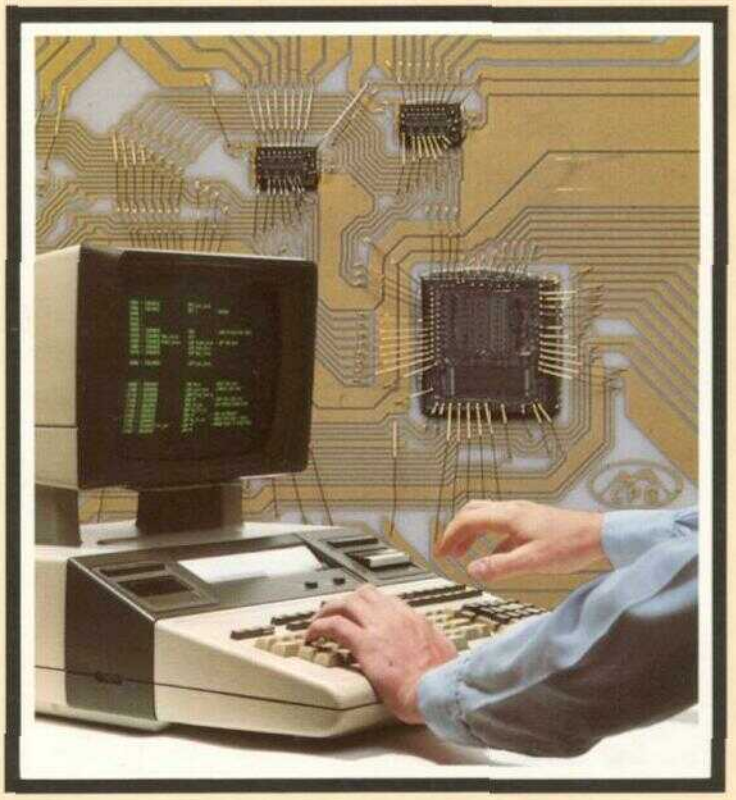
Series 9800 Desktop Computers

System 35/45 Assembly Language

Optimizing program execution times.



HEWLETT PACKARD

Enhancing a desktop computer's program efficiency and speed

Programming on today's desktop computer is best accomplished using a high level language such as HP Enhanced BASIC, however there are certain computation-intensive tasks and specialized I/O routines that could be made faster and more efficient if they could be coded in a lower level language. Often, the best solution is to combine the speed of a low level language plus the power and flexibility of a general-purpose BASIC — intermixed in the same program.

That's exactly what we've done.

Hewlett-Packard's new System 35/45 Assembly Language capability gives experienced programmers all the versatility and control they need to get the most out of desktop computer programs. It is designed specifically to enhance the System 35 and 45 Desktop Computer by providing speed increases in time-critical portions of BASIC programs. It offers the programmer complete control of the System 35 or 45 Central Processing Units (CPUs) through the use of machine instructions, pseudo-instructions and extensions to the BASIC language. It allows specialized subroutines to be written in Assembly Language which are then callable from a BASIC program.

If you have ever programmed in Assembly Language, you know it can be very unfriendly. But we have developed an Assembly Language System that takes advantage of all the friendly features of our desktop computers — and then adds a few new ones:

- source statements that are syntaxed as they are entered, allowing you to catch errors as they are made;
- an assembly process that is entirely integrated, i.e., the assembler and editor are in Read Only Memory (ROM) and the Source/Object codes are in Read/Write Memory, thereby eliminating time consuming mass storage accesses;
- an assembler that can assemble code at up to 800 statements per second;
- a system that is available in two different ROM configurations: an Execution and Development ROM, and an Execution ROM;
- and, most importantly, an extensive set of debug tools that allow interactive debugging from the System 35 or 45 keyboard plus the ability to write and execute debug routines in BASIC.

The sum of these features results in getting Assembly Language routines up-and-running much quicker than you might have thought possible.

Increasing the speed of computation. BASIC Language statements are usually written as general purpose routines capable of handling a wide assortment of requirements. With Assembly Language, you can tailor a subprogram to handle a specific task, thus eliminating unnecessary overhead. In general, speed increases with Assembly Language are the result of the programmer knowing very specific information about the task and then writing code in a streamlined fashion to efficiently handle that task. For example, significant increases can be realized when using integer precsion, as seen in the table below.

