LET ANI PRINT S
ANO READ RDI
RDO GOTO SDO
IF
CNO CLK RETUR
STOP CLK TIMER
CONTACT COUN
DISMISS LET PRIN
AND READ SDO

digital

INDUSTRIAL BASIC

...Solving the Industrial Control Problem

INDUSTRIAL BASIC is

Easy to Learn

Industry-Standard Language
English Command Expressions
Complete Documentation
One Week Training

Easy to Implement

Support of Process I/O
Extensive Program Development
Facilities on the Same System
Simple System Generation
Quick Start-Up with ROM Bootstrap
Power Fail Optionally Included

Easy to Expand

Supports Extended Digital and Analog I/O
Supports up to 32K Core Memory
Extended Arithmetic Capability
Disk Storage Capability

INDUSTRIAL BASIC...Solving the Industrial Control Problem

DIGITAL's continuing commitment to solving the problems of industry has evolved into its newest adaptation of the worldwide BASIC language...INDUSTRIAL BASIC. In the past, total minicomputer control of processes has been difficult (and expensive) due to the restrictions of assembly language programming and software development. INDUSTRIAL BASIC now solves the control problem.

Why INDUSTRIAL BASIC?

A major problem in applying a minicomputer to control of a production process is the high cost involved in system integration and software development...far exceeding any increases in productivity. Without control-oriented software, the number of options open to a control engineer in implementing a dedicated control system is limited: The first option is for the engineer to build and implement a computer-based control system without other assistance. The necessary hardware is available, and at a reasonable cost. However, many engineering hours must be spent customizing interfaces, integrating, and checking out the system. The engineer must also know assembly language or if not, hire a programmer. Here, the cost of the software is normally equal to or often higher than that of the hardware itself, and the cost of the system doubles. Moreover, the lack of flexibility automatically reduces the effectiveness of the control system.

The second option is to obtain a complete hardware and software "turnkey" control system from a computer manufacturer or custom system house. This package usually includes software, interfaces, and application programs. But the cost is usually three times that of the first option. This approach not only restricts the user from learning all phases of his process control system, but may force disclosure of proprietary information. Even further, the user is totally dependent on the computer manufacturer or "turnkey" system supplier, limiting future system development possibilities.

The Solution: INDUSTRIAL BASIC's Control-Oriented Software

There is a solution to industrial control; it's called INDUSTRIAL BASIC; and, it's supplied in DIGITAL's Industrial-800 Systems. Process or control engineers can now do their own programming and interact with processes in this new adaptation of the widely familiar BASIC language. INDUSTRIAL BASIC is geared specifically toward ease of implementation and programming. It is designed with the control engineer in mind, and for real-time control applications ranging from process control to quality control to testing to material handling to monitoring.

BASIC to INDUSTRIAL BASIC

High-level languages are easy to learn and in fact, many process and control engineers are already familiar with BASIC. Hence, the use of BASIC in the development of INDUSTRIAL BASIC. Inherently, BASIC provides mathematical and decision-making capability; but, it does not provide operations necessary in industrial control applications such as service of external interrupts, analog and digital input and output, and time-based scheduling. DIGITAL's new BASIC is INDUSTRIAL precisely because it offers features such as digital input, digital output, digital output readback, analog input, and analog output. In addition, it provides the statements and functions necessary to service external interrupts.

A word about BASIC: Originally developed by Dartmouth University, standard BASIC is an interactive, high-level programming language aimed at enhancing communication between user and computer. Vehicles for this communication are common English expressions (commands) which permit direct user influence of any aspect of the program or process. This means that programs can be created, modified, and debugged simply... reducing overall program development time. And because of the simplicity of BASIC, you need not be a programming expert to understand and use the language. The process or control engineer can easily and effectively work with BASIC, eliminating the need for an outside systems house for program development.

Some elementary BASIC commands are:

LET—assigns a variable a value, whether it is a constant or the result of arithmetic expressions.

