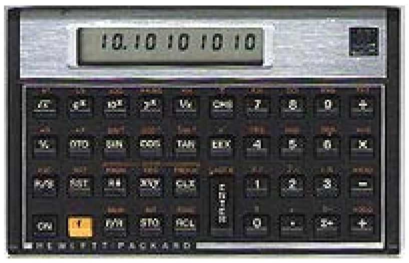
Firmware Bugs

The HP-35s ROMs contain a number of bugs. In some cases the accuracy of the trigonometric functions is less than 7 digits out of the claimed 15 digit precision. In addition, some programs give wildly incorrect answers or lock up the calculator requiring a hard reset that destroys all stored data and programs. Unlike the original HP-35, which had software bugs that were quickly fixed with users being offered a free replacement calculator, the HP-35S software bugs - many of which existed in the earlier HP-33S model - have not been fixed as of 2011.

Chapter 10

HP-10C Series

HP-10C



HP-10c

Type	Programmable Scientific
Manufacturer	Hewlett-Packard
Introduced	1982
Discontinued	1984
Cost	$80
Calculator	
Entry mode	RPN
Display Type	LCD Seven-segment display
Display Size	10 Digits
CPU	
Processor	Voyager
Programming	
Programming language(s)	RPN key stroke (fully merged)
Memory Register	0 … 9
Program Steps	9 … 79

Other	
Power consumption	0.25mW

HP-11C

HP-11c

Type	Programmable Scientific
Manufacturer	HP
Introduced	1981
Discontinued	1989
Cost	$135
Calculator	
Entry mode	RPN
Display Type	LCD Seven-segment display
Display Size	10 Digits
CPU	
Processor	Voyager
Programming	
Programming language(s)	RPN key stroke (fully merged)
Memory Register	0 … 20
Program Steps	63 … 203
Other	
Power consumption	0.25mW

HP-12C

HP-12C

Type	Programmable Financial
Manufacturer	HP
Introduced	1981
Discontinued	present
Cost	$135
Calculator	
Entry mode	RPN
Display Type	LCD Seven-segment display
Display Size	10 Digits
CPU	
Processor	Voyager / ARM
Programming	
Programming language(s)	RPN key stroke (fully merged)
Memory Register	0 … 20
Program Steps	63 … 203 8 … 400 (Platinum)
Other	
Power consumption	0.25mW

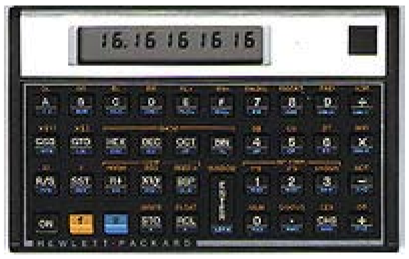
HP-15C

HP-15c

Type	Programmable Scientific
Manufacturer	HP
Introduced	1982
Discontinued	1989
Cost	$135

Calculator	
Entry mode	RPN
Display Type	LCD Seven-segment display
Display Size	10 Digits
CPU	
Processor	Voyager
Programming	
Programming language(s)	RPN key stroke (fully merged)
Memory Register	0 … 67
Program Steps	0 … 448
Other	
Power consumption	0.25mW

HP-16C



HP-16C

Type	Programmable Computer Science
Manufacturer	HP
Introduced	1982
Discontinued	1989
Cost	$135
Calculator	
Entry mode	RPN
Display Type	LCD Seven-segment display
Display Size	10 Digits
CPU	
Processor	Voyager
Programming	

Programming language(s)	RPN key stroke (fully merged)
Memory Register	0 … 20
Program Steps	63 … 203
Other	
Power consumption	0.25mW

The **HP-10C series** calculators were introduced by Hewlett-Packard in 1981. Also known as the "Voyager" series, all are programmable, use Reverse Polish Notation, and feature continuous memory. Nearly identical in appearance, each model provided different capabilities and was aimed at different user markets.

The HP calculators 10C series consisted of five models (with original retail price and years of production):

- HP-10C – basic scientific calculator. ($80 1982-1984)
- HP-11C – mid-range scientific calculator. ($135 1981-1989)
- HP-12C – business/financial calculator. ($150 1981-present)
- HP-15C – advanced scientific calculator. ($135 1982-1989)
- HP-16C – computer programmer's calculator. ($150 1982-1989)

The HP-12C remains in widespread use today.

HP-10C

The **HP-10C** is the last and lowest-featured calculator in this line, even though its number would suggest an earlier origin. The 10C was a basic scientific programmable. While a useful general purpose RPN calculator, the HP-11C offered twice as much for only a slight increase in price. Designed to be an introductory calculator, it was still costly compared to the competition, and many looking at an HP would just step up to the better HP-11C. Poor sales led to a very short market life.

HP-11C

The **HP-11C** is a mid-range scientific programmable.

HP-12C

The **HP-12C** is a popular financial calculator. It was such a successful model that Hewlett-Packard redesigned it from scratch, added several new functions, and introduced it as the **HP-12C Platinum** in 2003.

The HP-12C is HP's longest and best-selling product, in continual production since its introduction in 1981. Due to its simple operation for key financial calculations, the calculator long ago became the de facto standard among financial professionals – for example, most investment banks issue HP-12Cs to the members of each incoming class of its investment banking analysts and associates. Its popularity has endured despite the fact that even a simple, but iterative, process such as amortizing the interest over the life of a loan—a calculation which modern spreadsheets can complete almost instantly—can take over a minute with the HP-12C. The 1977 October edition of the HP Journal contains an article by Roy Martin, the inventor of the simple method of operation used in HP financial calculators, which describes, in detail, the mathematics and functionality built by Prof William Kahan (from UC Berkeley) and Roy Martin that is still in use today.

Later HP financial calculators are many times as fast with more functions, but none has been as successful. The HP-12C's programming mode is very intuitive and works like a macro operation on a computer. Basically, the keys one would press in the calculating mode to arrive at a solution are entered in the programming mode along with logical operators (*if*, *and*, etc.) applicable to the solution. After the programming is complete, the macro will run in computation mode to save the user steps and improve accuracy. There are 99 lines of programmable memory on the HP-12C, and 400 lines on the HP-12C Platinum.

Over its lifespan, the processor's technology has been redesigned to integrate all the circuitry into a single chip and to refresh the manufacturing process (as the foundry could no longer manufacture the necessary chips, having moved on to making higher-density chips). However, HP's market research found in the late 1980s that the users did not trust results obtained too quickly and so the CPU speed was never improved from the original 200 or so kHz. In the late 1990s, the CPU was changed to a 3V process and the battery was changed to a single 3V cell.

In 2008, HP modified the design so that new production runs contain an ARM processor which runs an emulated version of previous chips. This has brought advanced possibilities such as flashing new firmware, not previously possible. HP also released a software development kit (SDK), making it possible to make new and custom operating systems. The calculator runs 20 times faster on most benchmark operations. This version is colloquially known as the HP-12C+ although HP does not market it as a different product.

