Honeywell

# Series 16 Software



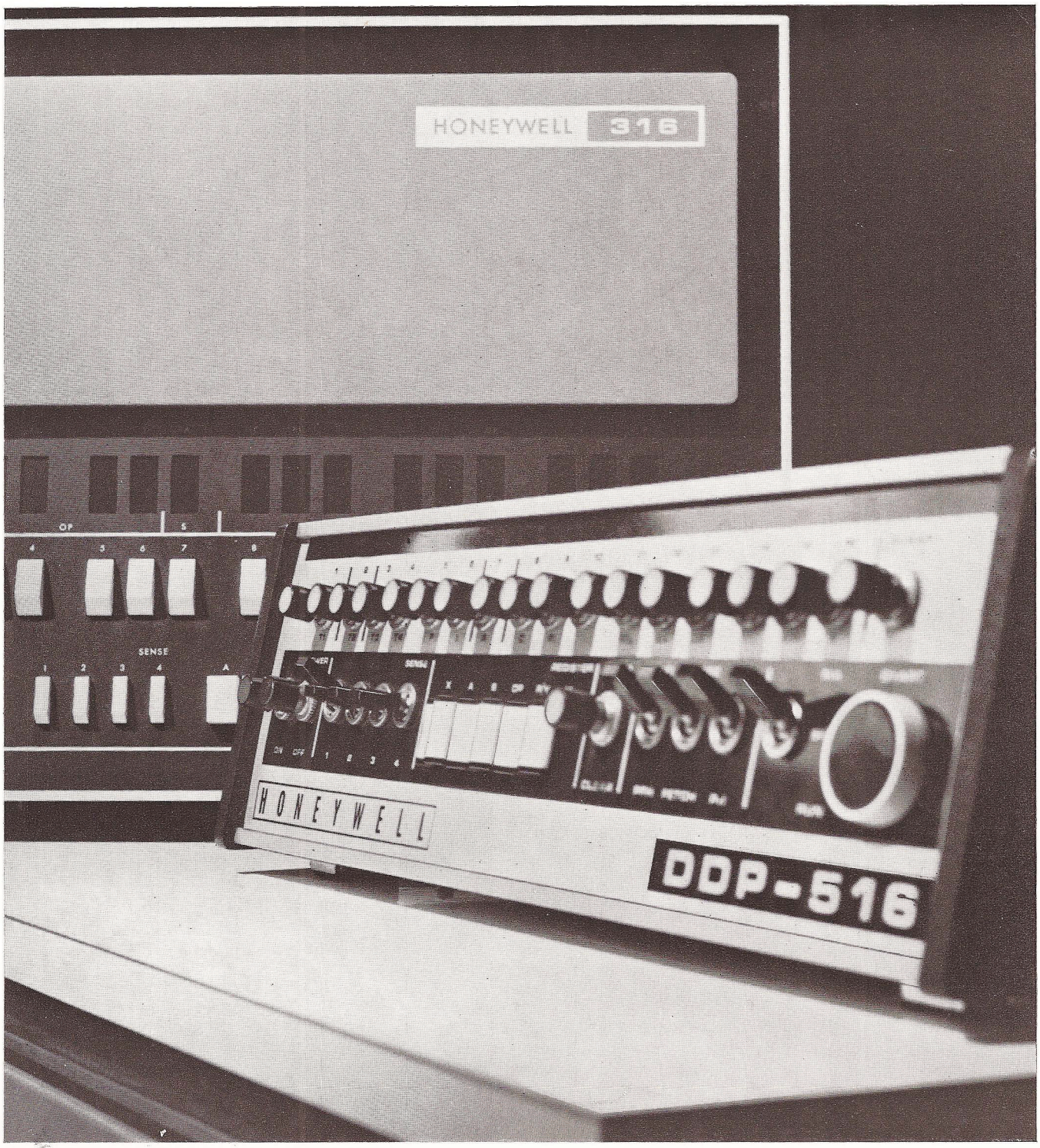
# Series 16 Software

# Contents

Kd-Data AB
Streteredsvägen 6, 430 50 Kållered
Tel: 031-75 14 13, Postgiro 42 07 05 - 6


| | page |
|---|---|
| Introduction | 2 |
| Data Representation and Addressing Modes | 4 |
| Symbolic Assembler DAP-16 | 6 |
| Fortran IV Compiler | 11 |
| System Library | 12 |
| Operating Systems | 22 |
| Batch Operating System BOS | 22 |
| Executive 16 EXEC. 16 | 24 |
| Interpretive Executive INTEX | 27 |
| On-Line Executive for Real-Time OLERT | 29 |
| Digital/Analogue Simulation MIDAS | 37 |


Honeywell reserve the right to amend the contents without prior notice


HONEYWELL 316
OP
SENSE
HONEYWELL
DDP-516

# Introduction

In selecting a computer system purchasers are now critically examining the accompanying software, for there is more to choosing a computer than just evaluating hardware performance specifications.

Honeywell invite you to study Series 16 system and utility software as summarised in this brochure. Series 16 was launched some six years ago with the introduction of the DDP-116 computer and with over one thousand 16-bit computers now installed, the purchaser of a 316, 416 or 516 computer inherits comprehensive field proven software.

The Honeywell systems available enable the Series 16 computer range to meet the requirements of several differing environments:

On-Line Real-Time.

General-Purpose Scientific Processing.

Multi-programming.

*and in three distinct configurations*

stand-alone.

batch operating.

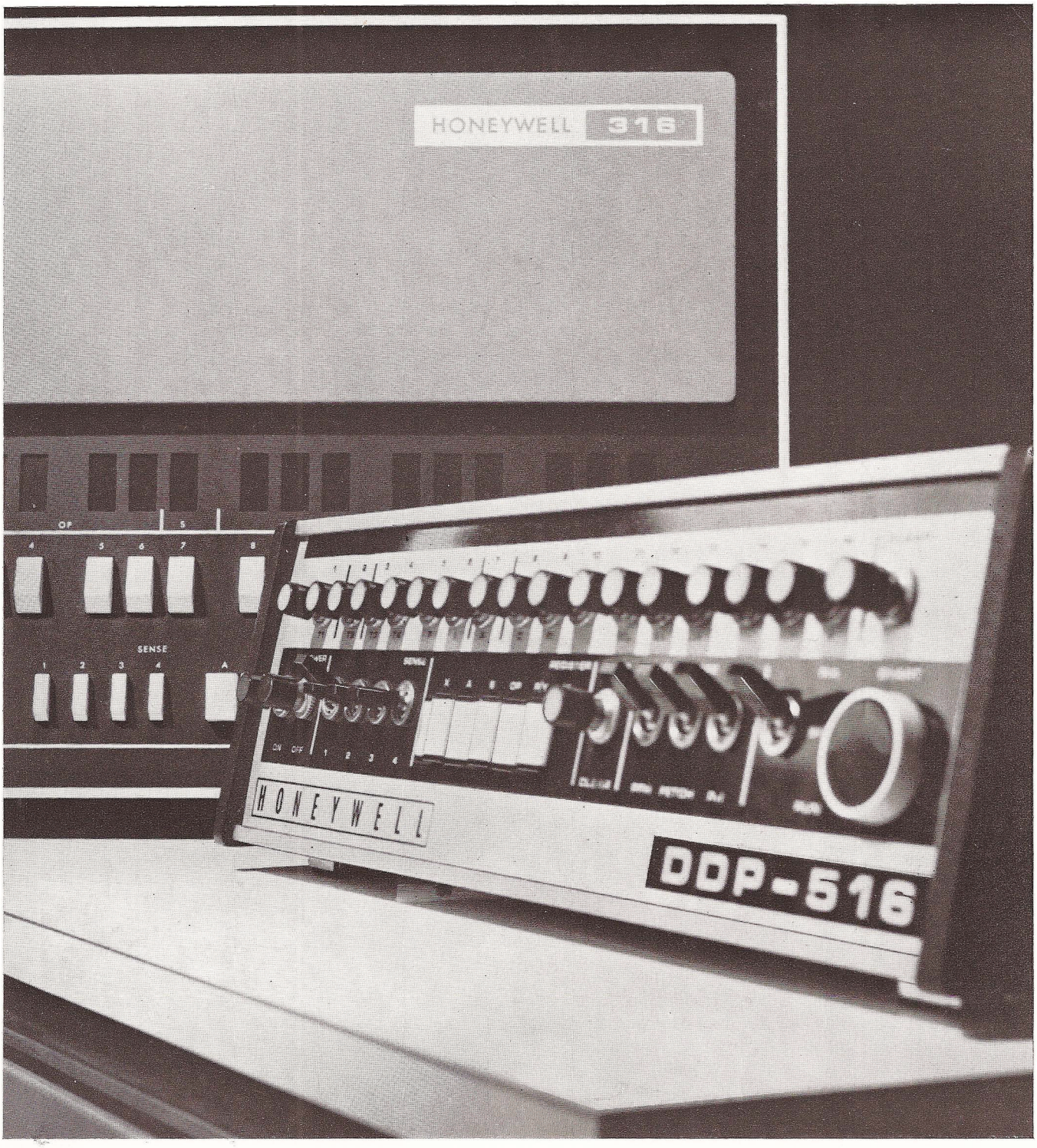
real-time executive control.

The Honeywell stand-alone systems operate in 4096 words of core memory without backing store except for Fortran IV which requires an 8K word memory store.

Series 16 Batch Operating System configurations are flexible and easier to use than 'stand-alone' requiring an 8196 word H316 or 516 central processor unit with backing storage in the form of magnetic discs.

Real-Time executive configurations for use on Honeywell Series 16 computers depend on the sophistication of the particular executive used, hardware requirements varying from a 4K H316, 416 or 516 central processor unit to a system in excess of 16K of memory available on the Honeywell 516 central processor. The hardware requirements for running a Real-Time executive system are specified in the description of the particular executive included in this software manual.

At the end of each section is a summary. This short description gives the computer model and core storage required, together with the availability of the software packages concerned.


HONEYWELL 316
OP
SENSE
HONEYWELL
DDP-516

# Data Representation and Addressing Modes

**Instruction Format**
Each instruction as held in a Honeywell Series 16 computer consists of a 16-bit word. Formats of these 16-bits vary depending on the type of instruction. Fig. 1.

**Sector Addressing**
When the sector flag is a 1-bit the address portion of the instruction refers to the same sector as that addressed by the program counter; when sector flag is zero the address portion refers to sector zero. Memory structure is divided in sectors of 512 words each. The memory reference instruction may only reference directly other words in the same sector as the instruction itself is located or words in the base sector. The base sector is usually sector zero (locations 000-777) but the memory lockout option includes base sector relocation enabling the user to specify another sector to act as base sector.

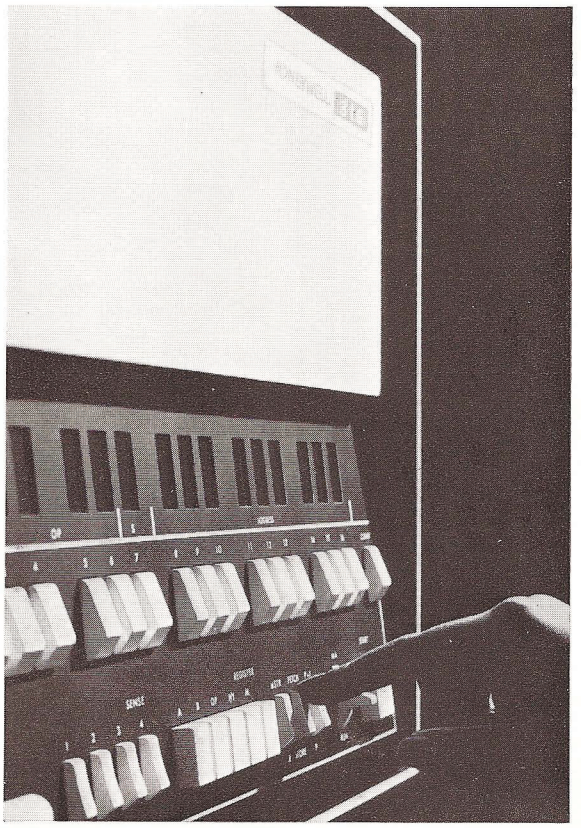
**Indirect Addressing**
When an instruction references a memory word outside the current sector (other than the base sector) indirect addressing is employed. Series 16 software allows the user to imagine that memory up to 32K words are directly addressable. All the desectorising of programs to conform to the 512 words sector boundaries are automatically handled by software which imposes indirect addressing on all memory reference outside the current sector boundaries.
Execution of these indirectly addressing instructions requires a one memory cycle addition for instruction execution. Indirect addressing may be chained to any level.

**Indexing**
When the index bit is set, the contents in the index register is added to the effective addressing of the instruction producing a new effective address. If indexing is specified in a given instruction, it occurs before indirect addressing. When indirect addressing is used, no additional time is required for instruction execution.

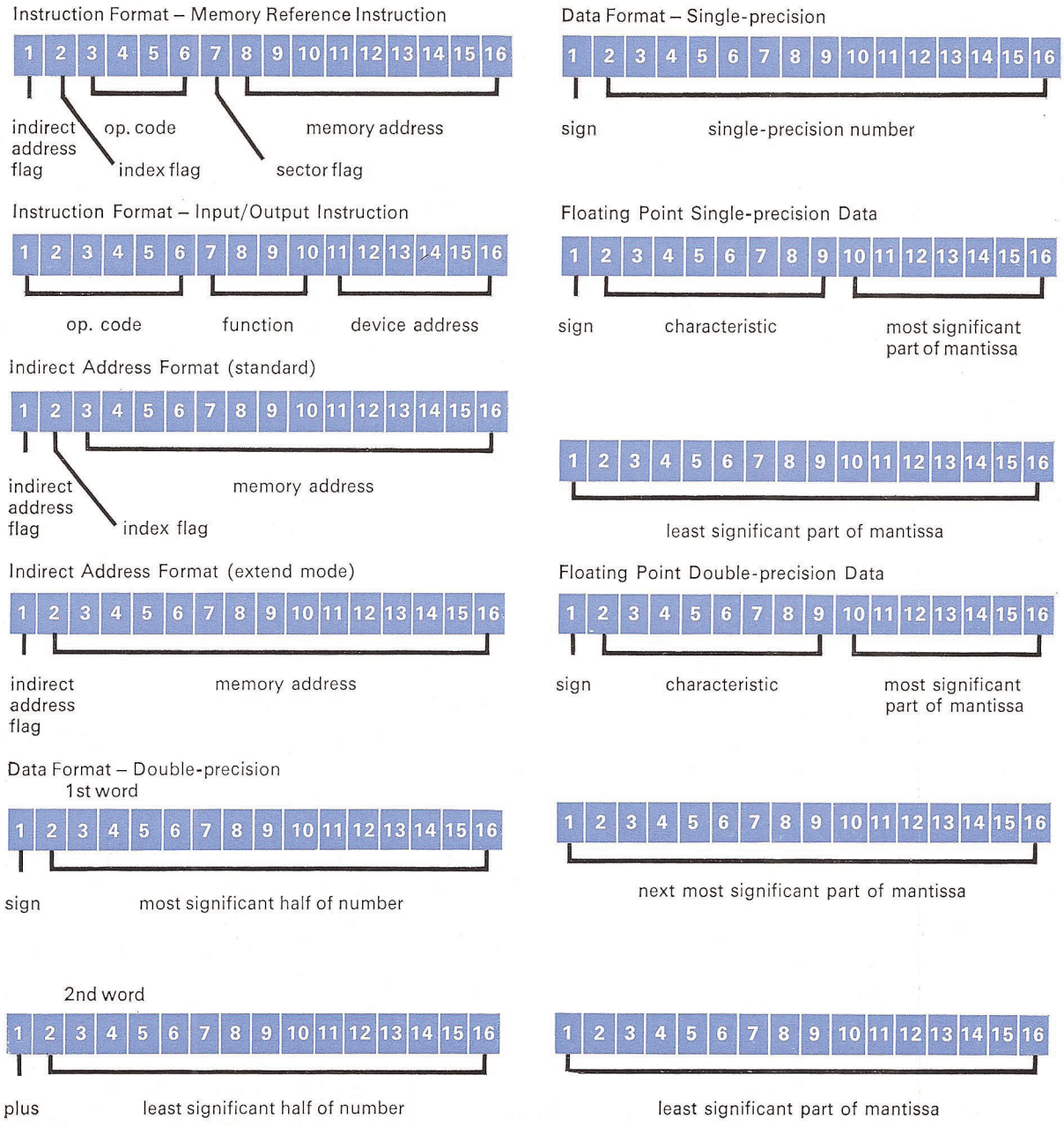
**Extended Mode**
Honeywell DDP-516 computer systems with 24K or 32K word memories are equipped with bank-switching logic whereby extended mode operation is included. When in the extended mode, the indirect address format includes 15 address bits in order to access 32K, indexing is specified in the instruction and is complete after indirect addressing. In this mode indexing is specified in the instruction and is applied after indirect addressing.

