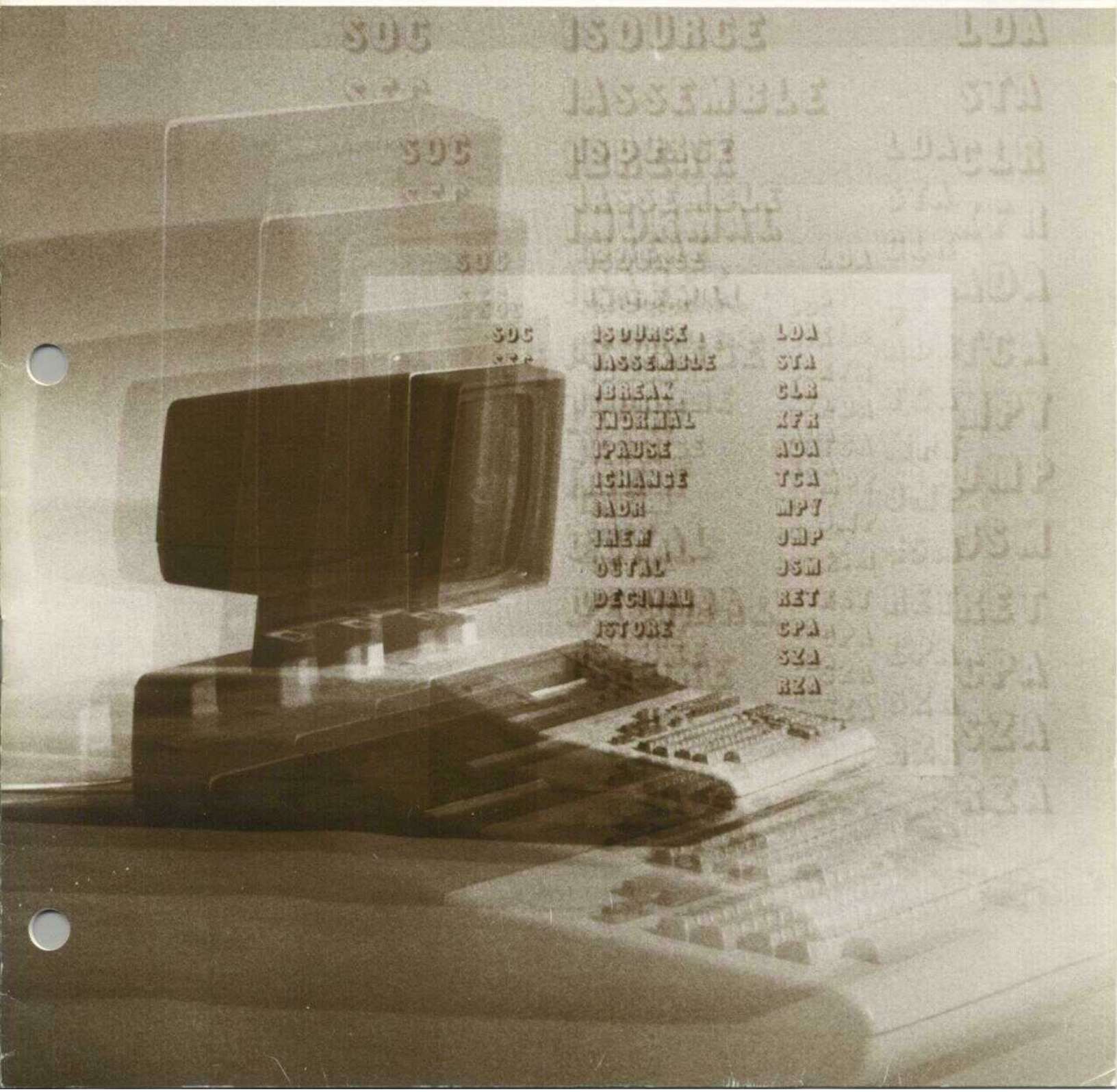
HP 9835 and 9845 Assembly Language

HEWLETT PACKARD

A Technical Supplement for Hewlett-Packard Computer Systems



Desktop computers that use interpretive operating systems and an enhanced version of BASIC language can provide highly flexible and easy-to-program computing solutions. In these systems, high level BASIC statements are entered via the desktop's integrated keyboard or from a mass storage device. The statements are then syntaxed and stored as internal code. This code is interpreted, line-by-line, into machine (low level) language instructions which are then executed. For most applications, this procedure produces sufficient speed and performance. However, in speed-critical situations such overhead is unacceptable. In these areas, a computer that uses a low level language can increase computation, graphics and I/O speeds dramatically since it bypasses the interpreter by creating object code ready for final assembly into machine code.