PRINT—prints out the indicated information.

READ—transmits values from the data list to the variables.

DATA—provides data for a program.

GOTO—changes the order of program execution.

IF THEN—conditionally changes the order of program execution.

GOSUB—directs the program to a subroutine within the program.

RETURN—the command to return from a subroutine.

STOP—terminates program execution.

Others are FOR TO STEP, NEXT, INPUT, REM, RESTORE, DEF, END, DIM, UDEF.

By its original Dartmouth definition, BASIC is considered a sequential system: Each command produces a result when it is executed; in fact, the only commands that may given are those which have immediate results.
For instance:

```
READY [1]
90 INPUT A
100 LET B = SQR(A)
110 PRINT A, "SQRT(A) = ",B
120 GOTO 90
130 END
RUNNH
?[1]
```

1. Computer output.

The BASIC user types in the computational procedure as a series of numbered statements, as shown.

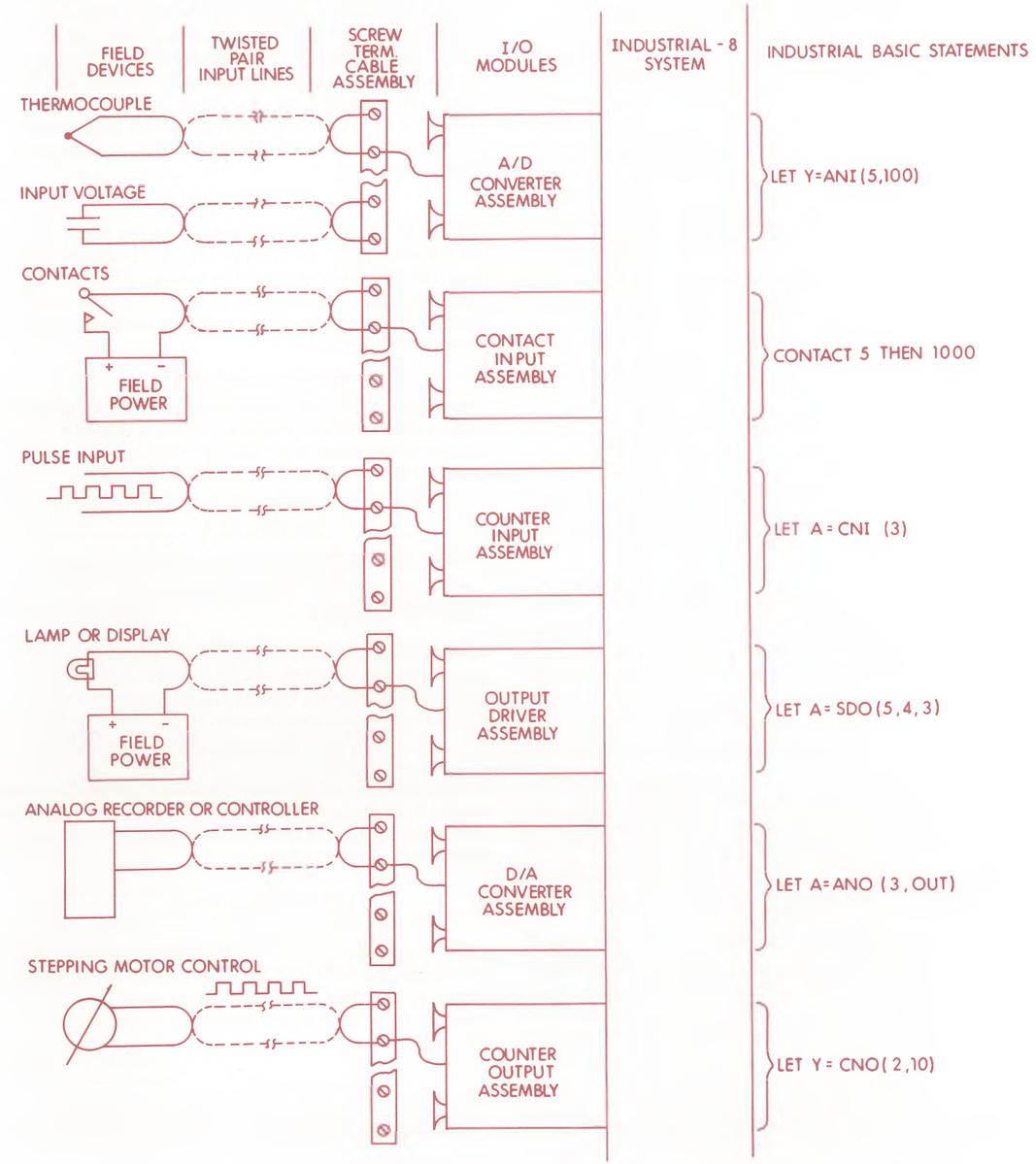
BASIC also includes certain functions which are a part of the language. These functions eliminate the need for writing many small subroutines to perform elementary calculations. Included are Sine, Cosine, Arctangent, Exponent, Natural Logarithm, Random Number, Absolute Value, Integer, Sign, Square Root, Trace Functions, and any User-defined Functions. They enable development and testing of the application program quickly and easily. Facilities for corrections and editing are also incorporated into BASIC, as well as looping, string-handling, listing, and tables.