The HP-12C is one of only four calculators permissible in the Chartered Financial Analyst exams, the others being its sister, the HP-12C Platinum, and the Texas Instruments BA II Plus and BA II Plus Professional.

HP-12C Platinum

The HP-12C Platinum is a revision to the successful 12C. The 12C Platinum is visibly distinguished by its silver-colored upper half as opposed to the gold-colored plate on the original 12C. The Platinum has a faster processor, larger memory and more built-in functions. It allows input to be entered in algebraic mode as well as RPN mode. There are two versions of the HP-12C Platinum. The early version did not have parentheses, which often led to awkward key sequences to solve problems in algebraic mode. Newer versions of the HP-12C Platinum have parentheses, on the blue-shifted functions of the STO and RCL keys.

In 2006, Hewlett Packard released a limited edition of the 12C Platinum to commemorate the 25th anniversary of the original 12C introduction. The 25th anniversary model has the parentheses feature.

Hewlett Packard makes a HP-12C / HP-12C Platinum solutions book available as a PDF on their website.

HP-15C

The **HP-15C** is a high-end scientific programmable with a root-solver and numerical integration. It is able to handle complex numbers and matrix operations. Although out of production, its popularity has led to high prices of US$200–400 on the used market and a petition asking HP to restart production. The HP-15C was a replacement for the (LED Display based) HP-34C.

HP-16C

The **HP-16C** is a computer programmer's calculator, designed to assist in debugging. It can display numbers in hexadecimal, decimal, octal, and binary and convert numbers from one base to the other. To accommodate long binary numbers, the display can be 'windowed' by shifting it left and right. For consistency with the computer the programmer is working with, the word size can be set to different values from 1 to 64 bits. Binary-arithmetic operations can be performed as unsigned, 1's complement, or 2's complement operations. This allows the calculator to emulate the programmer's computer. A number of specialized functions are provided to assist the programmer, including left- and right shifting, masking, and bitwise logical operations. Sales were poor.

Arithmetic

One of the least-known features of this calculator series is the quality of the arithmetic inside them. Hewlett-Packard retained the well-known numerical analyst Prof William Kahan, from UC Berkeley, the architect of the IEEE 754 standard for floating-point arithmetic, to design the numerical algorithms implemented by the calculators. He also wrote parts of the manuals.

Programming

The HP 10c series calculator are keystroke programmable, meaning that it can remember and later execute sequences of keystrokes to solve particular problems of interest to the user. These keystroke programs, in addition to performing any operation normally available on the keyboard, can also make use of conditional and unconditional branching and looping instructions, allowing programs to perform repetitive operations and make decisions.

The available programming features differentiate between the various HP 10c series calculator systems.

Function	HP 10C	HP 11C	HP 12C	HP 15C	HP 16C
BSP / ← [F 1]	No	Yes	No	Yes	Yes
LBL [F 2]	No	Yes	No	Yes	Yes
GSB/RTN [F 3]	No	Yes	No	Yes	Yes
x≤y, x=0	Yes	Yes	Yes	Yes	Yes
x=y, x≠y	No	Yes	No	Yes [F 4]	Yes
x<0, x≠0, x>y, x>0	No	No	No	Yes [F 4]	Yes
x>0, x≤0, x≥y, x≥0	No	No	No	Yes [F 4]	No
DSE, ISG [F 5]	No	Yes	No	Yes	No
DSZ, ISZ [F 5]	No	No	No	No	Yes
SF, CF, F?	No	Yes	No	Yes	Yes
I (I) [F 6]	No	Yes	No	Yes	Yes

1. ^ Without BSP (backspace) programs can only be edited by overwrite existing steps.
2. ^ Without LBL (Label) goto commands can reference only absolute program steps.
3. ^ Without GSB (Go Subroutine) / RTN (Return from Subroutine) one can not write subroutines.
4. ^ *a b c* Available via the g TEST n function
5. ^ *a b* Without DSZ/DSE (Decrement and Skip) and ISZ/ISG (Increment and Skip) writing loops is difficult.
6. ^ Without indirect addressing only the first 20 (0 .. 19) register can be accessed. Also the programming model is not turing complete.

Programming example

Here is a sample program that computes the factorial of an integer number from 2 to 69. The program takes up 8 bytes. The example is based on the feature set and display codes of to the HP 16C.

```
Step   Key-code     Display-code   Comment

01     STO 0        44 32          Store X in register I
02     1            1              Store 1 in X
03     LBL 0        43,22, 0       Label 1
04     RCL 0        45 32          Recall register I into X
05     *            20             Multiply x and y
```

```
06    DSZ         43 23         Decrement register I and skip next
command when I is 0
07    GTO 0       22  0         Goto label 1
08    R/S         31            Stop program - result displayed in x
```

This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59

Emulators

Several individuals and companies make software emulators of various HP 10C series calculators for Microsoft Windows, PalmPilots, PDAs, and smartphones.

- All HP-10C series calculators
  - hpcalc-iphone for the iPhone
- HP 12C
  - Finanx FX-12C Java
  - HP-12C emulator in Javascript
- HP 12C and 15C
  - Simulators for Windows and/or iPhone OS from HP
  - MXCalc from 3grtech
  - Lygea
  - HP 15C and HP45 Windows Phone 7 App
- HP 15C
  - HP-15C Simulator for Windows (2000 and following), Mac OS X (Intel) and Linux (x86)
- HP 16C
  - Java HP16C Emulator
  - HP 16C A Simulation
  - HP16C Emulator for Windows

Chapter 11

HP-41C

HP-41C series

HP-41CX with magnetic card reader and thermal printer

Type	Programmable Scientific
Introduced	1979
Discontinued	1990
	Calculator
Entry mode	RPN
Display Type	LCD Fourteen-segment display
	Programming
Programming language(s)	RPN key stroke (fully merged, Turing complete)
Memory Register	63 .. 319
Program Steps	441 .. 2233
	Interfaces
Ports	four vendor specific
	Other

The **HP-41C** series are programmable, expandable, continuous memory handheld RPN calculators made by Hewlett-Packard from 1979 to 1990. The original model, **HP-41C**, was the first of its kind to offer alphanumeric display capabilities. Later came the **HP-41CV** and **HP-41CX**, offering more memory and functionality.

The alphanumeric "revolution"

The alphanumeric LCD screen of the HP-41C revolutionized the way a calculator could be used, providing user friendliness (for its time) and expandability (keyboard-unassigned functions could be spelled out alphabetically). By using an alphanumeric display, the calculator could tell the user what was going on: it could display meaningful error messages ("`ZERO DIVIDE`") instead of simply a blinking zero; it could also specifically prompt the user for arguments ("`ENTER RADIUS`") instead of just displaying a question mark.