Speed up I/O response. Many I/O applications require the desktop computer to respond quickly to interrupts. Assembly Language Interrupt Service Routines (ISRs) on the System 35 and 45 Desktop Computers are attended to within 80 to 150 microseconds instead of waiting for the end of the BASIC line. This can result in a 300X increase in the efficiency of handling interrupts.

I/O throughput can be dramatically increased using Assembly Language by reducing the amount of overhead incurred by a BASIC program. This overhead is a result of the interpretive operating system of the System 35 and 45, plus the flexibility of the HP Enhanced BASIC statements.

Some specific areas where Assembly Language can aid in reducing overhead are:

- repetitive, small word-count transfers;
- fast scanning of devices on different select codes;
- intermixed computation and I/O;
- evaluating I/O "on-the-fly" for error conditions, etc.

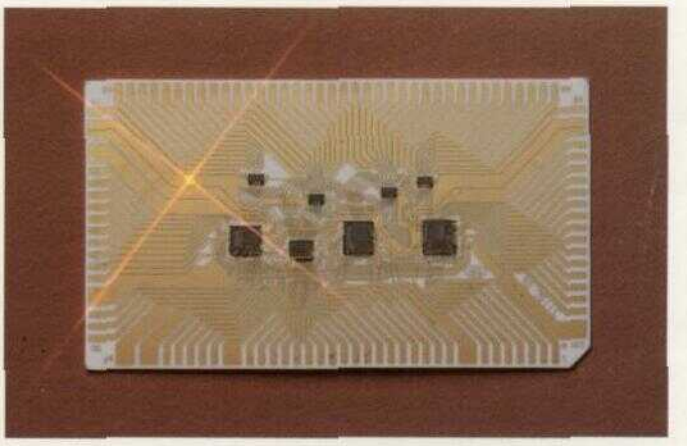
Graphics with Assembly Language. Assembly Language provides complete control of the System 45 CRT hardware and includes the setting and checking of individual dots, the writing and reading of full words of graphic information and the control of two graphics cursors. The major advantage of using Assembly Language rather than BASIC to create and manipulate graphics information is speed. Graphical data can be manipulated and input information plotted in real time using Assembly Language.

The display of graphics information is essentially an I/O operation to the graphics hardware. Thus, all speed increases associated with I/O also apply to graphics. An algorithm that intermixes data input, computation and display will realize a significant speed increase over BASIC. However, since full access to the CRT graphics hardware is available in BASIC (and is much simpler), the use of Assembly Language should be limited to those graphics areas where speed is critical.

APPROXIMATE SPEED INCREASES*

	Assembly (integer math) in microseconds	BASIC (floating point) in microseconds	Approx. Speed Improvement Factor
Add	2.2	290	130.0
Subtract	4.4	350	80.0
Multiply	13.5	1 000	75.0
Divide routine	1 000	3 100	3.1
Single floating point operation vs. BASIC			0.8
Array manipulation			
Integer			190.0
Real			3.0
Loops, Branching, Compares, Indexing			100.0

*These execution times are approximate and represent an average over a number of interations.



The Processors

The CPUs used in the System 35 and 45 Desktop Computers are Hewlett-Packard designed 16-bit hybrids that offer:

- 2 arithmetic accumulators (A & B),
- 2 general purpose stack pointers (C & D) with byte or word addressing capability,
- indirect addressing (one level),
- 2 levels of priority interrupts with 8 hardware vectors per level,
- Direct Memory Access (DMA) capability,
- hardware floating point BCD arithmetic with two floating point registers: Ar1 (located in R/W memory) and Ar2 (located in the processor),
- 16 bit instructions and data,
- integer arithmetic,
- 16 sixteen-bit bi-directional I/O ports.

The System 45 features a unique dual processor (CPU) system; a single one is used in the System 35. For non-ISR Assembly Language code, the System 45's dual processors function as a single unit to maintain compatability with the System 35. The major advantages of the dual processor system in the System 45 are:

- overlapped I/O can, in some cases, bring about increased throughput;
- Assembly Interrupt Service Routines can be executed simultaneously with a BASIC program.

The Assembler

The System 35/45 Assembly Language System has a relocating assembler whose pre-eminent feature is speed. It can assemble statements at 400 to 800 per second. This speed is possible because the assembly process is integrated and the source statements are syntaxed as they are entered.

Other features include:

- symbols up to 15 characters long; programs are more self-documented;
- symbols can represent constants, relocatable addresses, machine addresses or external symbols;
- constants can be octal or decimal integers, full or short precision floating point or ASCII;
- selected sections of source code can be omitted from the assembly process via conditional assembly instructions;
- statements are provided to generate listings and cross references.