**Fixed Point**
Data is represented in two's-complement form, with the sign in the most significant position followed by 15 magnitude bits. Single-precision fixed point values thus range from $-32768$ to $+32767$. While this is adequate for most applications the Honeywell Series 16 range of computers offers both hardware and software double-precision capabilities for users who require 30 bit accuracy.

**Floating Point**
Used in conjunction with numerous floating point subroutines in the Series 16 program library, floating point capabilities include both single- and double-precision accuracies. Convenient and fast, these routines offer the flexibility of either $7_{10}$ or $12_{10}$ digit precision for number ranges of $10^{\pm 38}$.



# Fig. 1 Word Formats



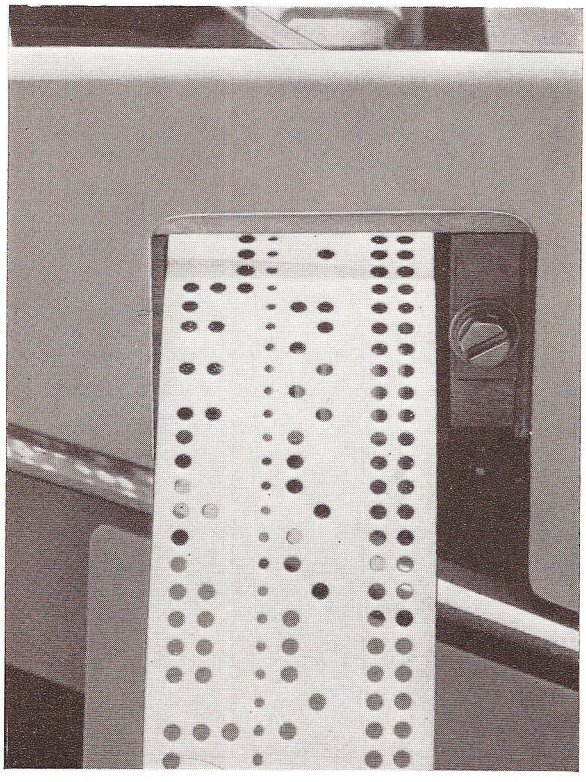
# Symbolic Assembler DAP–16

DAP-16 is the standard symbolic language assembly program for the Honeywell Series 16 range of computers. DAP-16 is both a programming language and a language processor. The Series 16 processor accepts as input a program coded in the DAP-16 language, processes it, and outputs a machine language object program and an assembly listing.

DAP-16 provides the user with a symbolic assembler ideally suited to programming the Series 16 range of computers. This Honeywell assembler allows the programmer to ignore addressing modes and to program the Series 16 processors as though the entire core memory was directly addressable. This ease of programming relieves the programmer from sector addressing and the indirect address linking necessary to cross-reference from one sector to another (Fig. 2).

**Programming Features**

DAP-16 is a programming aid that translates a symbolic (source) program into machine language (object) code.



*DAP-16 provides the following features:*

Enables symbolic programming while maintaining the characteristics, flexibility, speed and conciseness of machine language programming.

Permits the assignment of symbolic addresses to core memory storage locations.

Permits numerous pseudo-operations to supplement the standard computer instruction repertoires.

The pseudo-operations allow the programmer to express concepts that have no counterpart in machine language. Among the capabilities of the pseudo-operations are programmer defined assembly and loader controls, data definitions and program linkages:

Allows operation in either a one-pass or two-pass mode.

Assembles programs that take advantage of the extended addressing, memory lockout, memory priority and double-precision arithmetic options.

*DAP-16 incorporates the following features:*

Employs an input/output selector (IOS) concept for input/output device selection. (Preselected input/output for systems with only 4K of memory.)

Provides a pool table for storage of symbols and literals, thereby avoiding fixed-length tables.

Allows alphanumeric literals.

Allows compound expressions in the variable field.

Prints out and assigns storage for undefined symbols.

Flags illegal instructions and coding errors.

Allows single- or double-precision fixed or floating point constants.

Eight pseudo-operations are provided to specify assembly control, and are used to start and stop the program assembly, and to select the assembly model, e.g.

*REL Pseudo-Operation.*

The REL (relocatable) pseudo-operation is used to direct DAP-16 to assemble the subsequent instructions in the Relocatable mode. The effect of the REL pseudo-operation is to cause DAP-16 to assign relative locations to the instructions assembled.

*ORG Pseudo-Operation.*
The ORG (origin) pseudo-operation sets the location counter to a specified value, determined by the value of the expression in the variable field.

*CFx Pseudo-Operation.*
The CFx (configuration) pseudo-operation is used to inform the assembler as to which computer in the Series 16 range the object program is to be executed on. The suffix 'x' has the following connotation: 1 for the DDP-116, 4 for the DDP-416, and 5 for the DDP-516 and 316. The CFx pseudo-operation causes the assembler to flag any instructions that are illegal for the object computer without interrupting the assembly.
Storage allocation is defined by four pseudo-operations, that enable the programmer to allocate core memory words for data storage or working space, e.g.

*BSS Pseudo-Operation.*
The BSS (block starting with symbol) pseudo-operation effects to increase the value of the location counter by the value of the expression in the variable field. If a symbol appears in the location field, it is assigned the value of the location counter before the increase.

*COMN Pseudo-Operation.*
The COMN (common) pseudo-operation is used for assigning absolute storage locations in upper memory. The effect of the COMN pseudo-operation is to cause the assembler to subtract the value of the expression in the variable field from the COMMON base and assign this value to the symbol in the location field. COMMON base is a user option. The COMN pseudo-operation establishes a common data pool that can be referenced by several programs.

Communication links between programs are provided by pseudo-operations CALL and SUBR.

*CALL Pseudo-Operation.*
The CALL (call) pseudo-operation directs DAP-16 to generate instructions that will transfer control to an external subroutine, named in the variable field.

*SUBR Pseudo-Operation.*
The SUBR (subroutine) pseudo-operation is used to define a DAP-16 subroutine, and to symbolically assign a name to the subroutine for external reference.

The CALL and SUBR pseudo-operations produce FORTRAN IV compatible subroutine linkages allowing user written assembly language subroutines to be called by Fortran programs. It also allows pre-prepared subroutines to be linked into the main program at load time avoiding the need to recompile subroutines.

**Summary**
Computers: 116, 316, 416, 516.
Core Requirements (min.): 4096 words.
Availability: released.
Free issue as standard software.

# Fig. 2 Mnemonic Code for H316, DDP416 and DDP516 Computers

| | | Time (μsec) | | | |
|---|---|---|---|---|---|
| Type | Mnemonic | H316 | DDP416 | DDP516 | Description |
| Load and Store | LDA | 3.2 | 1.92 | 1.92 | Load A |
| | LDX | 4.8 | — | 2.88 | Load Index |
| | IMA | 4.8 | — | 2.88 | Interchange Memory and A |
| | IAB | 1.6 | — | 0.96 | Interchange A and B |
| | CRA | 1.6 | 0.96 | 0.96 | Clear A |
| | STA | 3.2 | 1.92 | 1.92 | Store A |
| | STX | 3.2 | — | 1.92 | Store Index |
| Arithmetic | ADD | 3.2 | 1.92 | 1.92 | Add |
| | SUB | 3.2 | 1.92 | 1.92 | Subtract |
| | IRS | 4.8 | 2.88 | 2.88 | Increment, Replace and Skip |
| | AOA | 1.6 | — | 0.96 | Add One to A |
| Control | SSP | 1.6 | — | 0.96 | Set Sign Plus |
| | SSM | 1.6 | — | 0.96 | Set Sign Minus |
| | SMK | 3.2 | 1.92 | 1.92 | Set Mask |
| | CMA | 1.6 | — | 0.96 | Complement A |
| | CSA | 1.6 | — | 0.96 | Copy Sign and Set Sign Plus |
| | ACA | 1.6 | — | 0.96 | Add C to A |
| | SCB | 1.6 | — | 0.96 | Set C |
| | RCB | 1.6 | — | 0.96 | Reset C |
| | HLT | 1.6 | 0.96 | 0.96 | Halt |
| | NOP | 1.6 | 0.96 | 0.96 | No Operation |
| | ENB | 1.6 | 0.96 | 0.96 | Enable Program Interrupt |
| | INH | 1.6 | 0.96 | 0.96 | Inhibit Program Interrupt |
| | TCA | 2.4 | — | 1.44 | Two's Complement A |
| | CHS | 1.6 | — | 0.96 | Complement A Sign |
| Input-Output | OCP | 3.2 | 1.92 | 1.92 | Output Control Pulse |
| | SKS | 3.2 | 1.92 | 1.92 | Skip if Ready Line Set |
| | INA | 3.2 | 1.92 | 1.92 | Input to A |
| | INK | 1.6 | — | 0.96 | Input Keys |
| | OTA | 3.2 | 1.92 | 1.92 | Output from A |
| | OTK | 3.2 | — | 1.92 | Output Keys |
| Byte Manipulation | ICA | 1.6 | — | 0.96 | Interchange Halves in A |
| | ICL | 1.6 | — | 0.96 | Interchange/Clear Left Half of A |
| | ICR | 1.6 | — | 0.96 | Interchange/Clear Right Half of A |
| | CAL | 1.6 | — | 0.96 | Clear Left Half |
| | CAR | 1.6 | — | 0.96 | Clear Right Half |
| Logical | ANA | 3.2 | 1.92 | 1.92 | Logic AND |
| | ERA | 3.2 | 1.92 | 1.92 | Exclusive OR |
| Shift | LGL | 1.6 + 0.8n | 0.96 + 0.48n | 0.96 + 0.48n | Logical Left Shift |
| | LGR | 1.6 + 0.8n | 0.96 + 0.48n | 0.96 + 0.48n | Logical Right Shift |
| | ALR | 1.6 + 0.8n | 0.96 + 0.48n | 0.96 + 0.48n | Logical Left Rotate |
| | ARR | 1.6 + 0.8n | 0.96 + 0.48n | 0.96 + 0.48n | Logical Right Rotate |
| | ALS | 1.6 + 0.8n | 0.96 + 0.48n | 0.96 + 0.48n | Arithmetic Left Shift |
| | ARS | 1.6 + 0.8n | 0.96 + 0.48n | 0.96 + 0.48n | Arithmetic Right Shift |

| | | Time (μsec) | | | |
|---|---|---|---|---|---|
| Type | Mnemonic | H316 | DDP416 | DDP516 | Description |
| Shift *contd.* | LLL | 1.6 + 0.8n | — | 0.96 + 0.48n | Long Left Logical Shift |
| | LRL | 1.6 + 0.8n | — | 0.96 + 0.48n | Long Right Logical Shift |
| | LLR | 1.6 + 0.8n | — | 0.96 + 0.48n | Long Left Rotate |
| | LRR | 1.6 + 0.8n | — | 0.96 + 0.48n | Long Right Rotate |
| | LLS | 1.6 + 0.8n | — | 0.96 + 0.48n | Long Arithmetic Left Shift |
| | LRS | 1.6 + 0.8n | — | 0.96 + 0.48n | Long Arithmetic Right Shift |
| Transfer Control | JMP | 1.6 | 0.96 | 0.96 | Unconditional Jump |
| | JST | 4.8 | 2.88 | 2.88 | Jump and Store Location |
| | CAS | 4.8 | — | 2.88 | Compare |
| | SKP | 1.6 | 0.96 | 0.96 | Unconditional Skip |
| | SPL | 1.6 | 0.96 | 0.96 | Skip if A Plus |
| | SMI | 1.6 | 0.96 | 0.96 | Skip if A Minus |
| | SZE | 1.6 | 0.96 | 0.96 | Skip if A Zero |
| | SNZ | 1.6 | 0.96 | 0.96 | Skip if A Not Zero |
| | SLZ | 1.6 | — | 0.96 | Skip if ($A_{16}$) Zero |
| | SLN | 1.6 | — | 0.96 | Skip if ($A_{16}$) One |
| | SSC | 1.6 | — | 0.96 | Skip if C Set |
| | SRC | 1.6 | — | 0.96 | Skip if C Reset |
| | SS1 | 1.6 | — | 0.96 | Skip if Sense Switch No. 1 Set |
| | SS2 | 1.6 | — | 0.96 | Skip if Sense Switch No. 2 Set |
| | SS3 | 1.6 | — | 0.96 | Skip if Sense Switch No. 3 Set |
| | SS4 | 1.6 | — | 0.96 | Skip if Sense Switch No. 4 Set |
| | SR1 | 1.6 | — | 0.96 | Skip if Sense Switch No. 1 Reset |
| | SR2 | 1.6 | — | 0.96 | Skip if Sense Switch No. 2 Reset |
| | SR3 | 1.6 | — | 0.96 | Skip if Sense Switch No. 3 Reset |
| | SR4 | 1.6 | — | 0.96 | Skip if Sense Switch No. 4 Reset |
| | SSR | 1.6 | — | 0.96 | Skip if No Sense Switch Set |
| | SSS | 1.6 | — | 0.96 | Skip if Any Sense Switch Set |
| Optional | DLD | 4.8 | — | 2.88 | Double Load |
| | DST | 4.8 | — | 2.88 | Double Store |
| | DAD | 4.8 | — | 2.88 | Double Add |
| | DSB | 4.8 | — | 2.88 | Double Subtract |
| | MPY | 8.8 | — | 5.28 | Multiply |
| | DIV | 17.6 (max.) | — | 10.56 (max.) | Divide |
| | SGL | 1.6 | — | 0.96 | Enter Single Precision Mode |
| | DBL | 1.6 | — | 0.96 | Enter Double Precision Mode |
| | DXA | — | — | 0.96 | Disable Extended Addressing |
| | EXA | — | — | 0.96 | Enable Extended Addressing |
| | RMP | — | — | 0.96 | Reset Memory Parity Error |
| | SCA | 1.6 | — | 0.96 | Shift Count to A |
| | NRM | 1.6 + 0.8n | — | 0.96 + 0.48n | Normalize |
| | ERM | — | — | 0.96 | Enter Restricted Mode |
| | SPN | — | — | 0.96 | Skip on No Memory Parity Error |
| | SPS | — | — | 0.96 | Skip on Memory Parity Error |


HONEYWELL
DDP-516
HONEYWELL

# Fortran IV Compiler

This compiler is available as standard software for the Honeywell Series 16-116, 316 and 516 computers. FORTRAN IV is a one-pass compiler operating in a system with 8192 words (minimum) of core memory. This compiler provides all of the power and flexibility of FORTRAN IV as defined by the American Standards Association. It features the use of type statement, block data, double-precision, complex, logical and octal constants. From a programmer's point of view it can be said that FORTRAN IV retains all of the ease of programming and the clarity of FORTRAN II while adding the advantage of generality and flexibility.