Earlier calculators needed a key, or key combination, for every available function. The HP-67 had three shift keys; the competing Texas Instruments calculators had two (2nd and INV) and close to 50 keys (the TI-59 had 45). Hewlett-Packard were constrained by their one byte only instruction format. The more flexible storage format for programs in the TI-59 allowed combining more keys into one instruction. The longest instruction required eleven keypresses, re-using the shift keys four times. The TI-59 also made use of the Op key, followed by two digits, to access another 40 different functions. But the user had to remember the codes for them. Clearly, a more convenient and flexible method of executing the calculator's instructions was in urgent need. The HP-41C had a relatively small keyboard, and only one shift key, but provided hundreds of functions. Every function that was not assigned to a key could be invoked through the XEQ key (pronounced ***EXEQ****TE* — "execute") and spelled out in full, e.g. XEQ FACT for the factorial function.

The calculator had a special user mode where the user could assign any function to any key if the default assignments provided by HP were not suited to a specific application. For this mode, the HP-41C came with blank keyboard templates; i.e. plastic covers with holes for the keys, so the user could annotate customized keys. Hewlett-Packard even sold a version of the calculator where hardly any keys had function names printed on them, meant for users who would be using the HP-41C for custom calculations only (thus not needing the standard key layout at all); this version of the calculator was colloquially known, within HP's Corvallis calculator team, as a "Blanknut" (because the development code name for the HP-41c's processor was known as the "coconut").

Alphanumeric display also greatly eased editing programs, as functions were spelled out in full. Numeric-only calculators displayed programming steps as a list of numbers, each number generally mapped to a key on the keyboard, often via row and column coordinates. Encoding functions to the corresponding numeric codes, and vice versa, was left to the user, having to look up the function–code combinations in a reference guide. The busy programmer quickly learned most of the codes, but having to learn the codes intimidated the beginners. In addition to this, the user had to mentally keep function codes separate from numeric constants in the program listing.

The HP-41C displayed each character in a block consisting of 14 segments that could be turned on or off; a so-called fourteen segment display (similar to the much more common seven segment displays, which can be used to display digits only). The HP-41C used a liquid-crystal display instead of the ubiquitous LED displays of the era, to reduce power consumption.

While this allowed the display of uppercase letters, digits, and a few punctuation characters, some designs needed to be twisted arbitrarily (e.g. to distinguish S from 5) and lowercase letters were unreadable (HP only provided display of lowercase letters a through e). HP's competitor Sharp, when introducing the PC-1211, used a dot matrix of 5×7 dots and displayed the characters in principle as we see them today on computer screens (and, in fact, many LCD screens on various embedded systems); this was later used by HP with the HP-71B handheld computer.

Expandability

The functions of the calculator could be expanded by adding modules at the back of the machine. Four slots were available to add more memory, pre-programmed solution packs containing programs covering engineering, surveying, physics, math, finance, games, etc. As such, an HP-41 could in fact be tailored to the personal needs of the user. Hardware extensions included a thermal printer, a magnetic card reader (HP-67 compatible via converter software), and a barcode "wand" (reader).

Extension modules could also add new instructions to the machine. The standard set of mathematical functions of the 41-series was somewhat limited when compared to the functionality of some of contemporary models among the HP-line of calculators (notably the HP-34C and the HP-15C). Among others, the standard function set offered no integration or root-finding capabilities and lacked support of matrices and complex numbers, which could be added by an extension module.

Another module, known as the Interface Loop allowed for connection of more peripherals: larger printers, microcassette tape recorders, 3½" floppy disk drives, RS-232 communication interfaces, video display interfaces, etc. The Interface Loop could also be used with the HP-71B, HP-75 and HP-110 computers.

The HP-41CV and CX

Many users had used all four ports for memory expansion, leaving no room for other modules. HP designed the *Quad Memory Module* with four times the amount of memory, providing the maximum available memory and leaving three empty ports available. The **HP-41CV** (V being the roman numeral for 5) included this memory module on the main board, thus providing five times the memory of the HP-41C, and four available slots.

The internal architecture prohibited the addition of more memory, so HP designed an extended memory module that could be seen as secondary storage. You could not access the data directly, but you could transfer it to and from main memory. To the calculator (and the user), data located in the extended memory looked like files on a modern hard disk do for a PC (user).

The final HP-41 model, the **HP-41CX**, included extended memory, a built-in time module, and extended functions. It was introduced in 1983 and discontinued in 1990.

Programming

The HP-41C is keystroke programmable, meaning that it can remember and later execute sequences of keystrokes to solve particular problems of interest to the user. These keystroke programs, in addition to performing any operation normally available on the keyboard, can also make use of conditional and unconditional branching and looping instructions, allowing programs to perform repetitive operations and make decisions.

The HP-41C still supports indirect addressing with which it is possible to implement a Universal Turing machine and therefore the programming model of the HP-41C can be considered Turing complete.

Programming example

Here is a sample program that computes the factorial of an integer number between 2 and 69. The program takes up 2 registers which is ≈14 bytes.

```
Step  Op-code       Comment

01    LBL'Fac       The program starts with XEQ Fac
02    STO 00        Store X in register 0
03    1             Store 1 in X
04    LBL 00        Label for goto
05    RCL 00        Recall register 0 into X
06    *             Multiply x and y
07    DSE 00        Decrement register 0 and skip next command when 0
08    GTO 00        Go to label 0
09    END           End program - result displayed in x
```

This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59

The HP-41C community and Synthetic programming

A large users' community was built around the HP-41C. Enthusiasts around the world found new ways of programming, created their own expansion modules, and sped up the clock. Most of these activities were coordinated by the PPC club and its president, Richard J. Nelson. The PPC club produced the PPC ROM, a collection of highly optimized low-level programs.

One of the discoveries of the community was that it was possible to exploit a bug in the program editor to assign strange functions to keys. The most important function was known as the byte jumper, a way to step partially through programming instructions and edit them in ways that were not otherwise allowed. The use of the resulting instructions was called *synthetic programming*.

Through synthetic instructions, a user could access memory and special status flags reserved for the operating system, and do very strange things, including completely locking the machine. It was possible to create sounds or display characters, and create animations not officially supported by the operating system. The system flags were also accessed as low-level shortcuts to boolean programming techniques. Hewlett-Packard did not officially support synthetic programming, but neither did it do anything to prevent it, and eventually even provided internal documentation to the user groups.