Many applications require both these capabilities; the ease of programming and friendliness of an interpretive system combined with the ability to call upon the speed and performance of a low level language system.

To meet the demands of these applications, we have implemented a ROM-based Assembly Language programming system for the System 35 and 45 Desktop Computers that allow you to call Assembly Language subprograms from a BASIC program. These Assembly instructions, pseudo-instructions and extensions to the BASIC language offer the programmer control over the System 35 and 45 CPUs and, at the same time, provide the power and flexiblity of BASIC. The system is totally integrated and includes powerful debug tools and a software Programmer Utility to aid in program development.

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;
- a software-based Programmer Utility that can be used to translate more than 75 Integer BASIC commands;
- 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 precision, as seen in the table below.

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.

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. 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 graphic cursors. The major advantage of using Assembly Language rather than BASIC to create and manipulate graphics information is speed. Graphical data can quickly be manipulated and input information plotted 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.

The Processors

The CPUs used on 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) capablility,
- 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 processor 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 compatibility 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 ISRs 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.

The 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.

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 exression 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 translation of BASIC statements into Assembly Language routines, 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 (2.4).
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 to the 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 register A 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 (PBD) 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 (3.8).
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 of 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 mantissa 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 Enables the DMA mode (2.0).
DDR Cancel the DMA instuction (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).

Assembly Programmer Utility. Another powerful assembly language tool.

The optional Assembly Language Programmer Utility software translates BASIC statements (source statements) into Assembly routines that are then saved in a target object data file. The Utility uses source statements entered from the keyboard or from a mass storage data file and then translates them using a system library of Assembly subroutines. The translated code and all necessary system routines are saved in the object file (see Fig. 1).

```
10      ! This program outputs a command to an HPIB device and
20      ! then inputs 2 data bytes and prints them 20 times.
30      INTEGER A,B,C,D     ! All variables must be declared as
40                          ! integer.
50      OUTPUT 724 USING "#,K";"R2T3F2D.5000SN20S"
60                          ! Output the string to HPIB without
70                          ! leading and trailing blanks and
80                          ! without a carriage-return/line-feed
90                          ! terminator.
100     TRIGGER 7           ! Trigger the HPIB device.
110     FOR A=1 TO 20       ! Loop 20 times, and
120       B=READBIN(724)    ! read 2 binary data bytes
130       C=READBIN(724)    ! from the device.
140       PRINT B;C         ! Print the data to the
150                         ! current printer.
160     NEXT A              ! Continue the loop.
170     BEEP                ! Beep when the loop is done.
180     END                 ! End the program.
```

Figure 1. *Source program example.*

The Programmer Utility can be used to translate more than 75 Integer BASIC commands. (i.e. real and string variables cannot be translated). However, due to size limitations, this is only a subset of the total BASIC integer, I/O and graphics commands available in the System 35 and 45. With these limitations in mind, the programmer can develop and debug programs in a high level, flexible language and then translate time-critical portions of his program into Assembly routines. These routines can then be linked together with the main BASIC driver resulting in low-level language speed being placed exactly where it's needed.

Translatable Commands

This Utility cannot be used as a general purpose Assembly Language program generator because only a portion of the available BASIC commands can be translated. The experienced BASIC user should be able to recognize those areas where the Utility can and cannot be used. Experienced Assembly Language programmers will be able to use the Utility to incorporate even more capabilities into their programs. Integer arithmetic (16 bit precision), logical (AND, OR, etc.), I/O and graphics commands that are translatable by this Utility are listed below. NOTE: an asterisk signifies that the full implementation of the command is NOT provided.