The DIGITAL *Programming Languages* handbook is available for additional information on BASIC. Or any other BASIC text is applicable; information can be readily obtained because standard BASIC is taught in elementary through college-level educational institutions.

INDUSTRIAL BASIC Real-time Operation

Standard BASIC operates only in a step-by-step fashion where one operation logically leads to the next. INDUSTRIAL BASIC makes total real-time control possible with the addition of new capabilities for analog and digital inputs and outputs and external event-driven routines. With external event routines, an external signal to the program causes a specific routine to be performed, or a routine may be performed every so many seconds. INDUSTRIAL BASIC accepts inputs from a process and can generate the necessary controlling outputs within a time acceptable for successful process operation...complete real-time control.

Expanded Input/Output Capabilities for Sequential Operation



Industrial-8 System with interfaces and INDUSTRIAL BASIC statements

Any given command produces direct results with sequential operation. For example, a voltage reading may be critical to some portion of your production process. You may need information regarding that voltage at any time. So you issue a command, get the data, and make an appropriate decision.

Even further, you might require either analog or digital inputs and outputs in a real-time situation. Analog inputs and outputs are those that vary between two arbitrarily selected limits of say, voltage, current, or resistance. Control devices that generate analog values are pressure transducers, thermocouples, or tachometer generators. Digital values, on the other hand, are those that switch between two levels of voltage, current, or resistance. One of the levels is usually OFF and the other ON. Limit switches or pushbuttons are examples of devices that emit digital values. The new expansions of INDUSTRIAL BASIC allow you to generate this input/output information easily.

Digital Input:

RDI reads digital input devices such as mechanical or solid-state switches and thumbwheels (single- or multi-point input capability). Digital input can be a single "point" or a group of contacts in a BCD thumbwheel, or a measuring device readout. Input points are organized in ascending order from "point" 1 to the maximum limit allowed in the system. A single toggle switch connected to the system as point 20 may be sampled by the statement:

```
LET Y=RDI (20,1)
```

Y is set to "0" or "1" depending upon the status of the toggle switch ("0" for open or off, "1" for closed or on). The RDI function performs the sampling. To avoid sampling each individual bit or point of a thumbwheel or readout device, a more expanded version of the same statement is:

```
LET Y=RDI (20,4)
```

Y will be set to the value of the 4 points starting at point 20. If a BCD readout device with 4 digits of output were connected to digital input points 1 through 16 (4 points per digit times 4 digits), the statement would be:

```
Y=RDI (1,4) * 1000 + RDI (5,4) * 100+RDI (9,4) *
10+RDI (13,4)
```

Y would contain the value from the readout device.

