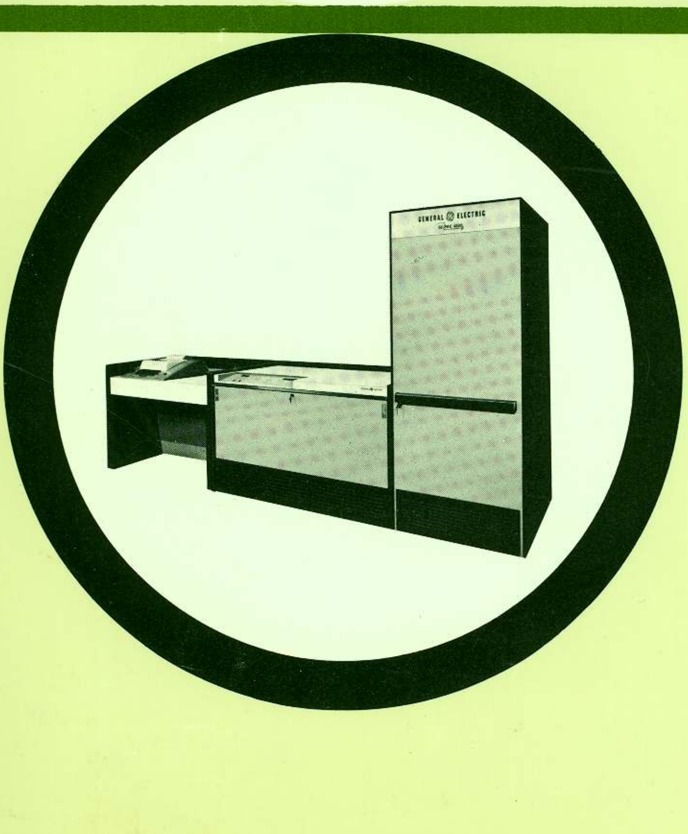
Box #1

GET-3471

GE PAC* 4020

COMPACT COMPUTER FOR PROCESS CONTROL

PROGRAMMING MANUAL



GENERAL ELECTRIC

*Reg. Trademark of General Electric Co.

CONTENTS

INTRODUCTION

SCOPE

This manual will familiarize the GE-PAC 4020 programmer with the registers, commands and techniques used to solve the computational and logical portions of typical process control computing problems.

All input/output functions in the GE-PAC 4020 computer are performed by the Real-Time Multiprogramming Operating System (RTMOS). RTMOS does the actual code conversion, queueing, diagnosing, and machine-level I/O work. The philosophy and functions of RTMOS are described on page 5 of this manual. For the details of RTMOS, consult the GE-PAC 4020 RTMOS manual (YPG 53M). This and other manuals of interest to the GE-PAC 4020 programmer are described below:

1. The Instruction Reference Manual - YPG10M. This manual shows the detailed instruction formats, the octal operation codes, the timing, the effects on each register, and other details of each PAL instruction, arranged alphabetically.

2. The GE-PAC Process FORTRAN Manual - YPG14M. This publication presents a second language in which the GE-PAC 4020 computer may be programmed. It assumes a knowledge of basic off-line FORTRAN and concentrates on the added features and capabilities of GE-PAC Process FORTRAN.

3. The GE-PAC 4020 System Manual - GET-3460. This manual covers the characteristics, specifications and machine-level programming details of input/output subsystems and devices, as well as remote communications between GE-PAC remote scanners and other computers.

FEATURES AND CHARACTERISTICS

The GE-PAC 4020 computer follows in the tradition of earlier GE-PAC 4000 series central processors (the 4040, 4050, and 4060) and extracts from them those elements proved by experience to be of most value in process control and general computational work. Combining these features with the higher speeds, reduced size and simplified construction possible with monolithic integrated circuits, the GE-PAC 4020 computer offers an unequalled combination of speed, ease of programming, reliability, and attractive environmental and mechanical characteristics.

The GE-PAC 4020 computer offers the user GE-PAC Process FORTRAN as well as Process Assembler Language, called PAL. Where running time and memory requirements are not critical, the programmer can enjoy the ease of coding, documentation and program modification inherent in Process FORTRAN. Alternatively, the programmer can gain the advantages of assembly-level programming — maximum efficiency in running time and program size and maximum control of machine operations. The programmer may also mix PAL and Process FORTRAN on a statement-by-statement basis and take advantage of the best features of each language.