This assembler can perform any add, subtract, multiply or divide operation that occurs within the operand field.

The System

The System 35/45 Assembly Language System consists of a set of plug-in ROMs that contain the Assembly Language statements and functions. These ROMs are offered in two configurations: Execution and Development, and Execution Only.

The advantage of having these two configurations can be seen when building an economical desktop computer-based system utilizing the Assembly Language capability. For example, you could have a single desktop computer that has full capability, i.e., equipped with the Assembly Execution and Development ROM for writing and debugging programs. The rest of the units in your system could be equipped with the lower cost Execution ROMs.

Figure 1. *BASIC program using an Assembly Language subprogram.*

```
   10   ! THIS PROGRAM OUTPUTS A STRING USING HANDSHAKE TO A GPIO-LIKE INTERFACE
 * 20   ICOM 1000                          ! SET ASIDE R/W FOR OBJECT CODE
   30   DIM Input$[160]                    ! ALLOW FOR 160 CHARACTER STRING
 * 40   IASSEMBLE Output;LIST,XREF         ! ASSEMBLE MODULE Ouput WITH LISTING
   50                                      !  AND CROSS REFERENCE
   60 Input:   LINPUT "STRING TO WRITE?",Input$ ! ASK USER FOR STRING TO OUTPUT
 * 70   ICALL Output_gpio_hs(Input$)       ! CALL THE ASSEMBLY SUBPROGRAM
   80   GOTO Input
   90   !
 * 100       ISOURCE                NAM Output          ! NAME THE MODULE
   110       ISOURCE                EXT Get_value       ! DECLARE ANY ROUTINES
   120       ISOURCE                EXT Error_exit      !  EXTERNAL TO THE MODULE
   130       ISOURCE String:        BSS 81              ! RESERVED FOR 160 CHAR STRING
   140       ISOURCE Select_code:EQU 9                  ! EQUATE FOR SELECT CODE
   150       ISOURCE !
   160       ISOURCE                SUB                 ! DECLARES ENTRY POINT TO BE
   170       ISOURCE Parm_str:      STR                 !  Output_gpio_hs
   180       ISOURCE Output_gpio_hs: LDA =Select_code   ! SET THE SELECT CODE
   190       ISOURCE                STA Pa
   200       ISOURCE                LDA =String         ! GET THE STRING PARAMETER
   210       ISOURCE                LDB =Parm_str       !  USING SYSTEM UTILITY
   220       ISOURCE                JSM Get_value
   230       ISOURCE                LDA =String+1       ! SET UP C TO GET BYTES FROM
   240       ISOURCE                SAL 1               ! THE STRING
   250       ISOURCE                STA C
   260       ISOURCE                CBL
   270       ISOURCE                LDA String          ! IF THE STRING LENGTH IS ZERO
   280       ISOURCE                SZA Done            ! THEN THERE IS NOTHING TO DO.
   290       ISOURCE Write_loop:    WBC A,I             ! GET THE NEXT CHAR FOR OUTPUT
   300       ISOURCE                JSM Write_byte      ! OUTPUT THE CHARACTER TO CARD
   310       ISOURCE                DSZ String          ! SEE IF DONE
   320       ISOURCE                  JMP Write_loop    ! IF NOT, REPEAT
 * 330       ISOURCE Done:          RET 1               ! RETURN TO BASIC
   340       ISOURCE !
   350       ISOURCE Write_byte:    SSC Card_down       ! SKIP IF CARD IS DOWN
   360       ISOURCE                SFC Write_byte      ! ELSE WAIT FOR CARD
   370       ISOURCE                STA R4              ! OUTPUT DATA TO CARD
   380       ISOURCE                STA R7              ! TRIGGER HANDSHAKE
   390       ISOURCE                RET 1
   400       ISOURCE !
   410       ISOURCE Card_down:     LDA =164            ! RETURN ERROR 164 TO BASIC
   420       ISOURCE                JSM Error_exit
   430       ISOURCE !
 * 440       ISOURCE                END Output          ! END THE MODULE
```

Parameter passing — 170

Subprogram entry point — 180

Calling utility — 220

Output routine — 350–380

*Essential statements for Assembly programming.

The Development ROMs provide the following BASIC statements and functions:

ISOURCE — allows the programmer to write Assembly Language source statements that are integrated within the framework of a BASIC language program.