Digital Output

SDO sends digital output. This function allows relays, solenoids, and stepping motors to be driven under program control. Simply, if a relay were connected to point 26, that relay could be energized by the simple statement:

```
LET Y=SDO(26,1,1)
```

In this operation, SDO is the send-digital-output function, and 26 the selected point. The number of points specified is 1 and the second 1 is the value to be sent. The relay could be de-energized by:

```
LET Y=SDO (26,1,0)
```

Here, 0 opens the relay. As with digital input, multiple points may be output to ease the display of BCD data. Consider the display via BCD display on points 1 through 12 of BCD thumbwheels on input points 1 through 12:

```
100 Y=SDO (1,4,RDI(1,4))
110 Y=SDO (5,4,RDI(5,4))
120 Y=SDO (9,4,RDI(9,4))
```

If the user desires, the above statements could be combined into one.

Digital Output Readback

RDO reads digital output. This function provides a convenient form of digital output readback for common algorithms that deal with output devices, rather than the sampling of input switches. The function actually returns information reflecting the commands that were given to the output devices. This is useful in systems where the output devices are set and reset by a number of routines:

```
LET Y=RDO (20,2)
```

Here, Y is set to a value reflecting the latest output commands given to points 20 and 21.

Analog Output

ANO generates a voltage or current output on the Digital to Analog Converter. This function provides the capability to drive instrumentation such as proportional valves, setpoint controllers, and measuring devices. For instance:

```
LET Y=ANO (15,OUT)
```

where analog output point 15 is selected and the analog value of OUT is converted to a voltage or current by the computer output module. The output may be in one of four ranges:

```
Voltages: 0 to +10
          1 to +5
Currents:  4 to 20MA
          10 to 50MA
```

In order to generate a value for a setpoint controller, and if OUT is the percentage of full scale, the statement would be:

```
LET Y=ANO (15,1.023 * OUT)
```

In this example, 1.023 is a scaling constant.

Analog Input

ANI samples data on the Analog to Digital Converter. Industrial processing requires both analog and digital inputs to be complete. Since analog "points" or channels can reflect a range of values, this treatment evolves:

```
LET Y=ANI (10,200)
```

Y is set to the value of channel 10 and read at a gain of 200. The value of Y will be the voltage sampled at channel 10 multiplied by 200. Polynomials can be constructed to convert the analog inputs to engineering units (degrees Fahrenheit, psi, etc.) so that the data may be processed and acted upon.

Counter Output

CNO loads a counter module with the number of items to count before generating an interrupt.

```
LET A=CNO (2,300)
```

The channel of the counter module is 2, and the number of items to count before generating the interrupt is 300.

Counter Input

CNI returns the number of counts remaining in a counter module.

```
LET A=CNI(3)
```

In this example, A is assigned the number of events remaining to be counted in counter 3.

Clock Function

CLK performs two functions: first, if the variable X in an operation is positive, the system clock is set to this value. (X must be less than 86,400—the number of seconds in 24 hrs.); second, if X is zero or negative, the value (in seconds) of the system clock is returned.

To set the system clock to 12:00:00:

```
100 LET T=12*3600
200 LET X=CLK(T)
300 END
```

To read the system clock

```
100 LET T=CLK (0)
```

Simply, four classes of input and output are supplied by INDUSTRIAL BASIC:

Analog (Input and Output)
Digital (Input and Output)
Counter (Load and Read)
Clock (Set and Read)

These functions permit operation of processes and devices, automatically under program control... the parameters being analog, digital, counting or timed conditions. While these are required features in most control/data acquisition systems, INDUSTRIAL BASIC uses them as a starting point and expands into the servicing of external interrupts.