The GE-PAC 4020 computer's 24-bit word and extensive command repertoire help it perform any given function in less memory and less time than competitive short-word machines. The 24-bit word can directly address up to 16,384 core locations with one instruction, can hold two 12-bit process readings or limits, and provides precise single- or double-word floating-point arithmetic capability. Even fixed-point arithmetic is comparatively easy to program thanks to the simplicity of long-word scaling.

Related GE products compatible with the GE-PAC 4020 computer include the GE 100, 200, 400 and 600-series business (and scientific) computer systems, GE-MAC* instruments, GE-TAC telemetering and remote supervisory equipment, Directo-Matic* II wired program analog and digital control systems, Mark Century numerical control, X-ray emission gauges, and the full line of GE communication and microwave products. To meld these products into systems, several organizational components of General Electric such as the Industrial Process Control Division, the Internal Management Operation, and the Industrial Drive Systems Division have the application engineers and knowhow to provide working systems on schedule and within budget.

*Reg. trademark of General Electric Co.

INTERNAL ORGANIZATION OF THE CENTRAL PROCESSOR

GE-PAC 4020 computer systems are organized as shown in Figure 1. The following discussion examines each functional block in more detail.

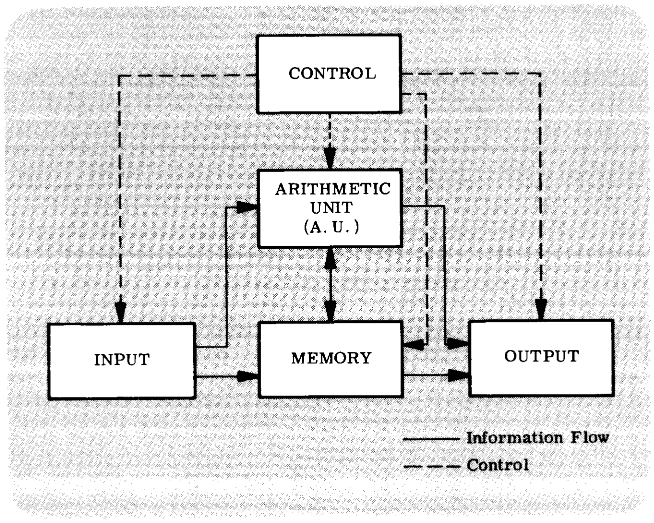


Figure 1

CONTROL

The control function governs the operation of system hardware. Knowledge of its operation is seldom necessary to a programmer.

MEMORY

The GE-PAC 4020 central processor includes a fast access (1.6 μs) magnetic core memory, which is available in 8K, 12K, 16K, 24K or 32K sizes. Each word contains 24 bits of information, plus an additional odd-parity bit which is generated when the word is written in core and checked upon retrieval.

As an auxiliary to core memory, bulk storage devices are normally used to provide storage for large volumes of programs and data when they are not required in core. The following bulk storage devices are available with the GE-PAC 4020 computer system:

- Magnetic Drum

  capacity - 16,384 to 262,144 words per controller

  typical access time - 8.3 ms

  maximum access time - 16.6 ms

  transfer rate - 15,360 or 30,720 24-bit words per second

- Magnetic Disc

  capacity - 1,048,576 to 8,388,608 words per controller

  typical access time - 87 ms

  maximum access time - 180 ms

  transfer rate - 40,960 24-bit words per second

- Magnetic Tape

  available in standard speeds and packing densities

All of these devices communicate directly with core memory. Odd parity is generated and checked by drum and magnetic tape controllers to insure reliable data transfer. The disc controller generates and verifies checksums of both addresses and data.

ARITHMETIC UNIT

All arithmetic and logical data manipulation takes place within the arithmetic unit, which also performs medium-speed input and output.

To perform these functions, the programmer uses registers in the arithmetic unit. A register is a group of related flip-flops, each of which can hold either a one or a zero. If a flip-flop is set, it contains a one; if it is reset, it contains a zero.

24 Flip-flops

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Flip-flop

1 = Set
0 = Reset

24 Bit Register

This method of showing the contents of registers will be used throughout this manual — the shaded portion indicates the relative position of the flip-flop, or bit, within the word; the unshaded portion indicates the state of that bit (zero, or one).

Shown below is a simplified diagram of the principal registers in the arithmetic unit and an explanation of their purpose.

B register — a 24-bit (flip-flop) register which acts as a buffer between core memory and the arithmetic unit.

A register — also 24 bits long; the computer's primary data register. Most commands affect its contents.