**Fortran IV Features**

*Five Types of Data.*
Data may be integer, real, double-precision, complex, or logical. Real numbers have more than six digits of significance (as have the real and imaginary parts of complex numbers). Double-precision numbers have twelve digits, and logical variables can have the values. TRUE. and FALSE.

**Fortran IV Input/Output Statements**

| | |
|---|---|
| READ (unit, format) | list—formatted for binary |
| WRITE (unit, format) | list—coded decimal operations |
| READ (unit) | list—unformatted for binary |
| WRITE (unit) | list—operations |

FORTRAN IV Input/Output statements allow the user to program input/output without knowing what kind of device (for example, line printer and card reader) is involved. The user can postpone until execution time the decision of what device to use.

*Compiler Error Messages.*
The compiler detects an error in the format of a FORTRAN statement, an error message is printed in the listing below the statement in error. The coded two character error message is typed in approximately the same position as the error in the statement above.

*Three-Dimensional Arrays.*
Arrays can be one, two or three dimensions.

*Tracing.*
An additional TRACE statement is included in the FORTRAN language. There are two types: the first is used in tracing selected variables only, and the second is used in tracing all variables within a specified area.

$$\text{TRACE } X_1, X_2, X_3, \ldots X_m.$$

Where $X_1$ is any variable or array name. When any of the variables defined in the list become re-defined by an arithmetic statement, a line of trace information is typed, which specifies the name of the variable and its new value.

The format of the second type of TRACE statement is

TRACE N

where N is any statement label not yet executed. This type of TRACE statement causes the results of all arithmetic expressions (including IF statements) that follow the TRACE statement up to and including the statement labelled N, to output a line of trace information.

**Summary**
Computers: 116, 316, 516.
Core Requirements (min.): 8192 words.
Availability: released.
Free Issue.

# System Library

**Mathematical Library**
A library of standard mathematical functions usable by Fortran IV is provided. The library is written in DAP-16 assembly language. Standard mathematical routines available for single- and double-precision, fixed and floating point and complex calculations are indicated in the accompanying chart (Fig. 3). In addition to these standard mathematical functions a comprehensive package of routines is included in the library for the solution of mathematical problems. These include differential equations, statistical, matrix, and algebraic routines. A selection of these routines is shown in Fig. 4.

**Application Programs**
These programs will run on the 116, 316 and 516 Honeywell central processors. All calculations are carried out in single-precision floating point arithmetic, with 23 significant binary digits in the argument, plus a sign bit and an 8-bit characteristic.

MEAN, VARIANCE, T-RATIO
To compute the mean, variance and T-Ratio for two groups of data.
*Storage 626 (decimal).*

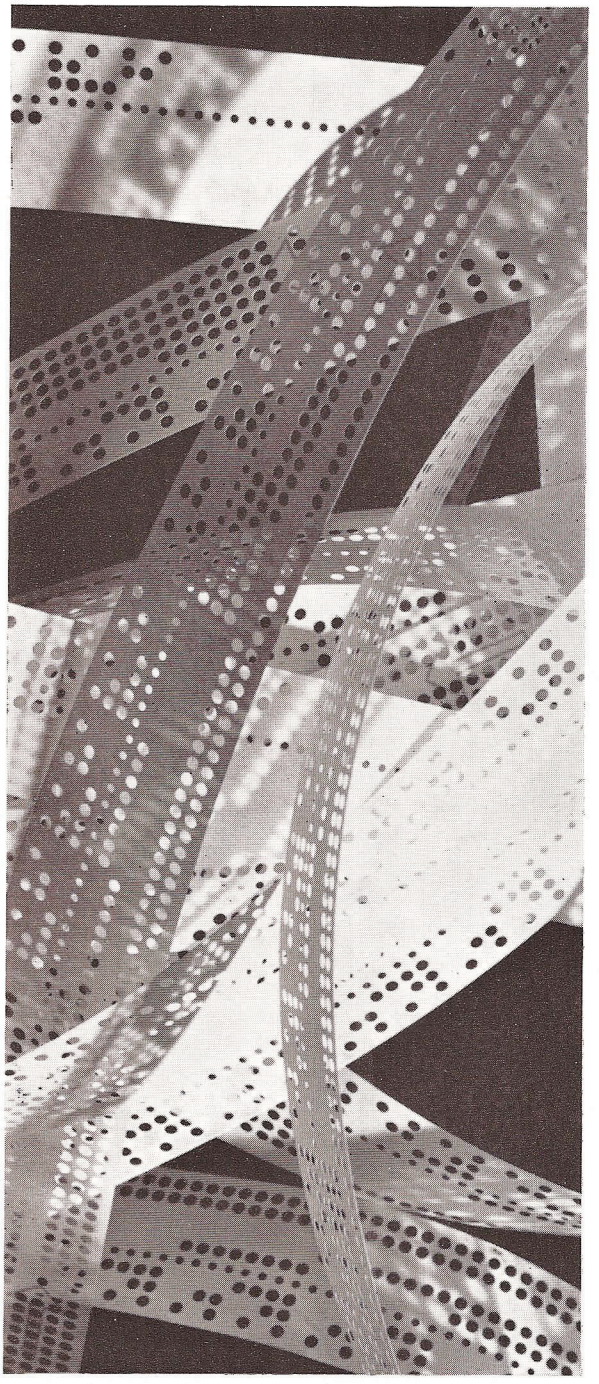
GEOMETRIC MEAN AND STANDARD DEVIATION
To compute the geometric mean and standard deviation for a geometrically normal series of data.
*Storage 134 (decimal).*

CORRELATION COEFFICIENT
To compute a correlation coefficient for N pairs of values.
*Storage 200 (decimal).*

SPEARMAN RANK CORRELATION COEFFICIENT
To compute the Spearman Rank correlation coefficient for two series of data.
*Storage 332 (decimal).*

PRODUCT MOMENT CORRELATION MATRIX
To compute the correlation matrix for N series of data and for each series the mean, variance and standard deviation.
*Storage 454 (decimal).*

RANDOM NUMBER GENERATION
To generate a sequence of random numbers.
*Storage 16 (decimal).*



## Fig. 3 Mathematical Library

| | | Fixed Point | | Floating Point | |
|---|---|---|---|---|---|
| Subroutines | Complex | Single-precision | Double-precision | Single-precision | Double-precision |
| Square Root | ● | ● | ● | ● | ● |
| Cos | ● | ● | ● | ● | ● |
| Sin | ● | ● | ● | ● | ● |
| Arc Tan | | ● | ● | ● | ● |
| Log Base$^{e}$ | ● | ● | ● | ● | ● |
| Log Base$^{2}$ | | ● | ● | | |
| Log Base $^{10}$ | | | | ● | ● |
| Exponential | ● | ● | ● | ● | ● |
| Add | ● | | ● | ● | ● |
| Subtract | ● | | ● | ● | ● |
| Multiply | ● | ● | ● | ● | ● |
| Divide | ● | ● | ● | ● | ● |
| Maximum Value | | ● | | ● | ● |
| Minimum Value | | | | ● | ● |
| Absolute Value | ● | ● | | ● | ● |
| Remaindering | | ● | | | ● |
| Hyperbolic Tan | | | | ● | |

MEDIAN TEST
To compare two groups of data using the median test to compute the Chi square value of the 2 by 2 table on one degree of freedom.
*Storage 274 (decimal).*

CHI SQUARE TEST FOR 2 BY 2 TABLE
To compute Chi square for 2 by 2 tables with Yates correction for continuity.
*Storage 110 (decimal).*

CHI SQUARE TEST FOR M BY N TABLE
To compute Chi square for M by N contingency tables.
*Storage 240 (decimal).*

BINOMIAL PROBABILITY DISTRIBUTION
To compute the binomial probability distribution.
*Storage 114 (decimal).*

LINEAR REGRESSION
To compute statistics relevant to the linear regression between two variables X and Y where Y may be either dependent or independent of X.
*Storage 532 (decimal).*

LEAST SQUARES REGRESSION
To fit a polynomial of degree N to M observations by the method of least squares.
*Storage 346 (decimal).*

ANALYSIS OF VARIANCE—ONE-WAY CLASSIFICATION
To use the data of a one-way, completely randomised experiment to compute the analysis of variance table and F-Ratio to test the significance of the difference between treatment means.
*Storage 282 (decimal).*

ANALYSIS OF VARIANCE—TWO-WAY CLASSIFICATION
To compute the analysis of variance for a two-way classification.
*Storage 408 (decimal).*

ANALYSIS OF VARIANCE—RANDOMISED COMPLETE BLOCK
To compute the analysis of variance for treatments and blocks of a randomised complete block design.
*Storage 420 (decimal).*

ANALYSIS OF VARIANCE—LATIN SQUARE
To compute the analysis of variance for a Latin square design.
*Storage 526 (decimal).*

ANALYSIS OF VARIANCE—GRAECO-LATIN SQUARE
To compute the analysis of variance for a Graeco-Latin square design.
*Storage 760 (decimal).*

ANALYSIS OF VARIANCE—BALANCED INCOMPLETE BLOCK
To compute the analysis of variance for treatments and blocks of a balanced incomplete block design.
*Storage 522 (decimal).*

ANALYSIS OF VARIANCE—YOUDEN SQUARE
To compute the analysis of variance for a Youden square design.
*Storage 618 (decimal).*

FIRST ORDER DIFFERENTIAL EQUATION—ADAMS' METHOD
To solve a first order differential equation.

FIRST ORDER DIFFERENTIAL EQUATION—RUNGE-KUTTA METHOD
To solve a first order differential equation $\dot{y} = F(x, y)$ using the Runge-Kutta method.
*Storage 174 (decimal).*

FIRST ORDER DIFFERENTIAL EQUATION—MODIFIED ADAMS' METHOD
To solve a first order differential equation $\dot{y} = F(x, y)$ using a modified Adams' method with starting values generated by the Runge-Kutta method.
*Storage 312 (decimal).*

FIRST ORDER DIFFERENTIAL EQUATION—HAMMINGS' METHOD
To solve a first order differential equation $\dot{y} = F(x, y)$ using Hammings' method with starting values generated by the Runge-Kutta method.
*Storage 304 (decimal).*

SECOND ORDER DIFFERENTIAL EQUATION—RUNGE-KUTTA METHOD
To solve a second order differential equation $\ddot{y} = F(x, y, \dot{y})$ using the Runge-Kutta method.
*Storage 364 (decimal).*

# Fig. 4 Selection of Mathematical Routines

| Function | Routine | Timing (μsec)** | Storage (decimal) |
|---|---|---|---|
| **Complex:** | | | |
| Absolute value | CABS | 2926.1 | 27 |
| Add | A$55 | 819.8 | 35 |
| Add single-precision argument | A$52 | 503.0 | 20 |
| Conjugate | CONJG | 307.2 | 30 |
| Convert imaginary part to real | AIMAG | 79.9 | 9 |
| Cosine | CCOS | 18067.2 | 14 |
| Divide | D$55 | 6047.0 | 87 |
| Divide by single-precision argument | D$52 | 2379.8 | 30 |
| Exponential, base e | CEXP | 14458.6 | 42 |
| Load | L$55 | 39.4 | 13 |
| Logarithm, base e | CLOG | 9478.1 | 52 |
| Multiply | M$55 | 2731.2 | 64 |
| Multiply by single-precision argument | M$52 | 1212.5 | 32 |
| Negate | N$55 | 345.8 | 25 |
| Raise to integer power | E$51 | 3094.1 | 41 |
| Sine | CSIN | 17047.7 | 59 |
| Square root | CSQRT | 7189.4 | 68 |
| Store (hold) | H$55 | 39.4 | 13 |
| Subtract | S$55 | 827.5 | 35 |
| Subtract single-precision argument | S$52 | 506.9 | 20 |
| **Double-precision:** *Floating point:* | | | |
| Absolute value | DABS | 173.8 | 11 |
| Add | A$66 | 345.6 | 553 |
| Add single-precision argument | A$62 | 653.8 | 12 |
| Add integer to exponent | A$81 | 28.8 | 9 |
| Arctangent, principle value | DATAN | 18778.6 | 134 |
| Arctangent x/y | DATAN2 | 23261.8 | 56 |
| Clear (zero, exponent) | Z$80 | 10.6 | 11 |
| Convert exponent to integer | C$81 | 12.5 | 8 |
| Convert to integer | C$61 | 268.8 | 4 |
| Convert to single-precision (from pseudo-accumulator) | C$62 | 6.7 | 5 |
| Cosine | DCOS | 14385.6 | 14 |
| Divide | A$66 | 4057.0 | 553 |
| Divide by single-precision argument | D$62 | 4441.9 | 16 |
| Exponential, base e | DEXP | 17441.3 | 103 |
| Load | L$66 | 35.5 | 11 |
| Logarithm, base e | DLOG | 15522.2 | 9 |
| Logarithm, base 2 | DLOG | 14327.0 | 83 |
| Logarithm, base 10 | DLOG10 | 15526.1 | 10 |
| Maximum value | DMAXI | 1790.4 | 35 |