BASIC

BEEP	ON GOSUB	–	< >
LET	ON GOTO	X	< =
!	OPTION BASE	DIV	<
REM	PAUSE	↑	>
CALL*	PRINT*	ABS	> =
DISP*	PRINT	SGN	AND
END	RETURN	SQR	OR
FOR-NEXT*	STOP	CHR$*	NOT
GOSUB	SUB*	INT	
GOTO	SUBEND	NUM*	
ICALL*	SUBEXIT	ROW	
IF-THEN	WAIT	COL	
INTEGER	+	=	

I/O

CLEAR	WAIT WRITE
ENTER*	BINAND
OUTPUT*	BINCMP
READ IO	BINIOR
SENDBUS	BIT
STATUS	DECIMAL
TRIGGER	SHIFT
WRITE BIN	ROTATE
WRITE IO	IOFLAG
WAIT READ	IOSTATUS

GRAPHICS (available only on the System 45)

DRAW	PLOT
EXIT GRAPHICS	PLOTTER IS*
FRAME	POINTER
GCLEAR*	LINETYPE*
GRAPHICS	
IPLOT	
MOVE	
PEN	
PENUP	

In addition to the above commands, the Utility contains a set of directives that can be included as comments in the BASIC line to control translator flow. Directives are not required, but they provide increased power and efficiency for the user who is familiar with Assembly. The following directives are available:

$COMPON / $COMPOFF – Enables/disables the translator for the current BASIC source line. If disabled, the line is not included even as a comment in the destination object file.

$BASON / $BASOFF – Enables/disables transfer of BASIC source lines to the Assembly program without translating the lines.

$OVFON / $OVFOFF – Enables/disables integer overflow checking routines.

$IGN – Ignores the BASIC source line for translation but includes it as a comment in the Assembly code.

$IMG – Transfers the ISOURCE Assembly code in the source program to Assembly code in the destination program even if it was a comment in the source program.

$REGSUP – Supresses generation of code to evaluate a select code. This eliminates identical code when several I/O operations are directed to the same select code.

$ISR – Allows a BASIC subprogram to be an interrupt service routine.

$COMDEF – Defines variables that will be common to Assembly modules.

$COMUSE – Allows access to variables defined by $COMDEF.

$HPIB – Allows use of HPIB I/O drivers if HPIB has not been specified in the I/O source statement.

$CALPARA – Allows up to 2 parameters to be passed in an ICALL statement in the source program.

Assembly Programmer Utility System Configuration

The 9835B cannot be used with the Assembly Programmer Utility software because it does not have a CRT. Graphics commands must be directed to the CRTs of the 9845B/T or 9845C. The following hardware is required for the efficient operation of the translator software:

9835A	192k bytes r/w memory (Opt. 202) I/O ROM (98332A) Assembly Execution and Development ROMs (98339A) Assembly Language Programmer Utility software (09835-10260) or 9835A, Opt. 110 (Application System with Opt. 201) Assembly Language Programmer Utility (09835-10260)
9845B/C Model 150 Model 250	187k bytes r/w memory (Opt. 204) I/O ROM (98412A) Assembly Execution & Development ROMs (98439A) Assembly Language Programmer Utility (09845-10260) Graphics ROM (98411A/B) optional

Assembly Language System Ordering Information

System 35	
Assembly Execution & Development ROMs	98339A
Execution ROM	98338A
Assembly Programmer Utility software (optional)	09835-10260
System 45	
Assembly Execution & Development ROM	98439A or Opt.439
Execution ROM	98438A or Opt. 438
Assembly Programmer Utility software (optional)	09845-10260

Supplied Accessories

	System 35	System 45
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	09845-90600
Assembly Language Training Course	09835-30020	

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. It depends upon the tasks to be performed.

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

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

For assistance write Hewlett-Packard, 3404 East Harmony Rd, Fort Collins, Colorado 80525; in Europe, Hewlett-Packard GmbH, Desktop Computer Division, Herrenberger Strasse 110, D-703 Boeblingen, Postfach 1430, West Germany; elsewhere in the world, Hewlett-Packard Intercontinental, 3495 Deer Creek Rd, Palo Alto, California 94304.

Printed in USA
3/81

5953-4570