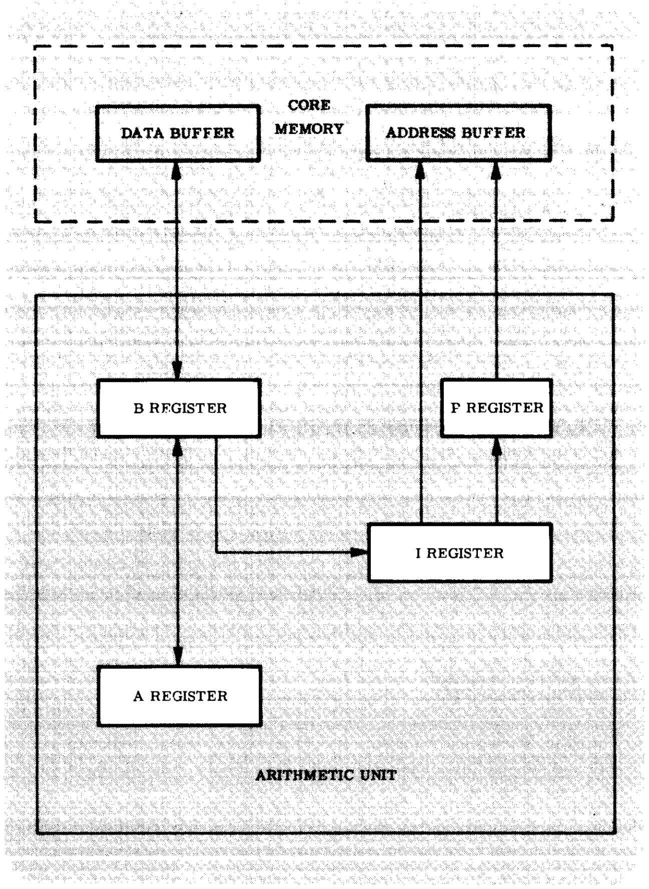


Figure 2

Q location (not shown) — a 24-bit memory location that acts as an extension of the A register.

I register — a 25-bit register; holds the instructions during their execution.

P register — a 15-bit register; holds the address of the next instruction to be executed.

Besides these arithmetic unit registers, there are also seven index, or X, locations, which are core locations one through seven. These aid in the execution of program loops.

INPUT AND OUTPUT

The GE-PAC 4020 computer permits I/O from memory through two types of channels: direct memory access channels and medium-speed channels through the arithmetic unit. Direct memory access channels permit data transfer to and from core memory at a maximum speed of 625,000 words per second. Because these channels are so fast, they are normally used only for the transfer of data between core and bulk memory, and between core memory and the arithmetic unit.

Since direct memory access channels require complex controllers and are needed only for the efficient operation of high-speed peripherals and bulk storage devices, most I/O devices use medium-speed I/O channels through the arithmetic unit. These channels transfer information into or out of core memory through the B register of the arithmetic unit at maximum speeds of 29,000 or 56,000 words per second. These channels may be dedicated to a single, medium-speed device, or shared by several lower-speed devices through an I/O buffer as shown in Figure 3. This arithmetic unit I/O arrangement coupled with unique TIM/TOM (Table Input to Memory/Table Output from Memory) hardware enables the GE-PAC 4020 computer to drive the peripherals simultaneously at full-rated speed, while requiring a minimum of central processor time.