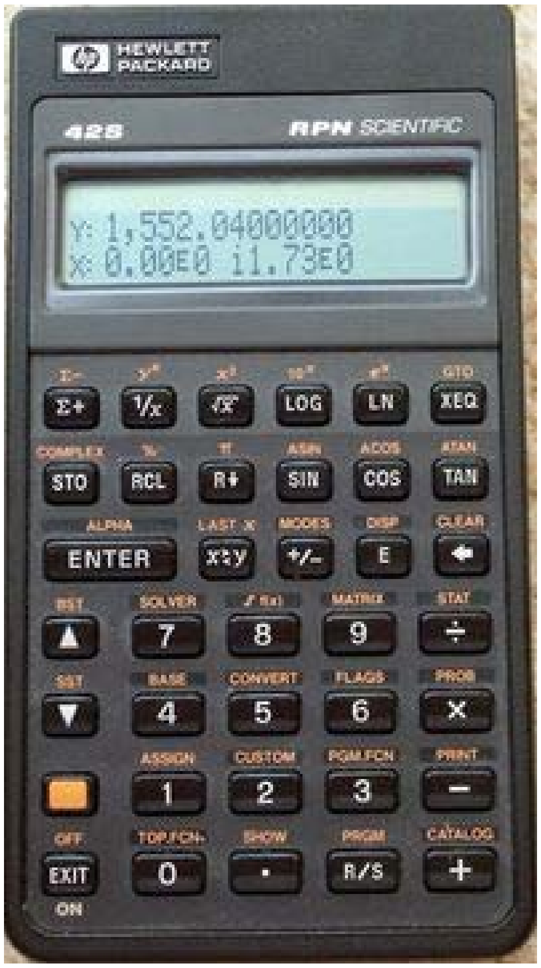
Smithsonian Museum

There is a HP-41C on display in the Smithsonian Air & Space Museum in Washington, D.C. It flew on seven Space Shuttle missions. It could have been used in an emergency to calculate orbit and re-entry information if there was failure of the shuttle main computer systems.

Chapter 12

HP-42S

HP-42S



The HP-42S

Type	Programmable Scientific
Manufacturer	Hewlett Packard

Introduced	1988
Discontinued	1995
	Calculator
Entry mode	RPN
Precision	12 display digits (25 digits internally), exponent ±499
Display Type	LCD Dot-matrix
Display Size	2 lines, 22 characters, 131x16 pixels
	CPU
Processor	Saturn
	Programming
Programming language(s)	RPN key stroke (fully merged, Turing complete)
Firmware Memory	64 KB of ROM
Program Steps	7200
	Interfaces
Ports	IR (Infrared) printing
	Other
Power supply	3×1.5V button cell batteries (Panasonic LR44, Duracell PX76A/675A or Energizer 357/303)
Weight	6 oz (170 g)
Dimensions	148×80×15mm

The **HP-42S** is a programmable RPN Scientific hand held calculator introduced by Hewlett Packard in 1988. It has advanced functions suitable for applications in mathematics, linear algebra, statistical analysis, computer science and others.

HP-41 replacement

The HP-42S was released as a replacement for the aging HP-41 series, and is designed to be compatible with all programs written for the HP-41. However, it has received criticism for its lack of expandability, and lack of any real I/O ability, both key features of the HP-41 series.

The 42S, however, has a much smaller form factor than the 41, and features many more built-in functions, such as a matrix editor, complex number support, an equation solver, user-defined menus, and basic graphing capabilities. Additionally, it features a two-line dot matrix display, which made stack manipulation easier to understand.

Production of the 42S ended in 1995, due at least in part to the substantial increase in the production cost of the CPU chip. In the HP calculator community, the 42S is infamous for its inflated prices in online auctions, the rare new-in-box calculators typically command in excess of $400 USD. This is nearly a four-fold increase in price over its introduction cost and has created a scarcity for utility end users. Yet, this calculator is often regarded as the best ever made in terms of quality, key stroke feel, ease of programming, and daily usability for engineers. Many scientists and engineers use this calculator today, 20 years after its introduction.

HP-42S specifications

HP-42S battery compartment and the IR diode

- Series: Pioneer
- Code Name: Davinci
- Introduction: 10-31-1988
- 64 KB of ROM
- Functions: Over 600
- Expandability: Officially no other than IR printing (32K memory upgrade and over-clocking hardware hacks are possible)
- Peripherals: HP-82240B Infrared Printer

HP-42S features

- All basic scientific functions (including hyperbolic functions)
- Statistics (including curve fitting and forecasting)
- Probability (including factorial, random numbers and Gamma function)
- Equation solver (root finder) that can solve for any variable in an equation

- Numerical integration for calculating definite integrals
- Matrix operations (including a Matrix Editor, dot product, cross product and solver for simultaneous linear equations)
- Complex numbers (including polar coordinates representation)
- Vector functions
- Named variables, registers and binary flags
- Graphic display with graphics functions and adjustable contrast
- Menus with submenus and mode settings (also custom programmable) that use the bottom line of the display to label the top row of keys
- Sound (piezoelectric beeper)
- Base conversion, integer arithmetic and binary and logic manipulation of numbers in Binary, octal, decimal and hexadecimal systems
- Catalogs for reviewing and using items stored in memory
- Programmability (keystroke programming with branching, loops, tests and flags)
- The ability to run programs written for the HP-41C series of calculators

Sample programs

The HP-42S is keystroke programmable, meaning that it can remember and later execute sequences of keystrokes to solve particular problems of interest to the user. The HP-42S uses a superset of the HP-41CX Focal language.

The HP-42S supports indirect addressing with which it is possible to implement a Universal Turing machine and therefore the programming model of the HP-42S can be considered Turing complete.

Factorial

This is a sample program which computes the factorial of an input integer number. The program consume 18 bytes.
** in all samples the line numbers are part of HP-42S code editor, and programmer does not have to write them in*

```
Step  Op-code      Comment
00    { 18-Byte Prgm }
01    LBL "FAC"    The Programm is started with XEQ FAC
02    STO 00       Store X in register 0
03    1            Store 1 in X
04    LBL 00       Label for goto
05    RCL× 00      Recall register 0 and multiply with X
06    DSE 00       Decrement register 0 and skip next command wenn 0
07    GTO 00       Goto label 0
08    .END.        End program - result displayed in x
```

This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59.

Speed

Program for calculating speed according to formula v = s / t

```
01 LBL "SPEED"
02 INPUT "s"
03 INPUT "t"
04 "Speed equals to"
05 AVIEW
06 RCL "s"
07 RCL "t"
08 ÷
```

Speed (for Solver)

Version for "Solver" application, which allows evaluating any variable from the formula v = s / t

```
01 LBL "SPEED2"
02 MVAR "v"
03 MVAR "s"
04 MVAR "t"
05 RCL "s"
06 RCL "t"
07 ÷
08 RCL "v"
09 -
```

Primality tester

A simple primality tester, from the alternative manual

```
01 LBL "PRIME"
02 STO 00
03 2
04 STO 01
05 MOD
06 X=0?
07 GTO F
08 3
09 STO 01
10 RCL 00
11 SQRT
12 STO 02
13 LBL B
14 RCL 00
15 RCL 01
16 MOD
17 X=0?
18 GTO F
19 2
20 STO + 01
21 RCL 02
```

```
22 RCL 01
23 X≤Y?
24 GTO B
25 RCL 00
26 STO 01
27 LBL F
28 RCL 01
29 RTN
```