**516 Timing only

| Function | Routine | Timing (μsec)** | Storage (decimal) |
|---|---|---|---|
| Minimum value | DMINI | 1791.4 | 36 |
| Multiply | A$66 | 1081.0 | 553 |
| Multiply by single-precision argument | M$62 | 1388.2 | 12 |
| Negate | N$66 | 22.1 | 21 |
| Raise to double-precision power | E$66 | 34239.4 | 16 |
| Raise to integer power | E$61 | 2113.9 | 38 |
| Raise to single-precision power | E$62 | 34545.6 | 16 |
| Remainder | DMOD | 6597.1 | 20 |
| Sine | DSIN | 13849.0 | 116 |
| Square root | DSQRT | 6108.5 | 24 |
| Store (hold) | H$66 | 35.5 | 11 |
| Subtract | A$66 | 412.8 | 553 |
| Subtract single-precision argument | S$62 | 745.9 | 13 |
| Transfer sign of second argument to first | DSIGN | 191.0 | 21 |
| Truncate fractional bits | DINT | 819.8 | 22 |
| **Integer:** | | | |
| Absolute value | IABS | 14.4 | 9 |
| Convert to double-precision | C$16 | 275.5 | 5 |
| Convert (FORTRAN-generated) to single-precision | FLOAT | 271.7 | 8 |
| Convert to single-precision | C$12 | 256.3 | 25 |
| Divide | D$11 | 218.9 | 34 |
| Maximum single-precision value | MAX0 | 336.0 | 34 |
| Maximum value | MAX0 | 57.6 | 34 |
| Multiply | M$11 | 147.8 | 19 |
| Positive difference | IDIM | 34.6 | 19 |
| Raise to integer power | E$11 | 405.1 | 80 |
| Remainder | MOD | 410.9 | 22 |
| Transfer sign of second argument to first | ISIGN | 34.6 | 21 |
| **Single-precision:** | | | |
| *Fixed point:* | | | |
| Arctangent | ATNX1 | 1057.0 | 36 |
| *Arctangent | ATNX2 | 146.9 | 33 |
| Cosine | COSX1 | 812.2 | 5 |
| *Cosine | COSX2 | 53.8 | 5 |
| Divide | DIV | 220.8 | 80 |
| Exponential, base e | EXEX1 | 838.1 | 75 |
| *Exponential, base e | EXEX2 | 110.4 | 73 |
| Exponential, base 2 | EX2X1 | 815.0 | 48 |
| *Exponential, base 2 | EX2X2 | 87.4 | 46 |
| Logarithm, base e | LGEX1 | 931.2 | 60 |
| *Logarithm, base e | LGEX2 | 111.4 | 59 |
| Logarithm, base 2 | LG2X1 | 728.6 | 37 |

*Operates with multiply/divide option only

**516 Timing only

| Function | Routine | Timing (μsec)** | Storage (decimal) |
|---|---|---|---|
| *Logarithm, base 2 | LG2X2 | 60.5 | 34 |
| Multiply | MPY | 154.6 | 74 |
| Round up binary number | ROND | 6.7 | 6 |
| Sine | SINX1 | 805.4 | 30 |
| *Sine | SINX2 | 47.0 | 25 |
| Square root | SQRX1 | 675.8 | 61 |
| *Square root | SQRX2 | 98.9 | 60 |
| *Floating point:* | | | |
| Absolute value | ABS | 53.8 | 9 |
| Add | A$22 | 238.1 | 184 |
| Arctangent, principle value | ATAN | 2332.8 | 228 |
| Arctangent, y/x | ATAN2 | 3647.0 | 228 |
| Convert (FORTRAN-generated) to double-precision | DBLE | 154.6 | 9 |
| Convert to integer or truncate fractional bits and convert to integer | IFIX | 293.8 | 8 |
| Convert pair to complex | CMPLX | 298.0 | 25 |
| Convert to complex format | C$25 | 327.4 | 9 |
| Convert to double-precision | C$26 | 10.6 | 8 |
| Convert to integer | C$21 | 252.5 | 24 |
| Divide | D$22 | 1021.4 | 295 |
| Exponential, base e | EXP | 4398.7 | 204 |
| Hyperbolic tangent | TANH | 5987.5 | 31 |
| Load | L$22 | 28.8 | 8 |
| Logarithm, base e | ALOG | 3883.2 | 188 |
| Logarithm, base 10 | ALOG10 | 3888.0 | 188 |
| Maximum integer value | MAX1 | 1488.0 | 46 |
| Maximum value | AMAX1 | 1201.9 | 46 |
| Minimum integer value | MIN1 | 1494.7 | 47 |
| Minimum value | AMIN1 | 1208.6 | 47 |
| Multiply | M$22 | 437.8 | 295 |
| Positive difference | DIM | 297.6 | 15 |
| Raise to double-precision power | E$26 | 34263.4 | 17 |
| Raise to integer power | E$21 | R²499.2 | 47 |
| Raise to single-precision power | E$22 | R²1522.6 | 29 |
| Remainder | AMOD | 2309.8 | 24 |
| Sine, Cosine | SIN, COS | 4442.9 | 140 |
| Square root | SQRT | 1545.6 | 71 |
| Store (hold) | H$22 | 33.6 | 13 |
| Subtract | A$22 | 241.9 | 184 |
| Transfer sign of second argument to first | SIGN | 71.0 | 20 |
| Truncate fractional bits | AINT | 531.8 | 24 |
| Twos complement | N$22 | 8.6 | 10 |

*Operates with multiply/divide option only

**516 Timing only

SECOND ORDER DIFFERENTIAL EQUATION—ADAMS' METHOD
To solve a second order differential equation $\ddot{y} = F(x, y, \dot{y})$ using Adams' method with starting values generated by the Runge-Kutta method.
*Storage 278 (decimal).*

SECOND ORDER DIFFERENTIAL EQUATION—HAMMINGS' METHOD
To solve a second order differential equation $\ddot{y} = F(x, y, \dot{y})$ using a modified Adams' method with starting values generated by the Runge-Kutta method.
*Storage 464 (decimal).*

TRANSPOSE A MATRIX
To transpose an N by N matrix.
*Storage 98 (decimal).*

MULTIPLICATION OF MATRICES
The multiplication of two-dimensional matrices.
*Storage 112 (decimal).*

LINEAR MATRIX ARITHMETIC
To perform matrix addition, subtraction and multiplication by a constant.
*Storage 138 (decimal).*

EIGEN VALUES AND EIGEN VECTORS
To calculate Eigen values and corresponding Eigen vectors of a real symmetric matrix.
*Storage 780 (decimal).*

EVALUATION OF A POLYNOMIAL
To evaluate a single variable polynomial of degree M.
*Storage 48 (decimal).*

MULTIPLICATION OF POLYNOMIALS
To multiply any two polynomials.
*Storage 104 (decimal).*

DIVISION OF POLYNOMIALS
To divide polynomials, retaining the remainder.
*Storage 316 (decimals).*

INTERPOLATION—AITKEN'S METHOD
To develop an interpolating polynomial according to Aitken's algorithm.
*Storage 168 (decimal).*

INTERPOLATION—LAGRANGIAN METHOD
To develop an interpolating polynomial by the Lagrangian method.
*Storage 96 (decimal).*

DIFFERENTIATION OF A POLYNOMIAL
To differentiate a polynomial of degree N.
*Storage 72 (decimal).*

INTEGRATION OF A POLYNOMIAL
To integrate any single variable polynomial of degree N.
*Storage 94 (decimal).*

REAL ROOT OF A POLYNOMIAL
To find one real root of a given polynomial using Newton's method.
*Storage 68 (decimal).*

ROOTS OF A POLYNOMIAL—BAIRSTOW'S METHOD
To find the real and/or complex roots of an Nth degree polynomial with real coefficients.
*Storage 744 (decimal).*

ROOT OF EQUATION—REGULA FALSI METHOD
To find the root of the equation $F(x)=0$ by the method of Falsi position (Regula Falsi method).
*Storage 334 (decimal).*

ROOTS OF EQUATION—MULLER'S METHOD
To find N (complex) roots of the equation $F(z)=0$ using Muller's method. The function may be transcendental and complex.
*Storage 490 (decimal).*

MATRIX INVERSION, SIMULTANEOUS EQUATIONS, DETERMINANT
To solve simultaneous linear equations and to find the inverse (if required) and the determinant of a real matrix.
*Storage 652 (decimal).*

SOLUTION OF SIMULTANEOUS EQUATIONS—GAUSS-SEIDEL METHOD
This standard Fortran IV subroutine solves a set of simultaneous linear equations using the Gauss-Seidel Iterative procedure.
*Storage 476 (decimal).*

GENERATION OF CHEBYSHEV POLYNOMIALS
To generate a Chebyshev polynomial.
*Storage 204 (decimal).*

SIMPLE LINEAR SORT
To sort numerically a one-dimensional floating point array.
*Storage 110 (decimal).*

DUAL LINEAR SORT
To sort numerically a one-dimensional floating point array, and duplicate the re-ordering in a second array.
*Storage 150 (decimal).*

POLAR-RECTANGULAR CO-ORDINATE CONVERSION
To convert Polar co-ordinates to rectangular co-ordinates, or rectangular co-ordinates to Polar co-ordinates.
*Storage 60 (decimal).*

**Input/Output Library**
Input/output routines are contained in the library for handling all available input/output peripheral devices (Fig. 5). Included are the conversion routines for ASCII to fixed point, floating point and complex; fixed point to floating point, floating point to fixed point and complex to floating point. The I/O library is made up of a set of subroutines for each I/O device. Each routine permits the user to specify the data format most convenient for his application. The I/O routine handles all necessary code conversion together with error checking and, where possible, recovery procedures are included.

## Fig. 5 Input/Output Library

| | ASCII | Binary |
|---|---|---|
| ASR-33 | ● | ● |
| ASR-35 | ● | ● |
| Paper Tape Reader | ● | ● |
| Paper Tape Punch | ● | ● |
| Card Reader | ● | ● |
| Card Punch | ● | ● |
| Line Printer | ● | |
| Magnetic Tape | ●* | ● |
| Disc File | ● | ● |

*Includes conversion to and from IBM 729 series tape code

**Loader Routines**
The function of these routines is to load the memory with object programs from input devices in either absolute or relocatable format. These routines are capable of loading the main program and subroutines called by it or called by other subroutines, and completes the transfer linkage between the main program and external subroutines. This is achieved by generating indirect address links in sector zero based on addressing information generated by the DAP-16 assembly program or Fortran IV compiler. It is the loader routines that desectorise the memory and which enable the user to address all the memory as if it was directly addressable.

### Plotter Software

Included in the I/O library are subroutines to drive a digital plotter. The subroutines are designed for use in Fortran IV programs and may be implemented using the call statements described below. With this set of programs it is possible to plot in graphical form any data which may be represented by a set of X, Y co-ordinates. Facilities are provided for displaying and automatically scaling the co-ordinate axes and writing the necessary numeric scale factors and alphanumeric captions on the graph. These subroutines are written partly in Fortran IV and partly in DAP16 assembly language. The subroutine names are PLOT, OFFSET FACTOR, WHERE, SYMBOL, NUMBER, ACCESS, SCALE and LINE. The PLOT routine changes the pen position and will draw a line when the pen is down. The OFFSET routine changes the position of the origin and provides new scale factors for each axes. WHERE provides the user with the current pen position and the current scale factor. SYMBOL plots all the standard alphanumeric characters plus 25 special characters starting at a specified point, at any angle or height.

### Utility Programs

### Debug Program (DEBUG)

The purpose of this program is to aid the check-out and debugging of programs by permitting the user to print out selected areas of core, delete or insert single words or blocks of words, enter break-points, initiate jumps or jumps and halts, and search all areas or part of memory for references to specify addresses. The storage required for this program is 628 locations and will run on either a 316, 416 or 516 system.

### Symbolic Source Update Program (SSUP)

SSUP reads symbolic records that are less than 80 characters and permits the user to delete, correct or insert records to the symbolic source creating a new revised symbolic source. The user specifies the SSUP operation by commands which specify the action required. SSUP requires 1055 locations and will run on any 316 or 516 having either a high-speed paper tape reader and punch options or two magnetic tape units or disc file.

### Dump

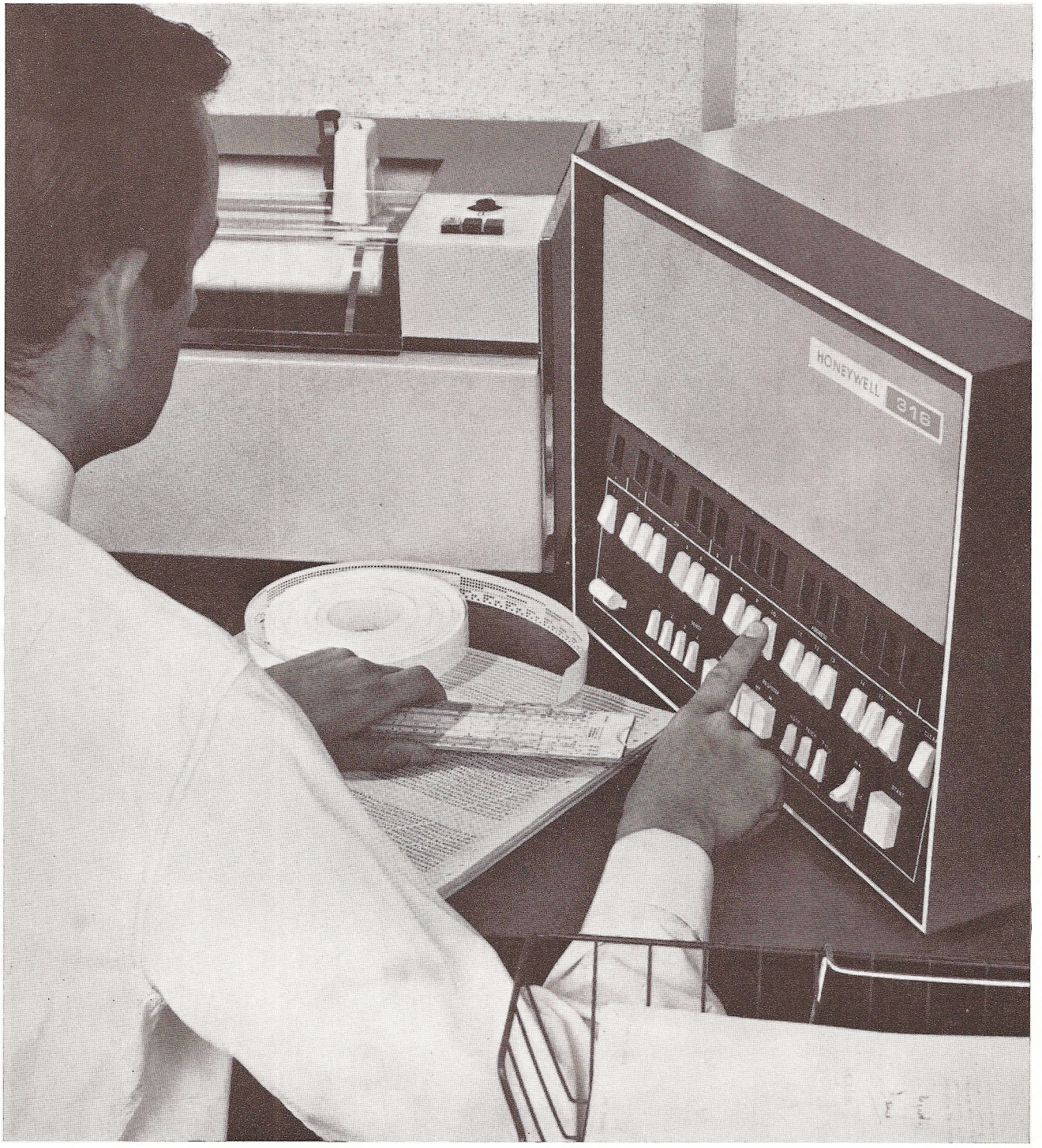
The purpose of this program is to print out, via the teletype, selected areas of core in octal or mnemonic instruction format. Dump requires one sector of memory (512 words) in which to operate in Series 16 computers.