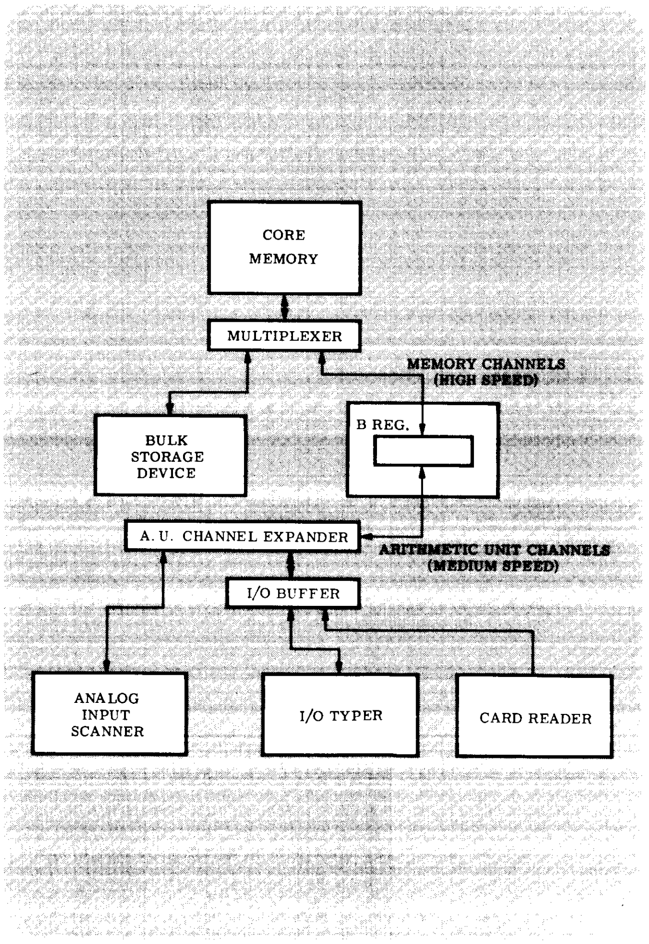


Figure 3

Process I/O subsystems or programmer peripherals can also communicate with the GE-PAC 4020 computer by means of its A register. This feature is used only for a few special functions and for situations where compatibility with the earlier GE-PAC 4040 machine is important, because it requires much more central processor time than the normal TIM/TOM operations.

CHAPTER 3

THE REAL-TIME MULTIPROGRAMMING OPERATING SYSTEM

There are two general types of software in a GE-PAC 4020 system – a process control system and an operating system. The process control system usually consists of many functional programs which perform the brainwork necessary to control a particular process. They typically examine inputs, perform computational and logical operations, and call for output. Functional programs vary in importance, some being more critical to system operation than others.

Operating systems vary significantly in scope and efficiency. They may only consist of a program to schedule the running of functional programs, or they may also perform a variety of other functions. The GE-PAC 4020 Real-Time Multiprogramming Operating System is the most complete process control operating system available today. RTMOS consists of a custom-tailored combination of standard program modules that correspond to the peripherals, subsystems, and functions of a particular system. A typical RTMOS schedules the running of programs, determines which programs should be in core at a given time, supervises process, peripheral, and bulk I/O, and performs a number of other useful functions to aid the programmer and increase the efficiency and reliability of the system. The philosophy and functions of the RTMOS are described briefly below. Consult the GE-PAC 4020 RTMOS manual (YPG53M) for further information.

PHILOSOPHY AND FUNCTIONS OF RTMOS

In the RTMOS multiprogramming environment, each functional program is assigned a priority relative to all other programs. Programs generally reside on bulk memory (drum or disc) until brought into core to be run.

An on-line snapshot of the GE-PAC 4020 computer's core memory at a given instant would reveal an unpredictable combination of programs in various stages of completion. One would be using the arithmetic unit, one might be coming in from bulk memory, and the others would either be awaiting their turn or using peripherals or process input/output devices.

This mode of operation assures a high load factor on all system resources – central processor speed, core capacity, bulk transfer and I/O speeds – and also assures that the system is working on the most important combination of programs.

RTMOS decides what mixture of programs to have in core and which one to execute on the basis of time, events and program priorities. It reviews this critical decision every time one of three things happens: 1/4 second passes; a bulk/core transfer is completed; or a functional program finishes.

The highest priority program that wants to run and is in core gets control of the central processor and keeps it until it calls for input/output, delays itself, shuts itself off, or until the RTMOS finds that another program has become more important.

In some cases, RTMOS may decide to overwrite low-priority programs and use their core area to run a higher-priority program. Before overwriting a program RTMOS will save selected registers and, if necessary, any intermediate data used by the program and store this on bulk memory. It does not waste time, however, by rewriting the entire program back onto bulk. After the high priority work is done, RTMOS reloads the overwritten programs from bulk memory, restores all registers and intermediate data, and continues where it left off.

Flexibility is the hallmark of RTMOS. The programmer can decide – statically or dynamically – the relative priority of his program, whether he wants to permit it to be interrupted or overwritten, how much temporary storage he wants to save, and whether or not it should be executed from a fixed core area. By adjusting all these and other parameters, the programmer can optimize system performance.

RTMOS takes care of these input/output functions: data or new program input from cards or paper tape; process instrument readings; analog and contact process control inputs and outputs; printed and punched outputs; and functions of GE-PAC Remote Scanners.

RTMOS also permits running programs to delay themselves, shut themselves off, share data areas, and run large bulk-resident subroutines.

The GE-PAC 4020 programmer instructs RTMOS to perform these functions by inserting special calling sequences at appropriate points in his program, saving himself time, work and confusion while assuring high overall system efficiency and performance.

An on-line operator package is also available with the RTMOS. This allows on-line access to memory through an input/output typewriter. It permits on-line core dumps, loading, and, with its on-line memory change capability, allows the operator to turn programs on or off and make other system modifications.