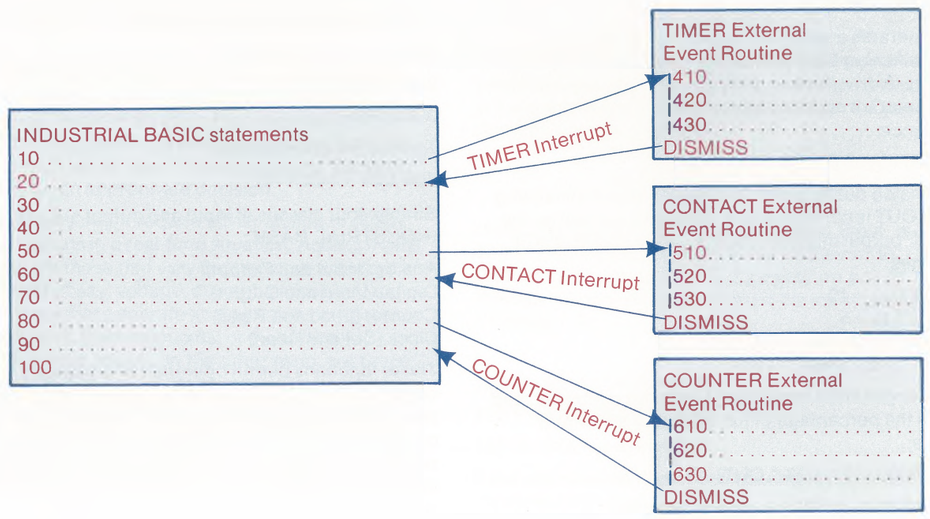
External-Event-Driven Control

Many industrial computer applications must make provision for operations, actions, or inputs not under direct control of the program. For instance, alarms may go off unexpectedly or other periodic activity may take place that has scheduling conflicts. Control of external devices and conditions is usually by sampling, which requires that all inputs must always be polled, though a critical situation may never arise.

Control of external situations requires a more powerful language capability to instruct the operating system in what specific action to take. The most significant additions to the BASIC language and operating system that create INDUSTRIAL BASIC are those that allow the system to recognize and deal with external interrupts.

In the INDUSTRIAL BASIC system, the control engineer is able to establish external event routines just as though they are subroutines in BASIC. In place of the normal GOSUB however, events are associated with or disconnected from an external event routine. The association statements define which external event routines are to be activated and when. These routines are priority operations and will interrupt the mainline program when called. The interrupting is controlled by the operating system and is completely transparent. (The mainline program is suspended and reactivated without assistance or knowledge of the programmer.)

When an external event routine is set up and an external event interrupt occurs, the operating system can recognize, schedule, and resolve conflicts in operations. Control is automatically transferred to the specified line in the INDUSTRIAL BASIC program that begins the external event routine. Upon the completion of this routine, control is returned to the original point of departure from the main program. New INDUSTRIAL BASIC statements for external-event driven control follow.



TIMER...

...associates a time interval with an external event routine. When the specified interval has elapsed, the routine is scheduled for execution, and the timer restarted. A maximum of four software timers may be active simultaneously. Time intervals that are zero or negative deactivate all the timers associated with the specified line number. This allows the user to change the elapsed time intervals or stop the timers entirely.

Examples of TIMER statements:

200 TIMER 10 THEN 400	Here, every 10 seconds the external event routine beginning at line 400 will be executed.
300 TIMER 0 THEN 400	In this statement, all timers associated with line 400 of the external event routine are deactivated.
100 TIMER 7 THEN 400 110 TIMER 3 THEN 400	These statements cause 2 timers to start; a 7-second timer that will cause the external event routine at line 400 to be scheduled for execution every 7 seconds, and a 3-second timer that will cause the external event routine at line 400 to be scheduled for execution every 3 seconds. At 21-second intervals, the external event routine is scheduled twice.