### Verification and Test Programs

A full package of verification and test programs is provided with the computer system including routines for verifying the operation of the central processor unit, core memory and all available input/output devices. These routines generate indicative information reflecting the operational status of the equipment being verified. The programs are supplied to suit individual installations.

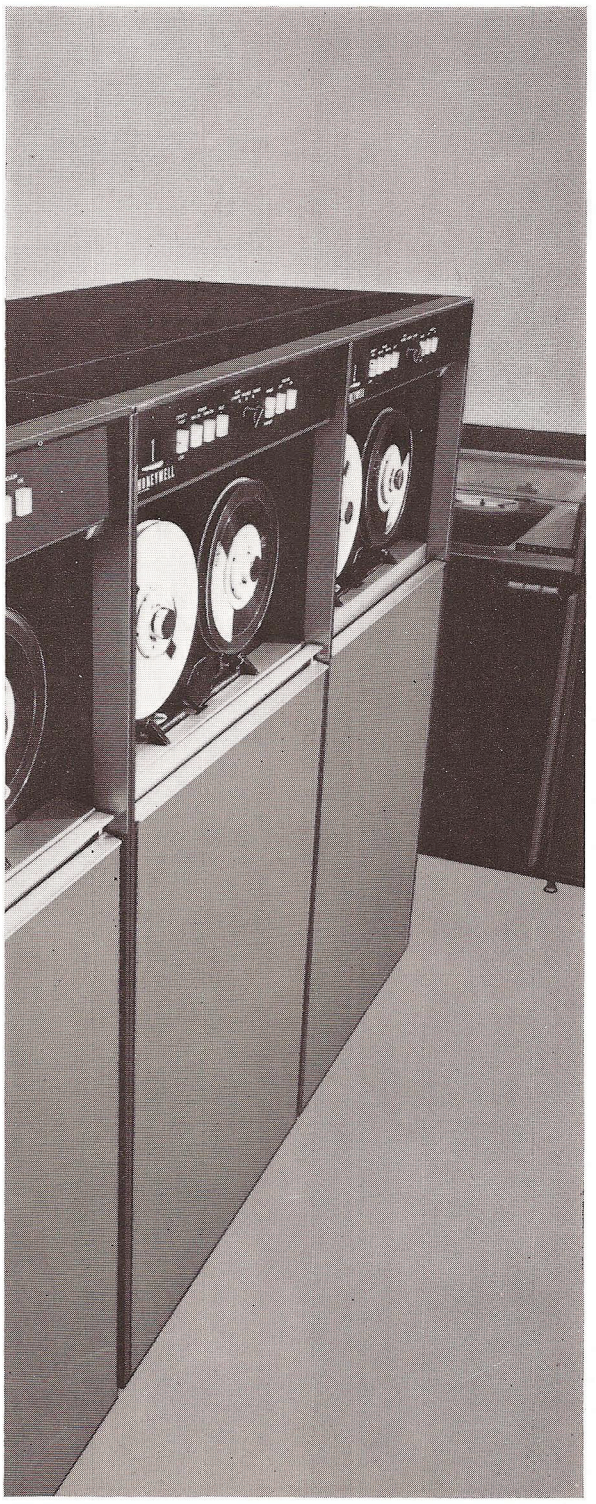
### Summary

Computers: 116, 316, 416 and 516 (except where stated).
Core Requirements (min.): 4096 words.
Availability: issued.
Free Issue: as standard software except plotter software.
Plotter software free issue with plotter purchase.


HONEYWELL 316

# Operating Systems
## Batch Operating System BOS

To meet the requirements of a wide range of applications and computer complexes the Honeywell 316 and 516 central processors have available operating systems which cater for on-line real-time systems and general-purpose scientific batch processing. These Series 16 systems are described fully under their individual headings.



**Batch Operating System (BOS)**

BOS is an operating system facilitating the processing of jobs sequentially in batches. Operating in this manner greatly increases the utilisation of Honeywell Series 16 computer systems by efficiently handling the execution and transfer on completion of the previous job to execution of the next job in the batch.

Jobs for processing are described to the operating system by the user through control commands. These commands direct the execution of programs and provide a link between the program and its environment. The system requirements for BOS are a 316 or 516 central processor unit with a minimum of 8192 words of memory together with backing store in the form of either a moving head disc file or fixed head disc file. The minimum recommended storage on disc files is 98K words. BOS offers complete and easy-to-use functions relieving the user of many tedious handling tasks. Jobs can be run by specifying a job description through control commands input through a command input device. This may either be an ASR keyboard, paper tape reader or card reader. The user specifies the input/output devices which are required to process the current job, e.g. for Fortran IV compilation the source input, object output and listing output need be specified. This is done by the ATTACH control command which causes a particular physical input/output device to be attached to an input/output stream. An example of this is to command ATTACH SI, PR connects the source input stream (SI) to the paper tape reader (PR). All subsequent calls for source input will cause the paper tape reader to be used. Devices remain attached until another ATTACH command is issued. Stream names are assigned two letter codes (CI for the command input, SO for source output, etc.). Similarly input/output devices are given two letter codes (PR for paper tape reader, AP for ASR printer, CR for card reader, etc.). To create a new file on the backing store the user

must give an ESTABLISH command specifying a file name and specifying whether the information contained in this file is in source, object or binary format. Source information is terminated by an end-of-file record or the next control command, and object information must be terminated by end-of-file record.

Storage of files on the backing store devices are allocated automatically by the operating system; BOS allocates storage in a manner which achieves the best utilisation of memory on backing store by reading/writing in blocks of 95 words. Blocks are chained together to form files, the disc unit is divided up into two areas at system generation time, one being system files, the other user files. File directory specifies the storage allocation for particular files. It is also possible at system generation to specify an area of disc file not accessible to BOS. This area is utilised for special user programs. The system files include all standard software, DAP-16 assembler, the Fortran IV compiler, mathematical and input/output library routines and utility routines. The users may create additional system files simply by naming these files with a label ($ character). The user may make a new file by combining one or more files by a MAKE command, in this way a specialised program library is created for his particular application.

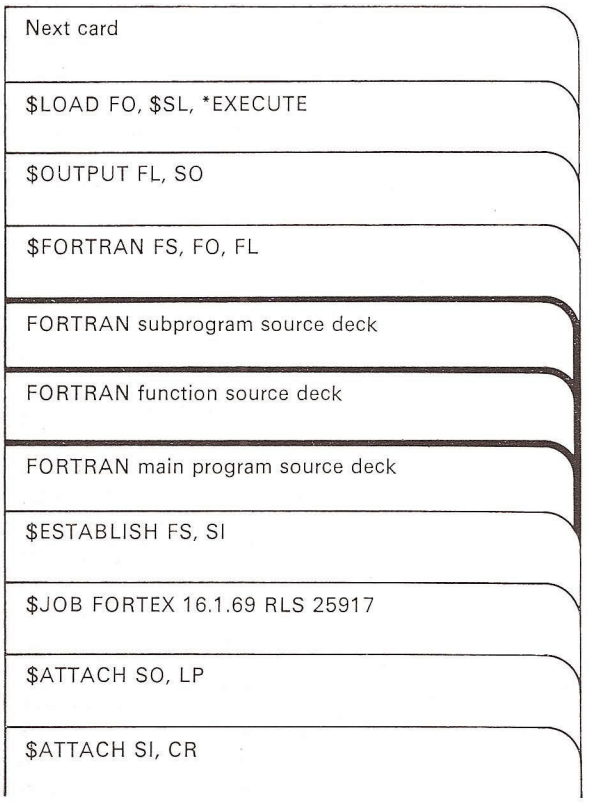
For a DAP-16 assembly job, the control command DAP followed by parameters that specify the name of the source file to be assembled, and the file name for object output and listing, causes the DAP-16 assembly to be loaded from backing store into memory and executed. On completion of this process DAP-16 will have produced an object file and listing file. The object file may be subsequently loaded by a LOAD command. An errors only listing may be obtained by output of the listing file to the error output stream or a full listing by output to the source output stream. In a similar manner a Fortran IV compilation is specified by a FORTRAN command with similar parameters specifying the source, object and listing file names. Execution of this command produces a one-pass Fortran compilation, creating an object program and a listing in the files specified. Options may be specified to produce an expanded listing including octal information or to insert trace code where possible. If the source file contains a Fortran IV main program this must precede any sub-programs or function sub-programs. The object file produced by the compiler may be loaded by a LOAD command. An object file containing the main program must be loaded first. A listing may be produced by output of the listing file to the source output stream. To load an object program resident in the disc file the user executes a LOAD command causing control to be transferred to the system relocating loader. This causes the loading of job files named in the parameter list following LOAD command to be transferred from backing store into core memory. Optional parameters are available specifying the base address in sector zero, the initial address of the program and the common base address. Other parameters specify a core map to be generated in a named disc file and to execute a program on completion of loading, starting at START address obtained from the memory map. An additional parameter to the load command is DEBUG. Appearing as the last parameter, causing the DEBUG package to be loaded into core and control to be transferred to the ASR keyboard where DEBUG command operations may be typed. The BOS command UPDATE calls in the updating sub-system to update the existing file (or old master) to produce a new file (called the new master) by updating instructions in the update master. The commands in the updating master are written as for the Symbolic Source Update Program (SSUP), enabling the users to create a source program on disc file without having to input or output programs on paper tape or cards.

### Example of a Complete Job

The following example shown in Fig. 6 illustrates how source programs and control commands may be combined to perform various complete jobs. The example illustrated here is in the form of cards but could equally well be read through punched paper tape. In the example source input is assigned to card reader and source output to the line printer. The job name is FORTEX, the information on the job card being printed out on the line printer. The Fortran IV main program, function and sub-program source card decks are all stored in a file FS, and then an object compiled to give a object file FO, and a listing file FL. The program listing is output on the line printer, the object file FO is loaded followed by the system

# Executive 16
# EXEC.16

## Fig. 6 Fortran Compilation and Execution




library. The program is then started at the first executable instruction in the main program. When the program is finished the Fortran IV statement CALL EXIT will be obeyed and the operating system will continue as directed by the next control command.

**Summary**
Computers: 316, 516.
Core Requirement (min.): 8192 words.
Backing Store (min.): 98K words.
Availability: released.
Free Issue: with appropriate purchased configuration.

EXEC-16, a Honeywell dedicated core executive operating system, has been developed to provide the user with a minimum-size real-time executive multi-programming capability. EXEC-16 runs on the H316 or 516 central processor and user programs are written in DAP-16 assembly language, using the Standard DAP-16 Assembler.

**Features**

**Multi-programming**
Every program running under EXEC-16 is described within the executive in the program library and can be requested from the console or by another program. The priority of the program is defined by its position on the list allowing several programs to run concurrently and their associated input/output operations to proceed in parallel.

**Scheduling**
The Honeywell EXEC-16 determines which program is to be executed next, system programs are checked in order of priority to determine which are currently due and are executed in order. Multi-programming capability is provided by the executive as a result of hardware interrupts. The executive searches the list of programs in the program executive tables in priority order, as determined by its position in the table. A maximum of fifty programs are allowed in the basic system.
Programs can become due either on demand from another program, a request through the keyboard, or from a set time interval elapsing in one of the three basic timing intervals. Regardless of how the program becomes due, all programs are started under executive control from the executive table list priority. Therefore, the user may plan his executive table list with relative importance to the overall system operation.

**Device Handling**
The input/output devices are associated with a priority interrupt and any and all such devices are accessible by any program within the system. When a program requests one of these devices, assuming the device is not busy, EXEC-16 gives the exclusive use of the device to the program requesting and continues the program. If the device is busy, the request is queued, first-in-first-out, and the

program requesting the device is suspended. Assuming the device becomes available, the program previously suspended is now rescheduled for execution with the input/output device available to the program.

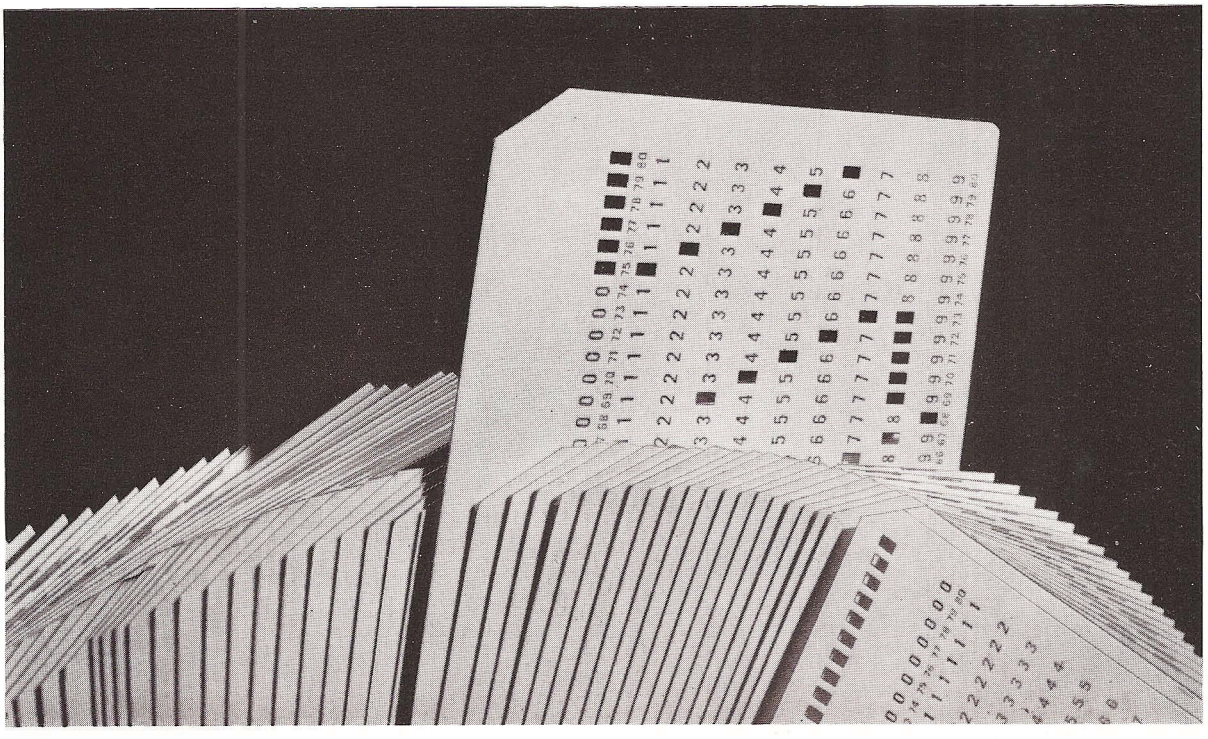
**Interrupt Handling**
One of the major hardware features of a real-time computer system is its priority interrupt capability. When a priority interrupt is received the contents of key system locations and registers are saved in order to return control easily to the interrupted program. The 'interrupt mode' program associated with the priority interrupt is then executed. When the 'interrupt mode' program is complete, control is returned to EXEC-16 in order to determine whether to continue the interrupted program or execute a program of higher priority which may now be due. Execution of the user's priority interrupt code commences approximately 100 cycles after the interrupt has occurred. (100 cycles represents worst-case response time, provided interrupts were not disabled when interrupt actually occurred.)