IASSEMBLE — reduces source code to object code.

IBREAK — allows breaking (pausing) at either a data or program location; allows transfer at break time to a BASIC subprogram and then resumption of the Assembly subprogram; provides 8 independent break points.

INORMAL — discontinues conditions set up by IBREAK.

IPAUSE ON/OFF — allows or disallows STEP and PAUSE to operate normally within an assembled routine; when STEPping through a routine, if the source is present, the Assembly Language instruction and associated comment is displayed.

IDUMP — allows printing of memory location in any of 5 formats: binary, octal, decimal, hexidecimal and ASCII.

ICHANGE — changes the content of a memory location to a specified value.

IADR — returns the value of a symbol, usually an address.

IMEM — returns the content of a memory location.

OCTAL — converts a decimal expression to its octal image.

DECIMAL — converts an octal expression to its decimal image.

The Execution ROMs support these BASIC statements:

ICOM — sets aside special Read/Write area in memory to accept the output of the assembler.

ICALL — transfers processor control to an assembled subprogram; allows the passing of parameters between BASIC programs and Assembly subprograms.

ISTORE — allows storage of object modules on mass storage devices.

ILOAD — allows retrieval of object modules from mass storage devices.

IDELETE — deletes selected Assembly Language modules.

ON/OFF INT# — establishes or discontinues end of line branch condition for ISRs.

Utilities. A number of utilities have been provided in the System to help make programming tasks easier and to give you direct access to some of the operating system's capabilities and routines. Such capabilities as basic arithmetic operations between full precision numbers, conversions between number types, storage and retrieval of string and numeric variables, mass memory Read/Write record and unformatted printing are provided in utility form.

The Language

The language supported by the System 35/45 processors and assemblers is extensive. Here is a listing of mnemonics, a brief description of the function and typical execution times in microseconds (numbers in parenthesis).

LOAD/STORE FUNCTIONS

LDA (LDB) Load register A (or B) with the content of a specified location (specified by an operand, the syntax of which is not detailed here). (2.2)

STA (STB) Store content of register A (or B) in specified location (2.2)

CLR Clear the specified number of words, beginning at the location pointed at by the A register. (6.7)

XFR Transfer the specified number of words, from the location starting at the address pointed at by the A register to the location pointed at by the B register. (11.5)

INTEGER MATH FUNCTIONS

ADA (ADB) Add the content of the specified location to register A (or B). (2.2)

TCA (TCB) Perform a two's complement of the A (or B) register. (1.5)

MPY Integer multiply. (13.5)

BRANCH FUNCTIONS

JMP Jump to specified location. (1.3)

JSM Subroutine jump to specified location. (2.2)

RET Return from subroutine (2.7)

TEST/BRANCH FUNCTIONS

CPA (CPB) Compare A (or B) to the content of the specified location skip if unequal. (2.7)

SZA (SZB) Skip to specified location if register A (or B) is 0. (2.3)

RZA (RZB) Skip to specified location if register A (or B) is not 0. (2.3)

SIA (SIB) Skip to specified location if register A (or B) is 0; then increment register A (or B) by 1. (2.3)

RIA (RIB) Skip to specified location if register A (or B) is not 0; then increment register A (or B) by 1. (2.3)

TEST/ALTER/BRANCH FUNCTIONS

ISZ (DSZ) Increment (decrement) content of specified location, skip if new content is 0. (3.2)

SAP (SBP) Skip to specified location if the A (or B) register is positive, i.e. bit 15 is 0. (2.3)

SAM (SBM) Skip to specified location if the A (or B) register is negative, i.e. bit 15 is 1. (2.3)

SLA (SLB) Skip to specified location if the least significant bit of register A (or B) is 0. (2.3)

RLA (RLB) Skip to specified location if the least significant bit of register A (or B) is 1. (2.3)

SOC (SOS) Skip to specified location if Overflow is clear (or set). (2.3)

SEC (SES) Skip to specified location if Extend is clear (or set). (2.3)

SHIFT/ROTATE FUNCTIONS

SAL (SAR) Shift the A register left (or right) the indicated number of bits with all vacated positions becoming 0. (2.8)

SBL (SBR) Analogous to SAL (SAR) using B register. (2.8)

AAR (ABR) Shift the A (or B) register right the indicated number of bits with the sign bit filling all vacated bit positions. (2.8)

RAL (RAR) Rotate the A register left (or right) the indicated number of bits; bit 15 will rotate into bit 0 (left shift) or bit 0 will rotate into bit 15 (right shift). (2.8)