The following is an example of a time-based operation:

```
 10 TIMER 10 THEN 400
 20 TIMER 15 THEN 500
 25 LET Y=0
 30 GOTO 25
400 PRINT "SEGMENT 400"
410 DISMISS
500 PRINT "SEGMENT 500"
510 DISMISS
600 END
```

This operation will print "SEGMENT 400" every 10 seconds, and "SEGMENT 500" every 15 seconds until terminated.

CONTACT...

...associates a change in a switch position with the scheduling of an external event routine. A maximum of three modules (36 points) of UDC contact interrupts is permitted. The variable in a CONTACT statement is the line number associated with the specified CONTACT.

```
200 CONTACT 1 THEN 600
300 CONTACT 2 THEN 600
400 CONTACT 3 THEN 700
```

After these statements are executed, a change of state in CONTACT 1 or 2 will cause the external event routine at line 600 to be scheduled. A change of state in CONTACT 3 will schedule the external event routine beginning at line 700.

COUNTER...

...associates the completion of a counting operation with a specified line number, the first line of an external event routine. Four UDC counter modules are supported. As with the TIMER statement, when the counter number is zero or negative, all counters associated with the line number are disabled. Counter modules should be loaded via the CNO function for counting operations.

Example of a COUNTER statement:

```
200 COUNTER 1 THEN 1000
```

This statement schedules the external event routine beginning at line 1000 when counter 1 counts to zero. Following is an example of a program to load a counter with a number of items to count and log the completion of the count:

```
 10 COUNTER 1 THEN 100
 20 REM NOW LOAD COUNTER
 25 LET A=CNO (1,100)
 30 LET A=0
 35 GOTO 30
100 PRINT "COUNT CYCLE COMPLETE"
110 DISMISS
900 END
```

DISMISS...

...terminates execution of an external event interrupt routine. This statement causes the mainline BASIC program to resume execution. If another user process interrupt routine is scheduled, it will receive control at this point. DISMISS performs an action similar to RETURN.

External Event Routines—Control Simplified

The whole point of servicing external interrupts via INDUSTRIAL BASIC is to simplify resolution of the control algorithm, provide all the possibilities necessary to operate and resolve conflicts automatically... in short, to make the job of industrial control much easier than it has been in the past. Multiple interrupts, conflicting timers, counters that complete their jobs simultaneously, can all be resolved by the operating system and external event routines. So that ultimately, you have more time to devote to major system considerations. In fact, the entire INDUSTRIAL BASIC solution to control makes the task of designing, developing, and implementing the control system in high-level language so easy—it's almost elementary.

File Statements and Chaining

INDUSTRIAL BASIC also provides a number of statements for operating on files: FILE #, PRINT #, INPUT #, CLOSE #, and IF END #. These statements are distinguishable from other INDUSTRIAL BASIC statements by the number sign (#) at the end of each statement.

The FILE # statement opens a file; the file can be of fixed or variable length and in numeric format (files stored as successive three-word floating-point numbers with the last word in each block unused), or ASCII format (files stored in standard OS/8 format—three 8-bit characters to every two words). The PRINT # statement writes data into files. INPUT # reads data from files. CLOSE # functions as the closure after a file has been searched for data or as a closure for all output files before ending the program in order to prevent the loss of data. IF END # allows the user to determine whether or not there has been an end-of-file detected for the file in question. In general, file operations permit the logging of data for later reduction, or the use of prepared files as control information (recipes, tables, etc.).

In chain-handling, the CHAIN statement provides a convenient means of dividing large programs into a series of smaller programs which are written and stored separately, and executed in a chain. When INDUSTRIAL BASIC encounters a CHAIN statement, it stops execution of that program, retrieves the program named in the CHAIN statement from the specified device and file, compiles the chained program, and begins execution of the program. This operation is the equivalent of running an OLD program with no header (RUNNH).

INDUSTRIAL BASIC Operating and Editing

Certain commands enable the user of INDUSTRIAL BASIC to edit and control execution of programs by erasing characters or lines, listing all or any part of a program, saving programs on various storage devices, and calling programs from storage devices.