Chapter 13

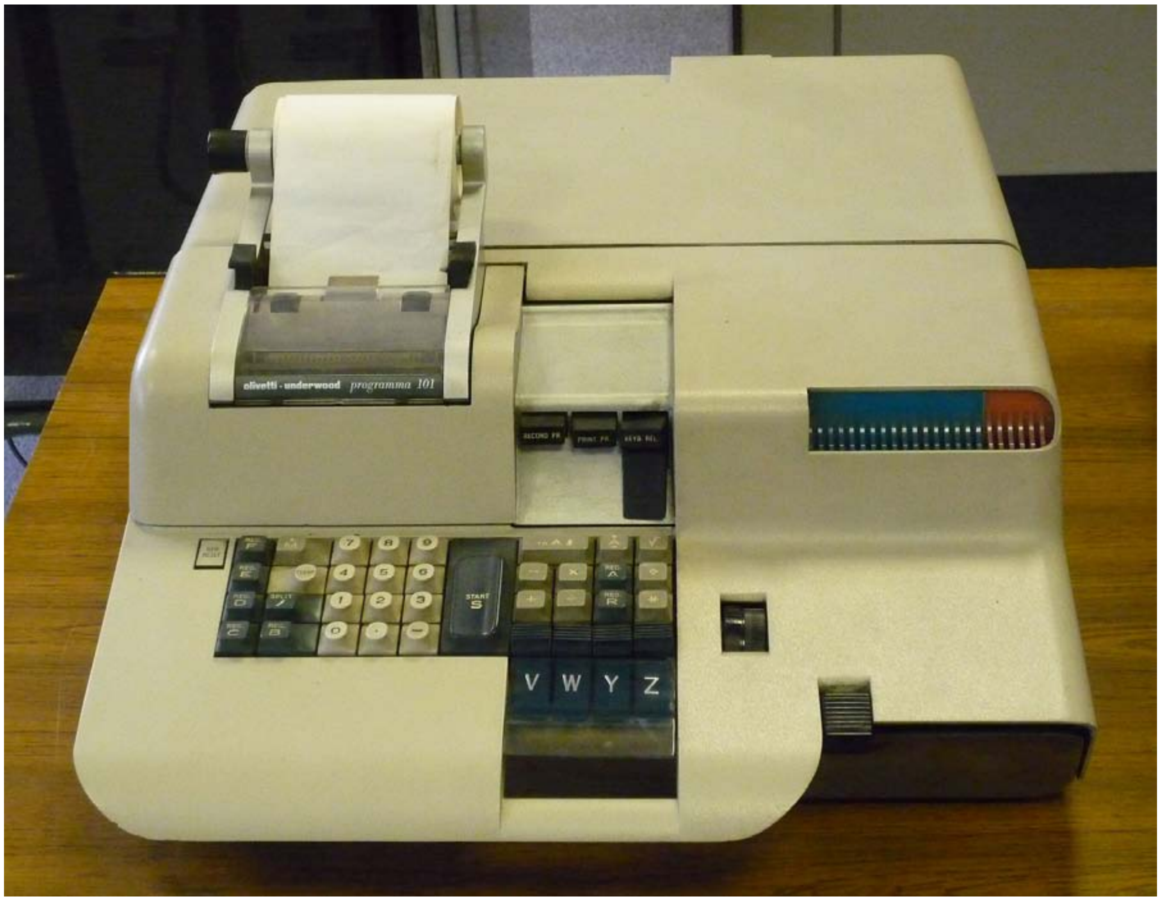
Programma 101 and Sharp EL-5120

Programma 101

A Programma 101.

The **Programma 101** was a printing programmable calculator manufactured by Olivetti in 1965. A futuristic design for its time, the Programma 101 was priced at $3,200. About 40,000 units were sold.

Capabilities



Front view of a programma 101 showing the printer and programming keys

The Programma 101 was able to calculate the basic four arithmetic functions (addition, subtraction, multiplication, and division), plus square root, absolute value, and fractional part. Also clear, transfer, exchange, and stop for input. There were 16 jump instructions and 16 conditional jump instructions. 32 label statements were available as destinations for the 32 jump instructions and/or the four start keys (V, W, Y, Z).

Each full register held a 22-digit number with sign and decimal point.

Its memory consisted of 10 registers: three for operations (M, A, R); two for storage (B, C); three for storage and/or program (assignable as needed) (D, E, F); and two for program only (p1, p2). Five of the registers (B, C, D, E, F) could be subdivided into half-registers, containing an 11-digit number with sign and decimal point. When used for programming, each full register stored 24 instructions.

It printed programs and results onto a roll of paper tape, similar to calculator or cash register paper.

Programming was similar to assembly language, but simpler, as there were fewer options. It directed the exchange between memory registers and calculation registers, and operations in the registers.

The stored programs could be recorded onto plastic cards approximately 10 cm x 20 cm that had a magnetic coating on one side and an area for writing on the other. Each card could be recorded on two stripes, enabling it to store two programs. All ten registers were stored on the card, allowing programs to use up to ten stored 11-digit constants.

The program to calculate logarithms occupied both stripes of one magnetic card.

Construction

All computation was handled by discrete devices (transistors and diodes mounted on phenolic resin circuit card assemblies), as there were no microprocessors, and integrated circuits were just beginning. It used an acoustical delay line memory with metal wires as a data storage device. Magnetostriction transducers inside an electromagnet attached to either side of the end of the wire. Data bits entering the magnets caused the transducer to contract or expand (based on binary value) and to twist the end of the wire. The resulting torsional wave moved down the wire. A piezoelectric transducer converted the bits into an electronic signal that was then amplified and sent back to the beginning with a delay time of 2.2 milliseconds. Typically, many bits would be in transit through the delay, and the computer selected them by counting and comparing to a master clock to find the particular bit it required. Delay line memory was far less expensive and far more reliable per bit than flip-flops made from vacuum tubes, and yet far faster than latching relays.