**Program Organisation**
In writing real-time programs, the program structure becomes a very important consideration.

*A program can be divided into two types of coding:*

**Non-Interrupt Coding**
This coding is executed in non-interrupt mode and is always initiated from the scheduling action of EXEC-16. (Coding can include several independent blocks of coding each identified by a label and ending in a statement returning control to the executive.) Each label defines a re-entry point in the program and non-interrupt blocks need not be in any specific order other than the initial entry block (first block).

**Interrupt Coding**
On requesting use of a device, assuming availability, the requesting program provides EXEC-16 with the label indicating start of interrupt coding to be executed on receipt of that device interrupt. The executive is then responsible for linking that device interrupt to the block of coding and subsequent execution of coding in interrupt mode when the



interrupt is generated. Coding is completed by a statement returning control to the executive, no strict ordering is required for interrupt blocks, and a program may have as many interrupt blocks as devices it requires.

**Time Management**
The real-time clock, a hardware interrupt, in the system is set at 100 millisecond intervals. From this interval three basic timing intervals are established: 100 milliseconds, 1 second, and 1 minute. The basic timing intervals are updated and checked to determine if programs linked to these intervals are due for execution. If any program is due, the status is recorded in the executive table list, and the time interval for that program reset to its original value.

**Keyboard Service**
The following functions with their descriptions and formats are available to the user through the ASR teletype:

*Enter Time:* ET, HOURS, MINUTES.
This function sets the real-time clock in hours and minutes.

*Display Time:* DT.
This function prints the current system's time in hours and minutes on the ASR Teletype.

*Request Program:* RN, NN=Name of Program.
This function allows the user to selectively start any user program contained in his executive table list.

*Enter Core:* EC, CORE LOCATION, CONTENTS OF CORE LOCATION.
This function allows the user to change any core location.

*Display Core:* DC, STARTING LOCATION, ENDING LOCATION.
This function allows the user to display on the ASR Teletype any core location.

**Hardware configuration**

**Minimum requirements**
H316 or 516 Computer with 4K of core memory, Real-Time Clock.
One input/output Typewriter.
(Teletype Model ASR-33 or ASR-35.)

*Optional items:*
One high-speed paper tape reader.
One high-speed paper tape punch.
Additional core memory capacity to a total of 16K.
Real-Time Interface
One Alarm Typewriter.
One Logging Typewriter, Model 'B'.
One Operator's Console.
One Moving Head Disc, operating with the DMC Option.

The absence of a standard option from the above listed options does not necessarily preclude it from inclusion with EXEC-16.

**Summary**
Computers: 316, 516.
Core Requirement (min.): 4096 words.
Availability: released.
Issue: subject to configuration charge.

# Interpretive Executive INTEX

INTEX is a small real-time executive for H316, 416 and 516 computers with 4K memory systems. This Honeywell program is capable of scheduling any number of system programs and will execute any number of system programs in parallel. It requires that the system programs are written in a systematic and definite method.

**Features**

**Interpretive Function**
The system programs are written in a definite format. Each format block is interpreted by INTEX and executed. The executive will not proceed to the next format block of the program until the previous one has been completed.

**Parallel**
During the execution of the format block of a particular program, spare processor time may be available. In such a case INTEX will search for other programs to execute, thus several system programs are capable of operating in parallel. There is no limit to the number of programs operating in this manner.

**Software Program Timers**
Each system program contains its own software timing location. Thus each program is capable of being suspended for a desired length of time.

**Input/Output**
All input/output drivers are subroutines of the executive. Each system program contains the demanding formats, but all scheduling and priority allocation remains the task of INTEX.

**Executive Organisation**
Honeywell's INTEX has been written in a completely modular concept. Routines may be added or omitted with ease in order to expand or contract its capabilities. The executive always contains three areas:

Input/Output Library.
Basic Routine Library.
Basic INTEX routine.

**Re-entrant Capabilities**
System programs, because of their special format, are completely re-entrant and also recursive, thus enabling a tremendous amount of saving in core space when several operations are identical except for parameter differences.

**Basic Intex routine**
The basic INTEX carries out the following objectives:
Schedules each main system program.
Organises the program format block into relevant data areas.
Calls all relevant executive routines from the executive library.

**Scheduling**
When a program is discovered to be in an active mode the pointer is used to access the program blocks. This and all subsequent blocks are then executed until a natural or forced program break is reached. Once the break has been encountered the executive continues scanning the program table and the process is repeated. Due to both the speed of the computer and, in exceptional cases, forced breaks, the overall resolution of the executive will not exceed in the worst case 500ms or 880ms on H316. Hence the executive is capable of driving several programs virtually in parallel; however, due to the scanning procedure there does exist a priority structure whereby higher priority programs can be given preference.

**Stack Nest**
An INTEX routine inspects the program block and organises it into a data area for that particular program. This data area is termed a Stack Nest. The program block consists of a number of parameters stored in the stack nest, from the top downwards and from the bottom upwards. The CALL word transfers control from the main program to the basic executive library routine. The latter then accesses the stack for its parameters and acts upon them.
It is interesting to note that this example is only one step in what could be a subroutine chain; as long as the final link is a basic executive library routine there is no theoretical limit to the depth of the subroutines written in the format block method. The practical limit depends on the capability of the stack nest to store all the required subroutines. A storage space of 64 locations is allocated for normal usage, though the stack nest levels fluctuate during program execution. Thus the final size of the stack nest required for a given system is determined during the checkout time of the specific system so that the worst case can then be taken into consideration.

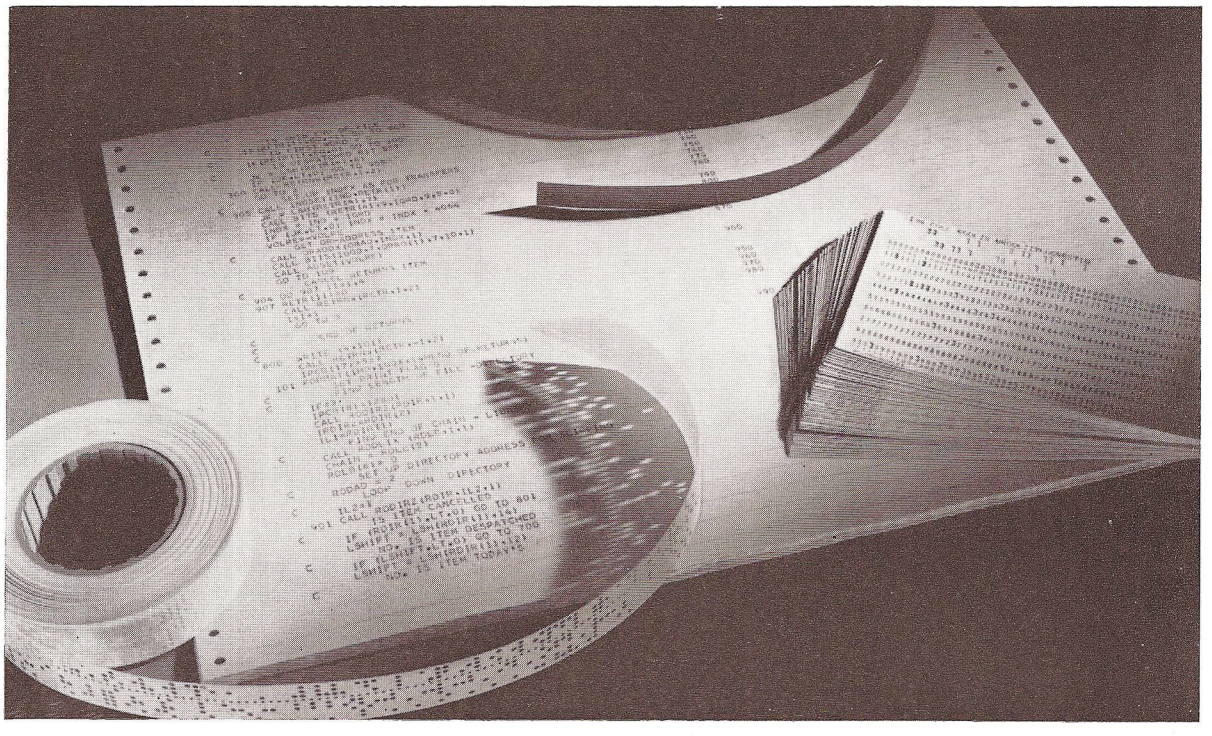
**Basic Executive Routine Library**
Because of the interpretative nature of the executive the final onus of INTEX is to execute the program demands. All such routines are contained in the Basic Executive Routine Library and can be expanded or contracted to meet the system requirements. The routines are written in machine code and access parameters from the stack nest.

*Routines available can be summarised as follows:*

| |
|---|
| Program Control Functions. |
| Boolean Algebra Packages. |
| Integer Arithmetic Packages. |
| Time Control Routines. |

**Input/Output Library**
In a real-time system, input/output techniques involve the use of interrupts. In order to control these interrupts, the philosophy of the executive demands the input/output drivers to form part of its library.
Input and Output formatting and demands are written as part of the user programs and interpreted by the executive, queued on the first-in, first-out basis. Most of these routines are written to customer specification; however, the Real-Time Clock and teletype handler are standard and already available.

**Summary**
Computers: 316, 416, 516.
Core Requirements (min.): 4096 words.
Availability: January 1970.
Issue: subject to configuration charge.



# On-Line Executive for Real-Time OLERT

On-Line Executive for Real-Time is a software package which forms the basis for Real-Time Multi-programming on the DDP-516 computer. OLERT provides scheduling, memory allocation input/output and other functions required for foreground/background multi-programming operation. Programs can be written for OLERT using either the DAP-16 assembly language or Real-Time FORTRAN IV. Source programs can be assembled or compiled in the off-line mode or, as a background function, under the control of OLERT using the on-line versions of DAP-16 and Real-Time FORTRAN IV. The object programs are in the OLERT formats linked into the system using either the on-line or off-line version of the OLERT system loader.

**Fortran Compatibility**
The user can write all of his programs in FORTRAN, including real-time input/output, interrupt response and time sequencing. OLERT programs carry out the real-time commands of Real-Time FORTRAN IV. This provides faster, simpler and less expensive solutions to programming problems.
Optionally, programs may be coded in the DAP-16 assembly language using standard calling sequences for linkages to the main executive.

**Memory Efficiency**
Re-entrant programs need appear in core only once for all users. They can be interrupted while being used by one program and then re-entered for use by other programs. Because of the method used to obtain this feature, there is virtually no limitation to the number of times such a program can be re-entered. This technique can be used to provide re-entrancy for a user-produced program which can be added to the permanent subroutine library.

**Multi-programming Capability**
OLERT can handle many concurrent real-time and background programs. The number is limited only by the memory capacity of the computer and peripheral devices.

**On-Line Program Development**
OLERT permits compilation, assembly, or loading and linking of programs and subroutines on-line. The operator can restructure the user software and can start and stop programs while other programs are being executed. On-line debugging is performed using the TRACE feature of FORTRAN IV.

**Interrupt Handling**
High priority interrupts are recognised and serviced during the servicing of lower priority interrupts. This ensures rapid response to emergency conditions.
Adaptability to changing process conditions is provided by the ability to change interrupt action during execution.

**Executive Control**
OLERT schedules the execution of programs by priority according to interrupt response, time, program requests, operator requests. OLERT provides disc and core memory management.

**Input/Output**
OLERT controls and executes all input and output operations. Editing is provided for all data formats (floating point, integer, ASCII, etc.) used with FORTRAN IV I/O statements. Devices are referred to symbolically, permitting checkout of special I/O on standard devices without reprogramming.

**System Protection**
OLERT uses the hardware protection features of the DDP-516 computer to prevent program errors from upsetting the entire computer system.

Memory protection prevents a program error from altering the memory assigned to a protected program. Errors are trapped and the operator is notified of the offending program and its location. Privileged instruction protection prevents the execution of input/output and other control instructions by those programs which are not allowed to use them.

**Modularity**
Only those portions of OLERT which are necessary for a particular system need be included. This reduces memory requirements for those applications which do not require all the features of OLERT.

**Structure of OLERT**
The basic OLERT package contains a scheduler, an interrupt handler, a clock-timer controller, an I/O request processor and a driver program for the Model ASR-33 or ASR-35 I/O teletypewriter.

**Scheduler** establishes the order of execution of user programs which have been scheduled or requested by other programs. Interrupt Handler controls the recognition and dynamic assignment of priority interrupts.

**Clock-timer Controller** module maintains the time-of-day and elapsed-time counters, and passes control to user programs on the basis of assigned times and time intervals.

**I/O Request Processor** controls the assignment of peripheral devices to user programs and controls the flow of data between the user's programs and the devices.

**ASR-33, ASR-35 and Other Device Drivers** are the interrupt-response programs which directly control the peripheral devices and pass the data between core memory and the devices. The driver programs for standard I/O devices are all interrupt-driven and designed to operate peripherals at or near rated speeds.
Other, optional, programs are re-entrant arithmetic package, re-entrant I/O editor, on-line program loader, record handler, file handler, disc driver. Some system functions are also optional.

**Re-entrant Arithmetic Package,** which includes single-precision floating and fixed-point, add, subtract, multiply and divide, is coded re-entrantly and can be interrupted and restarted for a higher priority user. The machine state is saved for restarting previous activities.

**Re-entrant I/O Editor** provides conversions for all data formats which are available with FORTRAN IV I/O statements. These include floating binary to decimal, integer binary to decimal, binary to alphanumeric (ASCII), double-precision binary to decimal, etc.