Corrections can be implemented very simply by using the SHIFT/O, the RUBOUT, or the CTRL/U commands, depending upon whether it is desirable to stop a long listing, delete whole lines or single characters. The RESEQ program allows the re-sequencing of line numbers when, after extensive program modifications, the user wishes a greater increment between line numbers.

An entire program, with program heading, can be listed on the terminal by typing the LIST command. Also, any portion of a program can be retrieved by using LIST followed by a line number. This causes that line and all following lines in the program to be listed. The LISTNH command is used as the LIST command, but eliminates the heading from the listed printout.

RUN and RUNNH (with or without program header) cause the program to run after it has been typed or loaded into core via the OLD command.

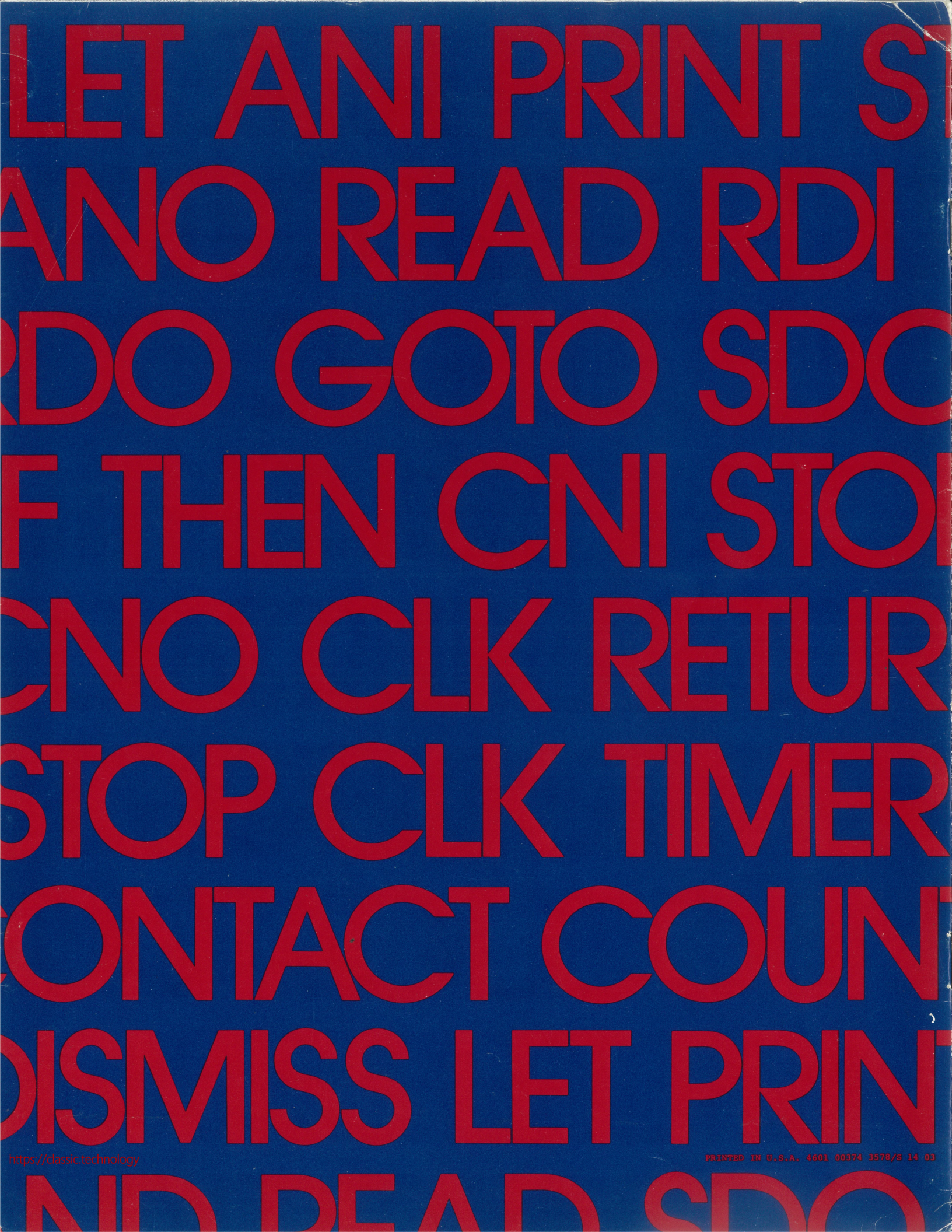
Hardware for INDUSTRIAL BASIC

The minimum hardware configuration supported by INDUSTRIAL BASIC is a PDP-8 with 8K of core, a UDC I/O interface, and a real time clock, with either dual DECtape or a cartridge disk as a systems device. INDUSTRIAL BASIC is offered in the Industrial 801-B and 811-B Systems , shown on the opposite page. Other devices supported are a DECcassette system, DECwriter, Teletype, Extended Arithmetic Element, Power Fail, and additional core memory.

In addition to INDUSTRIAL BASIC, the Industrial-800 Systems include the OS/8 software package. Installation assistance, training, and software support are also provided. For more information, contact your local field sales office; locations follow.