History

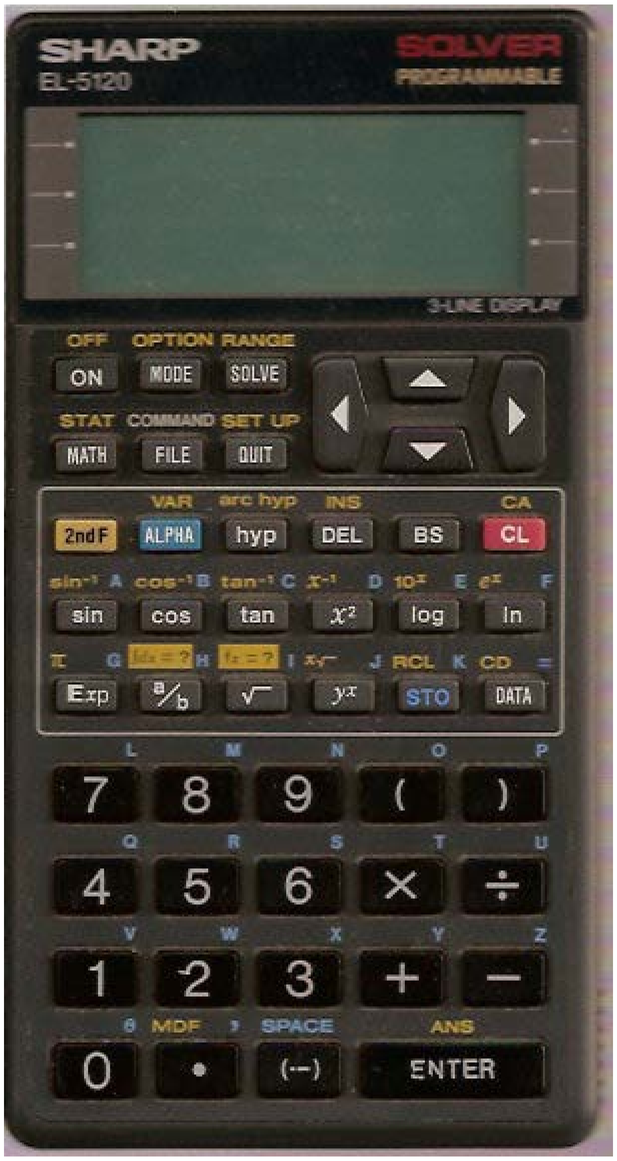
It was designed by Olivetti engineer Pier Giorgio Perotto in Ivrea. The styling, attributed to Marco Zanuso but in reality by Mario Bellini, was ergonomical and innovative for the time, and earned Bellini the Compasso d'Oro Industrial Design Award.

Developed between 1962 and 1964, it was launched for the first time at the 1964 New York World's Fair, attracting major interest. 40,000 units were sold; 90% of them in the United States where the sale price was $3,200 (increasing to about $3,500 in 1968.)

Hewlett-Packard was ordered to pay about $900,000 in royalties to Olivetti after using some of the solutions adopted in Programma 101 in the HP 9100.

The 101 is mentioned as part of the system used by the US Air Force to compute coordinates for ground directed bombing of B-52 Stratofortress targets during the Vietnam War.

Sharp EL-5120



A Sharp EL-5120 scientific calculator

The Sharp EL-5120 is a scientific programmable calculator. It has about 1 KB of total RAM available to the user, and has 4 basic operational modes:

- **Real** mode: it is the basic operational mode for direcltly performing standard algebraic and statistical calculations, as well as evaluating user-defined functions and numerically integrating them.
- **NBase**: can switch between Binary, Octal, Decimal and Hexadecimal base. Most functions from Real Mode don't work in this mode, but boolean operators for each numerical base are available. Hexadecimal base calculation are performed in 32

bits (8 digits) and there is support for signed operation, but Binary base is limited to 16 bits, though.
- **Solver**: an interactive expression solver which can, in theory, numerically solve any equation versus any variable, using Newton's method. It may however fail to solve certain classes of equations depending on the expression format and starting values of the variables, so it is often necessary to rewrite the expression or experiment with initial values.
- **Program mode**: Here the user can enter and execute short programs written in a language closely resembling a cut-down version of FORTRAN or BASIC. Programs can be made to operate in either **Real** or **NBase** mode, but not a mixture of both.

Main functions

- 3-line alphanumeric LCD display.
- Alphanumeric keyboard with **SHIFT** and **ALPHA** keys.
- All the standard trigonometrical functions (SIN, COS, TAN) as well as their inverse and hyperbolic versions.
- All of the standard power raising, logarithmic etc. functions
- Some functions like statistical operations and boolean logic functions are accessed via sub-menus, and thus they are not printed on any visible key.
- 28 global user variables (**A** through **Z** plus **ANS** and **θ**), stored in CMOS memory.
- Up to 9 local variables for each mode of operation and equation, solver or program file, with user defined names. Unlike the 28 global ones, using these local variables consumes user RAM.
- 1 and 2 variables statistics, has only a simple linear regression analysis.
- File "saving", "loading" and "deleting" from the small user RAM. Each mode can store its own "files", containing e.g. the last calculation or expression, a solver equation or a program plus any eventual local variables and the last ANS value.
- Expression evaluator (in **Real** mode).
- Numerical integration using Simpson's rule.
- Numerical equation solver vs a specific variable using Newton's method.
- Programs and solver equations can "exchange data" between them by appropriate use of the global variables, for solving more complex problems.
- Adjustable contrast.
- Uses one 3V CD2025 lithium battery.

Disadvantages

- Lack of built-in support for complex numbers (can only be emulated via a program or equations).
- Programs and equation files eat up RAM quickly, especially if they contain local variables.
- Integration/solver functions can be slow or erratic.
- Only one kind of statistical regression (linear).

- Lack of any built-in application formulae or physical constants, these have to be defined and saved by the user as expression and local variables, with a notable expense of RAM.
- Lack of some built-in functions like a simultaneous linear equations system or second grade equations solver, thus forcing to implement them by programming.
- The programming language used consumes RAM too quickly due to weak construction, lacking a proper **FOR**-like statement and thus forcing to use long and costly `LABEL`, `GOTO` and `GOSUB` statements.

Sample programs

Please note that the actual notation might be different, as some special EL-5120 characters cannot be directly typed on a PC, e.g. the square root and fraction operator:

```
 Hello world:

 HELLO: REAL
 LABEL 10
 PRINT"HELLO WORLD
 GOTO 10

 Solver of second grade equations:

 GRADE2:REAL
 INPUT A
 INPUT B
 INPUT C
 D=B²-4AC
 IF D<0 GOTO ERR
 X=(-B-sqrt(D))/(2A)
 Y=(-B+sqrt(D))/(2A)
 PRINT X
 PRINT Y
 GOTO E
 LABEL ERR
 PRINT D
 LABEL E
 END

 Calculate ICE current and VCE voltage for a BJT transistor,
 using the 4-resistor polarization method.

 Note: These local variables must be defined first:
   R1,R2,R3,R4,B0=gain ,V8=0.7 or 0.3 (base voltage in V for silicon or
germanium BJTs, accordingly)

BTJ-4R:REAL

INPUT R1
INPUT R2
INPUT R3
INPUT R4
INBUT B0
```

```
INPUT V
R=R1R2/(R1+R2)
T=VR2/(R1+R2)
I=(T-V8)/(R+(B0+1)R4)
C=V-I(B0R3+(B0+1)R4)
I=B0I
PRINT I
PRINT C
```

Chapter 14

TI-59

TI-59

A TI-59 showing one card in the holder on the front of the calculator and another being inserted into the card reader in the side.

Type	Programmable
Manufacturer	Texas Instruments
Introduced	May 1977
Discontinued	1983
Cost	US$300
Calculator	
Entry mode	Infix
Precision	13
Display Type	Light-emitting diode
Display Size	10 digits
Programming	
Programming	key stroke (Turing-

language(s)	complete)
Memory Register	100
Program Steps	960
Other	
Weight	240 grams
Dimensions	16.3x7.3x3.6 cm

The **TI-59** was an early programmable calculator, manufactured by Texas Instruments from 1977. It was the successor to the **TI SR-52**, quadrupling the number of "program steps" of storage, and adding "ROM Program Modules" (an insertable ROM chip, capable of holding 5000 program steps.) It was one of the first LED calculators with the capability and flexibility to take on many real-world calculation challenges, and quickly became popular with professionals in many fields.