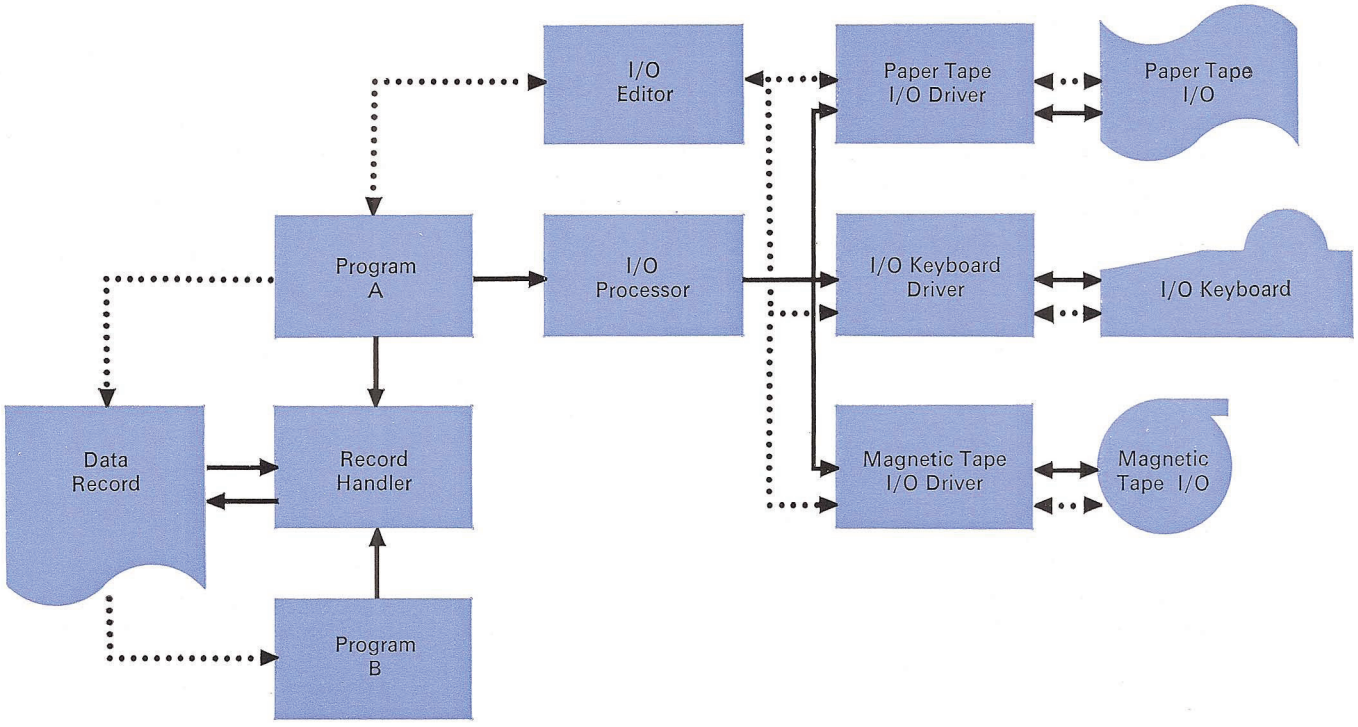
**On-Line Loader** accepts user programs in OLERT format and adds them to the system without disturbing concurrent computing activities.

**Record Handler** allows user programs, which are protected from each other, to pass data records using normal FORTRAN IV READ OR WRITE statements and formats. The OLERT record is a table of consecutive core locations used for passing data from one program to another. This provides a convenient means for program-to-program communications as well as memory-to-memory data conversion. The transmission of data through these records is shown in Fig. 7 by programs A, B and the record handler program.
File handler permits user program to transfer blocks of data between named files in bulk storage and arrays in core memory. Communication with the file handler program can be done using FORTRAN extension statements.

**OLERT Disc Driver** is the interrupt-driven software interface between the bulk storage device and the OLERT schedule and user programs.

**Systems Functions** allow the operator to communicate with the scheduler program through the ASR teletypewriter input/output keyboard in a conversational mode. The following operations are provided: call the on-line loader, assign memory, assign software priority level, print a memory map, print program library, initiate a program, abort a program tree, delete a program tree, delete a program from disc.

## Fig. 7 I/O and Records




Program A is passing data to several I/O devices, and also to program B using "records". The data is formatted, converted to characters and passed on to the appropriate drivers by the I/O editor.
The I/O processor ensures that the correct device is associated with program A at the proper time, that no other program can interfere with the flow of information to this device while program A has control, and that the data from program A reaches the correct driver for this device.

## Structure of User's Programs

**Interrupt/Noninterrupt Block Structure**—Programs can be separated into blocks. The rules of formation for blocks are given under the 'Real-Time Extensions' section on page 35. The first portion of a program, which is initiated as a result of a request from either the I/O typewriter or another program, is called the 'noninterrupt block'. In many cases, this is the entire program. Additional blocks of instructions, which are executed following an automatic interrupt or an elapsed time interval, are called 'interrupt blocks'. When an interrupt block is entered, all system registers and status indicators are saved. When the execution of interrupt code is complete, the scheduler restores all operating conditions and returns control to the highest priority program which is waiting to be executed. The interrupt blocks should have short execution times for most efficient operation.

**Program Levels**—The Scheduler establishes the order of execution of user programs requested by other programs or initiated by the operator. These programs are ordered by level on the basis of user-assigned priority (Fig. 8). Levels provide a software-based priority similar to that provided by hardware interrupts. As many as nine separate priority levels can be provided for each system. Within each level, programs are scheduled on a first-called first-run basis. A program is assigned a level when it is called for execution either by a central I/O typewriter command or by another program.

**Program Trees**—While level defines the relative priority of a program, the grouping of programs by tasks or function is defined by the task tree structure. The tree is used to determine the memory protection boundaries and command data requirements for programs involved in the same job. For a particular job, programs of different levels can be grouped into the same tree; however, any of the programs in the tree will be automatically assigned to a level which is the same or lower than that of the original program initiated from the I/O typewriter.

## Program Initiation and Control

**Interrupts**—A program's execution can be temporarily suspended by a hardware priority interrupt, the initiation of a program at a higher priority level, or by reaching some impasse such as waiting for input. As higher priority programs are completed and as input/output requests are serviced, the suspended program's executions are resumed.

The statement CONNECT is used to associate a block of user-generated code with a particular hardware interrupt. Whenever that interrupt occurs, the block of code will be executed according to the priority of the associated interrupt line.

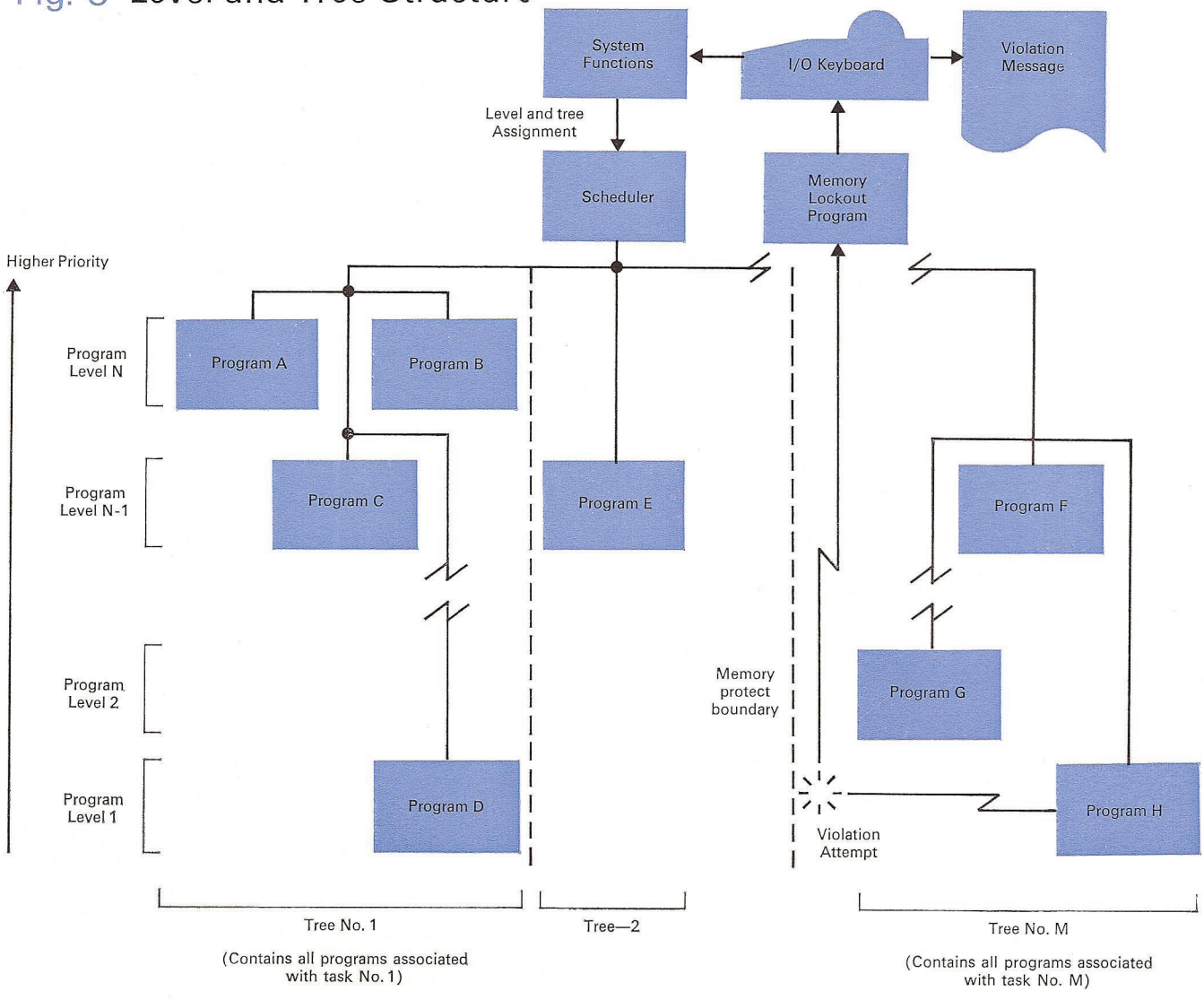
**Clocks and Timers**—The real-time clock is considered a special case of the hardware interrupts. The CONNECT statement is also used to cause initiation of blocks of user code either at a particular time or at specific time intervals.

The statement DISCONNECT removes the association between a block of interrupt code and the interrupt, clock or timer.

**Initiation of Noninterrupt Blocks**—A program in an OLERT system can consist of a main or noninterrupt block of instructions, usually computational, and several associated interrupt blocks which are usually I/O oriented. Computations in the noninterrupt blocks, as indicated by labels, can be scheduled by a schedule statement, typically within an interrupt block. The SCHEDULE statement is also useful for selecting alternate or multiple paths as the result of program decisions. It is meaningful only within a program, that is, program A cannot schedule a label in program B.

The REQUEST statement is used by a program to request execution of another program. This statement also provides the ability to pass parameters to the requested program. The calling program can assign a level to the requested program for the sake of establishing relative priorities. This is done by adding the assigned level number to the statement. Requests to execute a particular program can be made in one of two modes: sequential or nonsequential. When a sequential request is made, control passes from the requesting program to the operating system and is not returned until the requested action is completed. When a nonsequential request is made, control remains in the requesting program until the program is interrupted or voluntarily releases control.

## Fig. 8 Level and Tree Structure




Programs A, B, C and D (Tree No.1) are all involved in the same job. A, B and C are all high priority programs while D is low priority.
Tree No.2 involves a single-program task – Program E.
Program H has attempted to modify the memory of another tree. The attempt has been trapped and the tasks of Tree No.M will be aborted. The operator will be notified by a violation message.

**Input/Output**—Input and Output operations are requested through FORTRAN IV READ and WRITE statements or their equivalent DAP-16 calls. Both READ and WRITE statements can be sequential or nonsequential like the REQUEST statement. Input or output can also be requested on a different priority level than that of the calling program. A symbolic device name can be assigned to an actual device by the DEVICE statement.

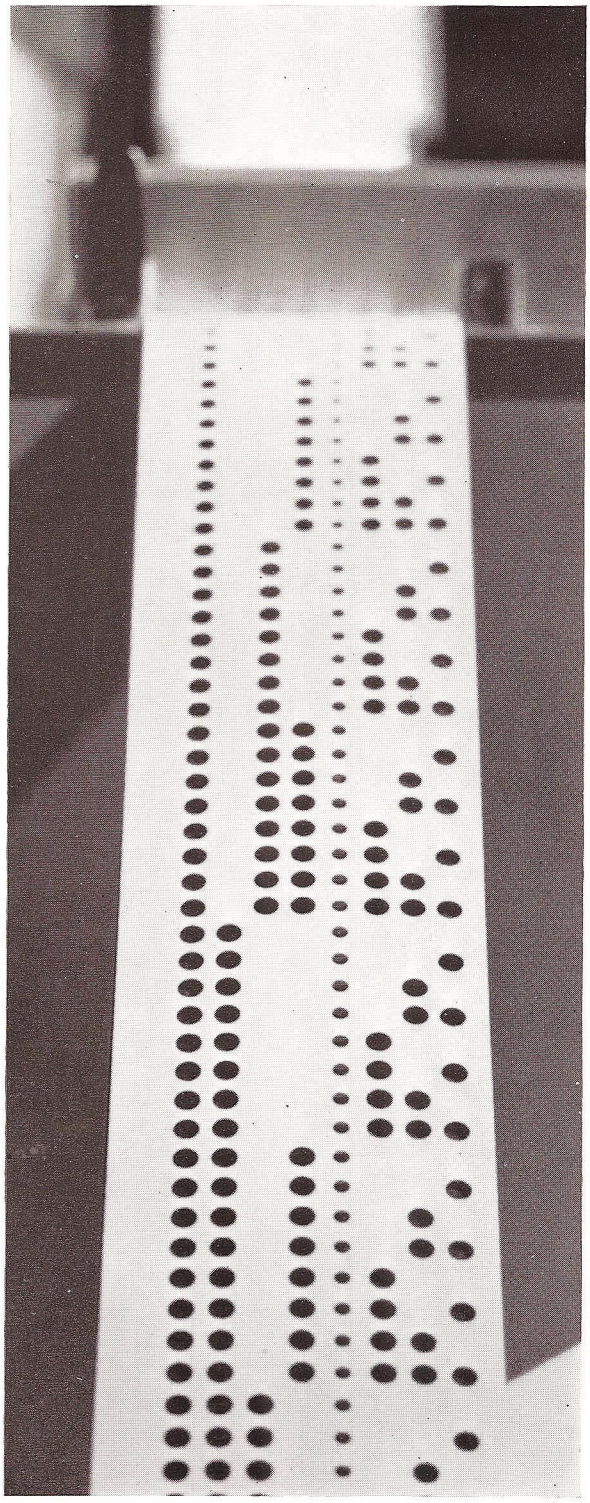
A program can obtain exclusive use of a device through the ATTACH statement. In this case, no other program is allowed to use the peripheral until the command DETACH is given. The current status and availability of a specific I/O device can be found by using the STATUS statement.

Record devices are core areas which can be treated as though they were devices. They are a convenient method of passing data between programs and trees, and performing internal formatting. The same READ, WRITE and FORMAT statements used with actual devices are used with these pseudo-peripherals.

Standard FORTRAN IV FORMAT statements can be used to describe the editing desired on the input or output data.

**Pending Event Test**—At any time during the execution of a program, the programmer can use the TEST PENDING statement to determine whether any action requested, such as input/output, is still to be completed. If such a status test is made and requested actions are incomplete the program will be suspended and control will return to the scheduler. When the actions have been completed, the program will be resumed. If all pending requests have been completed the program can optionally continue or terminate.