DIGITAL EQUIPMENT CORPORATION, Maynard, Massachusetts, Telephone: (617) 897-5111 • ARIZONA, Phoenix • CALIFORNIA, Sunnyvale, Santa Ana, Los Angeles, Oakland, San Diego and San Francisco (Mountain View) • COLORADO, Engelwood • CONNECTICUT, Meriden • DISTRICT OF COLUMBIA, Washington (Riverdale, Md.) • FLORIDA, Orlando • GEORGIA, Atlanta • . ILLINOIS, Northbrook • INDIANA, Indianapolis • LOUISIANA, Metairie • MARYLAND, Riverdale • MASSACHUSETTS, Cambridge and Waltham • MICHIGAN, Ann Arbor and Detroit (Southfield) • MINNESOTA, Minneapolis • MISSOURI, Kansas City and Maryland Heights • NEW JERSEY, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Huntington Station, Manhattan, New York, Syracuse and Rochester • NORTH CAROLINA, Durham/ Chapel Hill • OHIO, Cleveland, Dayton and Euclid • OKLAHOMA, Tulsa • OREGON, Portland • PENNSYLVANIA, Bluebell, Paoli and Pittsburgh • TENNESSEE, Knoxville • TEXAS, Dallas and Houston • UTAH, Salt Lake City • WASHINGTON, Bellevue • WISCONSIN, Milwaukee • ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Crows Nest, Melbourne, Norwood, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BRAZIL, Rio de Janeiro, Sao Paulo and Porto Alegre • CANADA, Alberta, Vancouver, British Columbia; Hamilton, Mississauga and Ottawa, Ontario; and Quebec • CHILE, Santiago • DENMARK, Copenhagen and Hellerup • FINLAND, Helsinki • FRANCE, Grenoble and Rungis • GERMANY, Cologne, Hannover, Frankfurt, Munich and Stuttgart • INDIA, Bombay • ISRAEL, Tel Aviv • ITALY, Milano • JAPAN, Osaka and Tokyo • MEXICO, Mexico City • NETHERLANDS, The Hague • NEW ZEALAND, Auckland • NORWAY, Oslo • PHILIPPINES, Manila • PUERTO RICO, Miramar and Santurce • REPUBLIC OF CHINA, Taiwan • SCOTLAND, West Lothian • SPAIN, Barcelona and Madrid • SWEDEN, Solna and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Edinburgh, London, Manchester, Reading and Warwickshire • VENEZUELA, Caracas

Industrial 811-B



PRINTED IN U.S.A. 4601 00374 3578/S 14 03