The calculator could be powered from an external adapter or from its internal NiCd rechargeable battery pack.

Display

10-digit LED display

The red LED display showed 10 decimal digits of precision.

Programming

Programming simple problems with the TI-59 was a very straightforward process. In programming mode, the TI-59 simply recorded key presses. Alphabetical keys provided easy access to up to ten entry points. It was also possible to activate any of the programs in the pre-programmed memory module, and run it like if you had programmed it yourself. Programs written by the user could also use programs in the module as subroutines. The module's programs ran directly from ROM, so they left the calculator's memory free for the user.

However, exploiting the computerlike capabilities of the TI-59 was a different matter. Although the TI-59 was Turing-complete, supporting straight-line programming, conditions, loops, and indirect access to memory registers, and although it supported limited alphanumeric output on the printer only, writing sophisticated routines was essentially a matter of planning machine language and using a coding pad.

A large degree of sharing occurred in the TI-59 and TI-58 C community.

Programming example

Here is a sample program that computes the factorial of an integer number from 2 to 69. For 5!, you'll type 5 A and get the result, 120. Unlike the **SR-52**, the **TI-59** didn't have the factorial function built-in, but it did support it through the software module which was delivered with the calculator.

```
Op-code         Comment

LBL A           You'll call the program with the A key
STO 01          stores the value in register 1
1               starts with 1
LBL B           label for the loop
*               multiply
RCL 01          by n
DSZ 1 B         décrements n and back to B until n=0
=               end of loop, the machine has calculated 1*n*(n-
1)*...2*1=n!
INV SBR         end of procedure
```

Here is the same program written for **TI Compiler**:

```
#reg  01 counter
#label A factorial

LBL factorial
  STO counter
  1
  FOR counter
    * @counter
  LOOP
  =
RTN

#end
```

This example is also available for the following calculators and pocket computers: Casio FX-502P, Casio FX-602P, Casio FX-603P, HP-16C, HP-41C, HP-42S, HP 35s, Sharp PC-1403, TI-59

Memory

In comparison to its contemporary main competitor, Hewlett-Packard's HP-67, the TI-59 had an extremely large memory of up to 960 user program steps or 100 memories (adjustable between the two, in steps). The **TI-59** was the first programmable pocket calculator where the manufacturer provided a system for sharing memory between data registers and program storage. The memory was only about twice as large as in the **SR-52**, but more flexible, and thus the possible number of program steps was four times as high. Contents of this memory were lost, when the calculator was turned off.

Magnetic card reader

The Master Library Module shown removed from its socket in the back of the calculator. Magnetic card storage folio also shown.

Programs and data could be stored on small magnetic cards when the calculator was turned off and quickly reloaded when needed.

Solid State Software Library

The TI-59 was the first hand-held calculator to utilize removable ROM program modules. The Master Library Module ROM was included with the TI-59, and contained several useful pre-programmed routines and even a game. Additional modules - for such applications as real estate, investment, statistics, surveying and aviation - were sold separately.

Printer

Also available for the TI-59 was a thermal printer (the PC100C); the calculator was mounted on top of the printer. The printer was extremely valuable because it could print out a hard copy of the calculator's program, where the instructions were listed with the same alphanumeric mnemonics as the keys were labelled with, not just the numeric key codes.

Chapter 15

TI-83 Series

TI-83

TI-83, original design

Type	Graphing calculator
Manufacturer	Texas Instruments
Introduced	1996
Discontinued	2004
Successor	TI-83 Plus
Calculator	
Entry mode	D.A.L.
CPU	

Processor	Zilog Z80
Frequency	6 MHz
Programming	
Programming language(s)	TI-BASIC,Assembly
User Memory	32 KB RAM
Other	
Power supply	4 AAAs, 1 CR1616 or CR1620

The **TI-83 series** of graphing calculators is manufactured by Texas Instruments. The original TI-83 is itself an upgraded version of the TI-82. Released in 1996, it is one of the most used graphing calculators for students. In addition to the functions present on normal scientific calculators, the TI-83 includes many features, including function graphing, polar/parametric/sequence graphing modes, statistics, trigonometric, and algebraic functions. Although it does not include as many calculus functions, applications and programs can be downloaded from certain websites, or written on the calculator.

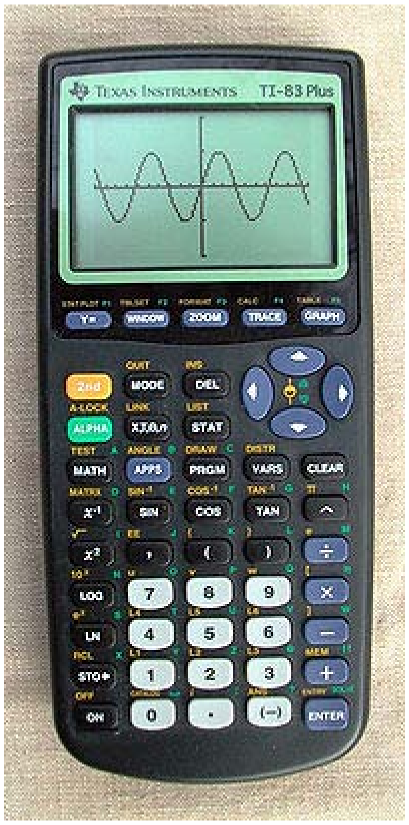
TI replaced the TI-83 with the **TI-83 Plus** calculator in 1999, which included flash memory, enabling the device's operating system to be updated if needed, or for large new Flash Applications to be stored, accessible through a new Apps key. The Flash memory can also be used to store user programs and data. In 2001, the TI-83 Plus Silver Edition was released, which featured approximately nine times the available Flash memory, and over twice the processing speed (15 MHz) of a standard TI-83 Plus, all in a translucent "sparkle" grey case.

The TI-83 was the first calculator in the TI series to have built in assembly language support. The TI-92, TI-85, and TI-82 were capable of running assembly language programs, but only after sending a specially constructed (hacked) memory backup. The support on the TI-83 could be accessed through a hidden feature of the calculator. Users would write their assembly (ASM) program on their computer, assemble it, and send it to their calculator as a program. The user would then execute the command "Send (9prgm*XXX*" (where *XXX* is the name of the program), and it would execute the program. Successors of the TI-83 replaced the Send() backdoor with a less-hidden Asm() command.

The TI-83 was redesigned twice, first in 1999 and again in 2001. The 1999 redesign introduced a design very similar to the TI-73 and TI-83 Plus, eliminating the sloped screen that has been common on TI graphing calculators since the TI-81. The 2001 redesign (nicknamed the TI-83 "Parcus") introduced a slightly different shape to the calculator, eliminated the glossy screen border, and reduced cost by streamlining the printed circuit board to four units.