RBL (RBR) Rotates the B register analogous to RAL (RAR). (2.8)

LOGICAL FUNCTIONS

AND Logical AND between register A and specified location, result in register A. (2.2)

IOR Inclusive OR between A register and specified location, result in register A. (2.2)

CMA (CMB) Perform a one's complement of the A (or B) register. (1.5)

STACK FUNCTIONS

PWC (PWD) Push the specified register (full word) onto the stack pointed at by the C (or D) register. (3.8)

PBC (PDB) Push the lower byte (right half) of the specified register onto the stack pointed at by the C (or D) register. (3.8)

WWC (WWD) Withdraw a full word from the stack pointed at by the C (or D) register and place it in the specified register. (3.8)

WBC (WBD) Withdraw a byte from the stack pointed at by the C (or D) register and place it in the lower byte (right half) of the specified register.

CBL (CBU) Clear (or set) the Cb register (C and Cb together act as a 17-bit address for PBC and WBC). (2.0)

DBL (DBU) Clear (or set) the Db register. (2.0)

BCD MATH FUNCTIONS

MRX (MRY) Mantissa right shift on Ar1 (or Ar2), a special BCD floating point machine register. (11.0 for MRX, 6.2 for MRY)

MLY Mantissa left shift on Ar2 for one digit. (5.3)

DRS Mantissa right shift of Ar1 for one digit. (9.3)

NRM Normalize the Ar2 mantissa. (4.8)

CMX (CMY) Ten's complement of Ar1 (9.8) or Ar2. (3.8)

FXA Fixed point addition; the mantissas of Ar1 and Ar2 are added together and the result placed in Ar2. (6.7)

MWA Mantissa word addition of B to Ar2. (4.7)

FMP Fast BCD multiply. (24.3)

FDV Fast BCD divide. (23.5)

CDC Clear decimal carry. (1.8)

SCD (SDS) Skip to specified location if decimal carry is clear (or set). (2.3)

I/O FUNCTIONS

SFC (SFS) Skip to specified location if I/O Flag line is false (or true). (2.3)

SSC (SSS) Skip to specified location if I/O Status line is clear (or set). (2.3)

EIR (DIR) Enable (disable) the interrupt system. (2.0)

SDI (SDO) Set DMA inwards (or outwards); reads from peripheral (or memory), writes to memory (or peripheral). (2.0)

DMA Enable the DMA mode. (2.0)

DDR Cancel the DMA instruction. (2.0)

MISCELLANEOUS FUNCTIONS

NOP Null operation. (2.2)

EXE Execute the contents of any of the first 32 registers; the operand specifies which register. (1.3)

Ordering Information

SYSTEM 35

Assembly Execution & Development ROMs	98339A
Execution ROM	98338A

SYSTEM 45

Assembly Execution & *Development* ROM	98439A or Opt 439
Execution ROM	98438A or Opt 438

SUPPLIED ACCESSORIES

	System 35	System 45B
Quick Reference Manual	09835-90080	09845-91080
Execution ROM Manual	09835-90082	09845-91082
Development ROM Manual	09835-90083	09845-91083
Demonstration Cartridge	11141-10154	11141-10155
BASIC Language Interfacing Concepts Manual	09835-90600	09835-90600

Scope of the System. The System 35/45 Assembly Language capability is limited to 32K words of assembled code and data storage space. It can significantly increase the speed of most, but not necessarily all, System 35/45 BASIC Language programs depending upon the tasks to be performed.

Writing Assembly Language code requires more development time than an equivalent BASIC routine. Also, Assembly Language capability may not be available on other HP Desktop Computers. Therefore, it is recommended that this System be used only in speed critical areas where this additional development time can be afforded.

Hewlett-Packard will support the System 35 and 45B Assembly Language and the BASIC language extensions listed in this brochure, the I/O structure, and the operating system details described in the Assembly Language Manuals. Hewlett-Packard will not support modifications of the operating systems or accesses to them (other than those provided by the Utilities).

For assistance call the HP regional office nearest you: Eastern 301/258-2000, Western 213/877-1282, Midwest 312/255-9800, Southern 404/955-1500, Canadian 416/678-9430. Ask for an HP Desktop Computer representative. Or write to Hewlett-Packard, 3404 East Harmony Road, Fort Collins, Colorado 80525.

Printed in U.S.A.
12/79

HEWLETT PACKARD

5953-1091