**Memory Management**—The disc provides storage for the System's program library as well as bulk storage for the programs. A map of disc and core utilisation is maintained by the memory allocator, which assigns areas on disc for program data files. Disc files can be named and are either private to a single program, or public to many programs. Information is transferred to or from these files by STORE or FETCH commands.



**Memory and System Protection**—The memory lockout feature of the DDP-516 computer is used to prevent programs involved in one job from interfacing with programs involved in another job. Each tree contains its own protection boundaries.
OLERT maintains control over the memory protection system by setting the proper hardware masks to establish protection boundaries, by activating the restricted execution mode, and by interpreting the attempted execution of a privileged instruction or memory protection violation. A tree will be involuntarily cancelled if a program in its structure attempts to execute a privileged instruction or to write in a protected area of memory. If this happens, the operation will be notified via the I/O typewriter.

**Relocation**—The relocation feature of OLERT allows a program to be loaded from bulk storage or paper tape into any available block of contiguous 512-word core sectors for execution. The program must start on a sector boundary.

**Operator Communications**—The operator communicates with OLERT through the I/O typewriter keyboard. He can request loading of a user's program via the on-line system loader. He can also delete programs or trees when necessary. The system loader allocates core at load time, eliminating much of the record keeping normally required of the programmer. The loader also constructs and maintains a program library containing the programs which have been entered in the system.

**Real-Time Extensions to Fortran IV**
The following four statements illustrate the function of the additional real-time extensions to FORTRAN IV. There are 34 additional statements. For each Fortran IV extension statement there is an equivalent DAP-16 calling sequence so that the same operations can be carried out when coding in assembly language.

**Request**
The REQUEST statement causes the execution of another program to be scheduled. If the requested program is on disc, a copy is transferred to unoccupied core sections for execution. Only one copy of a program need be on disc, but a copy will be made in core for each tree.

REQUEST *PROGRM* (*AI . . . AN*) PROGRM is the name of the requested program and *AI . . . AN* are arguments which are treated in the same way as FORTRAN subroutine arguments. Following the REQUEST statement, when execution of the program *PROGRM* is finished, control returns to the next statement in the calling program. It is a sequential statement.

**Connect**

CONNECT INTERRUPT *ACTION* (3).
This associates an interrupt's with a program's interrupt block. *ACTION* will be initiated whenever interrupt 3 occurs. Special cases of the interrupt blocks are the clock blocks. These are initiated as the result of elapsed or absolute time.

**Schedule**

SCHEDULE *200*
Causes the portion of the program starting at the statement with the label *200* to be scheduled for execution by the operating system.

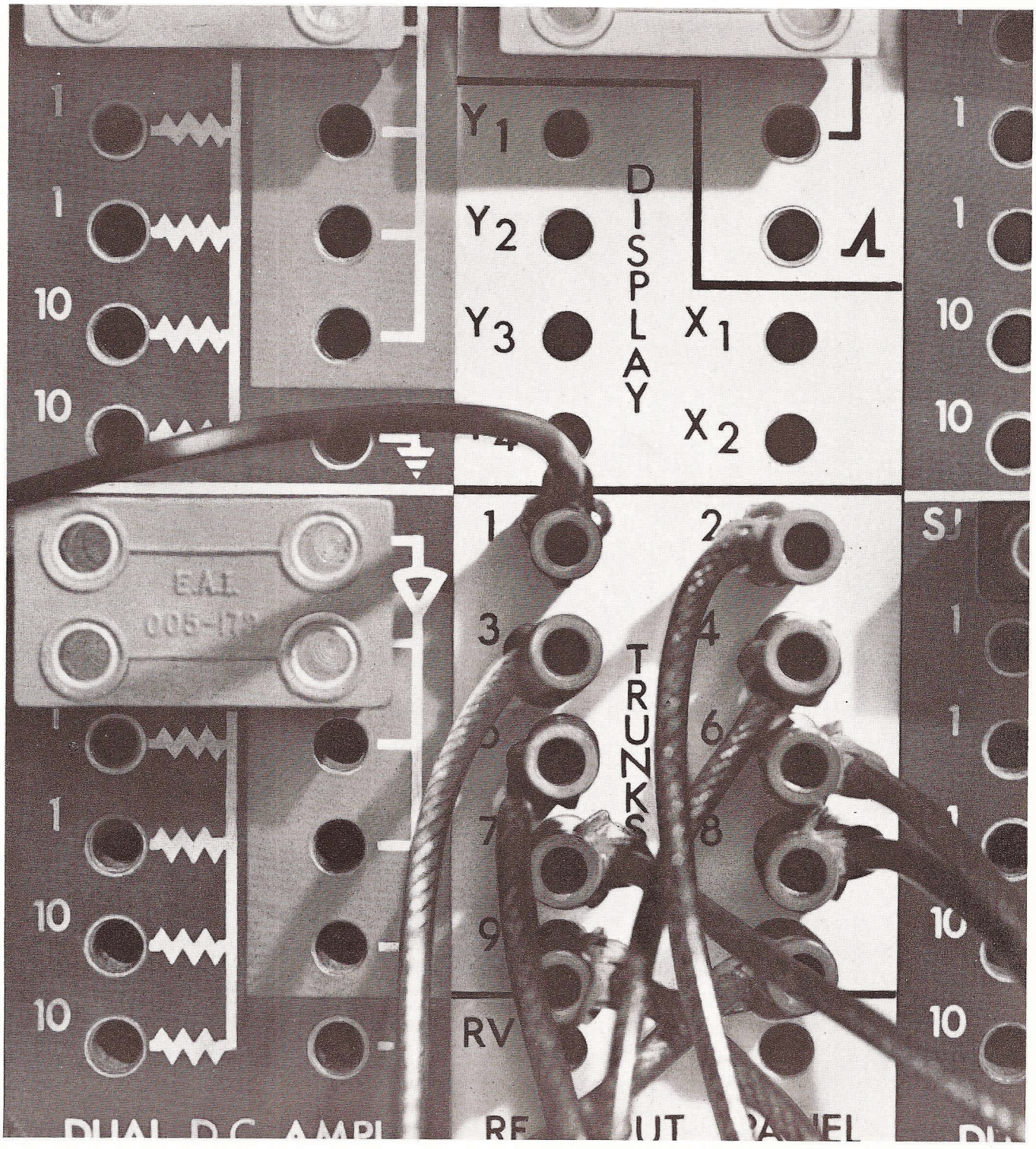
**Attach/Detach**

ATTACH (*OPERIO*)
Assigns exclusive use of the device *OPERIO* to the calling program. No other program can use that peripheral until the command DETACH (*OPERIO*) is given.

**Interrupt Response**
The elapsed time for the sensing of an interrupt to the execution of the first instruction of an interrupt associated program is nominally 100 microseconds and can reach a worst-case value of 300 microseconds.

**Hardware required and supported**
Minimum hardware requirements for OLERT are a 24,576 word DDP-516 computer with the memory lockout option, real-time clock, ASR-33 teletypewriter and fixed head disc. The basic OLERT system can be expanded to accommodate memory sizes up to 32K, priority interrupt option, high-speed paper tape reader and punch.

**Summary**
Computer: 516.
Core Requirements (min.): 24,576 words.
Backing Store (min.): 98K words disc file.
Availability: issued.
Issue: subject to configuration charge.


Y1
Y2
Y3
DISPLAY
X1
X2
TRUNKS
RV
1
2
3
4
6
7
8
9
10

# Digital / Analogue Simulation MIDAS

**Midas**

The most common scientific and engineering problems involve a study of the behaviour under specified conditions of dynamic systems. It is desirable to simulate the behaviour of a proposed system by means of a mathematical model. The behaviour of a dynamic system is obtained by solving the equations which describe that system and analogue or hybrid computation systems offer a convenient method of simulating the system and thus implementing the solution of the system equations.

Dynamic systems are in general only described adequately by a set of non-linear differential equations of such complexity that analytical techniques are not possible and probably not desirable; thus numerical methods developed for a digital computer offer the only alternative.

*System simulation languages are used currently to fulfil two main purposes:*

The independent solution of system equations in order to verify or increase the accuracies of results obtained from an analogue or a hybrid computer.

The solution of system equations on a digital computer whilst still retaining the programming and operational conveniences of the analogue and hybrid computational techniques.

The Series 16 MIDAS program provides a large number of operational elements such as would be found on an analogue/hybrid computer. These elements include integrators, summers, multiplier, relays, inverters and many others which have specific input/output relationships.

Programming by means of MIDAS consists basically of 'interconnecting' these elements so as to set up the equations describing the physical system under consideration. This is entirely analogous to the use of a patchboard system on an analogue computer for the interconnections of the electronic operational elements. Just as it has been proven essential to the analogue programmer to prepare a schematic or flow diagram to indicate the elements and their associated interconnections, a very similar form of block diagram is useful as a first step in preparing a MIDAS program. Next a listing is prepared from the block diagram specifying for each element the source of its inputs. This listing is prepared according to a few simply defined rules in input format. With the addition of several other items of information such as a calling sequence, numerical data, etc., the programming is complete. Thus the user requires no computer experience to be able to utilise MIDAS. Fig. 9 gives a summary of the MIDAS elements available.

**Summary**

Computers: 316, 516.
Core Requirements (min.): 16,384 words.
Availability: released.
Issue: subject to configuration charge.

# Fig. 9 Summary of Midas Elements

| Item | Name | Inputs | Outputs |
|---|---|---|---|
| **1. Mathematical Operations** | | | |
| Integrate | Ij | A | $\int_0^t A\,dt + IC$ |
| Sum | Sj | $A_i (i=1,2,..K \leq 6)$ | $\sum_{i=1}^{K} A_i$ |
| Negative | NEGj | A | −A |
| Multiply | Mj or Pj | A, B | A.B |
| Divide | Dj | A, B | A/B |
| Absolute Value | ABSj | A | (A) |
| Square Root | SQRj | A | $\sqrt{A}$ for $A \geq 0$ |
| Exponential | Ej | A | $e^A$ |
| Natural Logarithm | LNj | A | ln A for A> 0 |
| Resolver | RESj | A | B output = sin A (Note: A must be in radians)<br>C output = cos A |
| Arc Tangent | ATj | A | $\tan^{-1} A$ where $-\frac{\pi}{2} < \tan^{-1} A < \frac{\pi}{2}$ |
| **2. Switching Elements** | | | |
| Output Relay | ØRj | A, B | C output: A if $B \geq 0$, 0 if $B < 0$<br>D output: 0 if $B \geq 0$, A if $B < 0$ |
| Input Relay | IRj | A, B, C | Output: A if $C \geq 0$, B if $C < 0$ |
| Function Switch | FSWj | A, B, C, D | Output: A if $D > 0$, B if $D = 0$, C if $D < 0$ |
| Limiter | LIMj | A, B, C, | Output: B if $A > B$, A if $C \leq A \leq B$, C if $A < C$ |
| Bang-Bang | BBj | A, B | Output: B if $A \geq 0$, −B if $A < 0$ |
| Dead Space | DSj | A, B, C | Output: A−B if $A > B$, 0 if $C \leq A \leq B$, A−C if $A < C$ |
| **3. Arbitrary Functions** | | | |
| Function | Fj or DFGj | A, Data Lines | f(A) Linear Interpolation; Data for every run. |
| Constant Function | Gj | A, Data Lines | f(A) Linear Interpolation; Data for first run only. |
| Curve Follower | CFj | A, Data Lines | f(A) Quadratic Interpolation; Data for every run. |
| Constant Curve Follower | CGj | A, Data Lines | f(A) Quadratic Interpolation; Data for first run only. |
| **4. Iterative Element** | | | |
| Implicit Function | IMPj | A, B | |
| **5. Run Termination** | | | |
| Finish | FIN | A, B | Stops computation when $A \geq B$ |
| **6. Insertion of Numerical Items** | | | |
| Constant | CØN | Data Lines | 1 to 6 items/line. Data for first run only. |
| Parameter | PAR or K | Data Lines | 1 to 6 items/line. Data for every run. |
| Initial Condition | IC | Data Lines | 1 to 6 items/line. Non-zero IC's for every run. |
| **7. Special Statements** | | | |
| Header | HDR | — | 1 to 6 items/line. Provides titles for the readout. |
| Readout | RØ | — | 1 to 6 items/line. Specified items to be printed. |
| End | END | — | Signifies the end of the symbolic program. |
| **8. Special Names** | | | |
| Independent Variable | IT | — | Current value of the independent variable. |
| Time between Readouts | TR | — | 0.1 units of the independent variable, usually time, unless otherwise given on a CØN or PAR line. |
| Minimum Interval of Integration | MINI | — | $10^{-7}$ unless otherwise given on a CØN or PAR line. |
| Integration Option | ØPTIØNj<br>ØPTIj | — | Specified a non-standard integration option. |

# Honeywell

## United Kingdom

Honeywell Ltd.,
Computer Control Division,
53 Clarendon Road,
Watford, Herts.
☎ Watford 42391
Telex: 934227

Honeywell Ltd.
Computer Control Division,
Honeywell House,
Station Road,
Cheadle Hulme,
Cheshire.
☎ 061-485 6116
Telex: 66509

Honeywell Ltd.
Computer Control Division,
Observatory House,
Windsor Road,
Slough, Bucks.
☎ Slough 33366
Telex: 84601

## Sweden

Honeywell AB,
Computer Control Division,
Storsätragrand 5,
127 86 Skärholmen,
☎ 08/88 00 00,
Telex: 854-10673

## Finland

OY Honeywell,
Hitsaajankatu 5,
Helsinki 81,
☎ 780311
Telex: 12–1229

## France

Honeywell S.A.,
Computer Control Division,
92 rue de Courcelles,
Paris 8eme.
☎ 267 4435
Telex: 29623

## Germany

Honeywell GmbH,
Computer Control Division,
6050 Offenbach/Main,
Kaiserleistrasse 55.
☎ 8.0641
Telex: 41–52758

Honeywell GmbH
Computer Control Division
4 Düsseldorf
Moersenbroicher Weg 200
☎ 6 21 61
Telex: 858 6754

## Netherlands

Honeywell NV,
Computer Control Division,
Rijswijkstraat 175,
Amsterdam.
☎ 020-159343
Telex: 13066

## Switzerland

Honeywell AG,
Computer Control Division,
8008 Zürich,
Dufourstrasse 47.
☎ (051) 47.44.00
Telex: 53.561

Honeywell AG
Computer Control Division
1200 Geneva/Switzerland
73, Route de Lyon
☎ 44 25 50
Telex: 22 670

## Italy

Honeywell S.p.A.,
Computer Control Division,
Via V. Pisani, 13
I 20124 Milano.
☎ 6245
Telex: 32092

## Other countries export

Honeywell Ltd.,
Computer Control Division,
53 Clarendon Road,
Watford, Herts.
☎ Watford 42391
Telex: 934227

THE QUEEN'S AWARD TO INDUSTRY 1969

Designed and printed in U.K. by Copia Productum, Harrow