TI-83 Plus

TI-83+



The TI-83 Plus (second version of TI-83)

Type	Graphing calculator
Manufacturer	Texas Instruments
Introduced	1999
Predecessor	TI-83
Successor	TI-84 Plus
Calculator	
Entry mode	D.A.L.
CPU	
Processor	Zilog Z80
Frequency	6 MHz
Programming	
Programming language(s)	TI-BASIC Assembly
User Memory	27 KB RAM 160 KB flash ROM
Firmware Memory	512 KB

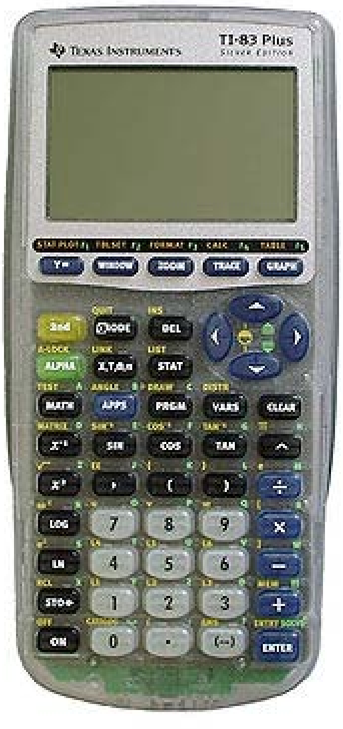
	Other
Power supply	4 AAA's, 1 CR1616 or CR1620

The **TI-83 Plus** is a graphing calculator made by Texas Instruments, designed in 1999 as an upgrade to the TI-83. The TI-83 Plus is one of TI's most popular calculators. It uses a Zilog Z80 microprocessor running at 6 MHz, a 96×64 monochrome LCD screen, and 4 AAA batteries as well as backup CR1616 or CR1620 battery. A link port is also built into the calculator in the form of a 2.5mm jack. The main improvement over the TI-83, however, is the addition of 512 KB of Flash ROM, which allows for OS upgrades and applications to be installed. Most of the Flash memory is used by the OS, with 160 KB available for user files and applications. Another development is the ability to install Flash Applications, which allows the user to add functionality to the calculator. Such applications have been made for math and science, text editing, organizers and day planners, editing spread sheets, games, and many other uses.

Designed for use by high school students, though now used by middle school students in some public school systems, it contains all the features of a scientific calculator as well as function, parametric, polar, and sequential graphing capabilities; an environment for financial calculations; matrix operations; on-calculator programming; and more. Symbolic manipulation (differentiation, algebra) is not built into the TI-83 Plus. It can be programmed using a language called TI-BASIC, which is similar to the BASIC computer language. Programming may also be done in TI Assembly, made up of Z80 assembly and a collection of TI provided system calls. Assembly programs run much faster, but are more difficult to write. Thus, the writing of Assembly programs is often done on the computer.

TI-83 Plus Silver Edition

TI-83+SE



TI-83 Plus Silver Edition

Type	Graphing calculator
Manufacturer	Texas Instruments
Introduced	2001 (83+SE)
Discontinued	2004
Successor	TI-84 Plus Silver Edition
Calculator	
Entry mode	D.A.L.
CPU	
Processor	Zilog Z80
Frequency	15 MHz (83+SE)
Programming	
Programming language(s)	TI-BASIC Assembly
User Memory	128 KB RAM

	1.5 MB flash ROM
Other	
Power supply	4 AAA's, 1 CR1616 or CR1620

The **TI-83 Plus Silver Edition** is a newer version of the TI-83 Plus calculator, released in 2001. Its enhancements are 1.5 MB of Flash memory, a dual-speed 6/15 MHz processor, 96 KB of additional RAM (but TI has yet to code support for the entire RAM into an OS), an improved link transfer hardware, a translucent silver case, and more applications preinstalled. It also includes a USB link cable in the box. It is almost completely compatible with the TI-83 Plus; the only problems that may arise are with programs (i.e. games) that may run too quickly on the Silver Edition or with some programs which have problems with the link hardware. The key layout is the same. The TI-83 Plus Silver Edition is listed on the Texas Instruments website as "discontinued."

In April 2004, the TI-83 Plus Silver Edition was replaced by the TI-84 Plus Silver Edition. They feature the same processor and the same amount of Flash memory, but the TI-84 Plus Silver Edition features a built in USB port, clock, and changeable faceplates.

TI-84 Plus series

The TI-84 Plus series was introduced in April 2004 as a further update to the TI-83 line. Despite the new appearance, they are not vastly superior to the TI-83 Plus series. The main improvements of the TI-84 Plus and TI-84 Plus Silver Edition are: a modernized case design, changeable faceplates (Silver Edition only), new built in functions, more speed and memory over the TI-83 and TI-83 Plus, a built-in clock, and built-in USB port connectivity. The Ti-84 Plus also has a brighter screen with more contrast, and the keys feel crisper, though they are smaller than the ones on the Ti-83. The TI-84 Plus has 3 times the memory of the TI-83 Plus, and the TI-84 Plus Silver Edition has 9 times the memory of the TI-83 Plus. They both have 2.5 times the speed of the TI-83 Plus. The operating system and math functionality remain essentially the same, as does the standard link port for connecting with the rest of the TI Calculator series.

Technical specifications

CPU

Zilog Z80 CPU, 6 MHz (TI-83, 83+), or 15 MHz (Silver Edition), or Inventec 6S1837 (TI-83+ revision A)

ROM

24 KB ROM (TI-83)

Flash ROM

512 KB with 163 KB available for user data and programs (83+) or 2 MB (Silver Edition)

RAM

32 KB RAM with 24 KB available for user data and programs (128 KB on Silver Edition, however the extra 96 KB is *not* user accessible by default, this extra

memory is used in some Applications such as Omnicalc for a RAM recovery feature and a *virtual calc*)

84 series expandable via special software to use up disk space on a USB memory drive.

Display

Text: 16×8 characters (normal font)

Graphics: 96×64 pixels, monochrome

I/O

Link port, 9.6 kbit/s

50 button built-in keypad

Power

4 AAA batteries plus 1 CR 1616 or CR 1620 for backup

Integrated programming languages

TI-BASIC, Assembly language and machine code. C requires a computer with a Z80 cross-compiler or an on-calc assembler.

Cryptographic Keys

In 2009, a group of enthusiasts used brute force and distributed methods to find all of the cryptographic signing keys for the calculator firmwares, allowing users to directly flash their own operating systems to the devices. The key for the TI-83 calculator was first published by someone at the unitedti.org forum. They needed several months to crack it. The other keys were found after a few weeks by the unitedti.org community through a distributed computing project. Texas Instruments then began sending out DMCA take-down requests to a variety of different websites mirroring the keys, including unitedTI and reddit.com. They then became subject to the Streisand effect and were mirrored on a number of different sites.