GE-PAC PROGRAMMING LANGUAGES

A program is a group of instructions describing a task to be performed by a computer. Since the computer cannot interpret words and sentences, instructions must be presented to it in the form of binary numbers. Since it would be a formidable exercise to write programs in binary, assembly languages like PAL exist to make programming easier.

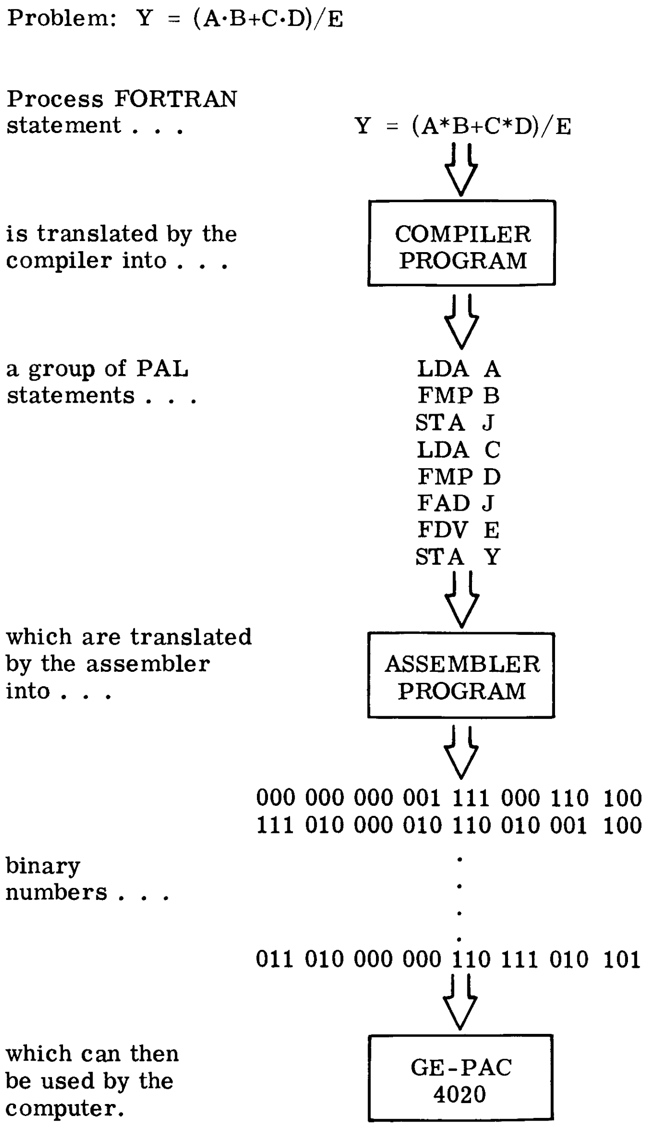
PAL is one step removed from binary code. It is a symbolic language meaningful to the programmer and, with some decoding, to the computer. The decoding is done by a computer program called an assembler which translates PAL coding into binary instructions and constants for the computer.

To make programming even easier, compiler languages like Process FORTRAN have been developed. Process FORTRAN allows programmers to write in a simple English/engineering language. GE-PAC Process FORTRAN includes powerful bit test and bit manipulation instructions. It is possible to mix PAL and Process FORTRAN statements to take advantage of the best features of each language.

Process FORTRAN programs are translated into PAL by a program called a compiler. The resulting PAL program is then translated into binary machine code by the assembler. Since this manual deals with programming in PAL, consult the GE-PAC FORTRAN Reference Manual for detailed Process FORTRAN information.

The GE-PAC 4020 computer uses binary machine code, PAL, and Process FORTRAN. The sequence in which these languages are decoded is shown in Figure 4. System programs are written in PAL, Process FORTRAN, or a combination of the two. The following sections describe the PAL instruction repertoire and some of its uses.

Before beginning these sections it may be helpful to review flow-charting and computer number systems since they will be referred to throughout the manual. The Special Discussion section in the back of the manual provides adequate review material.



Problem: Y = (A·B+C·D)/E

Process FORTRAN statement . . .

```
Y = (A*B+C*D)/E
```

is translated by the compiler into . . .

COMPILER PROGRAM

a group of PAL statements . . .

```
LDA  A
FMP  B
STA  J
LDA  C
FMP  D
FAD  J
FDV  E
STA  Y
```

which are translated by the assembler into . . .

ASSEMBLER PROGRAM

binary numbers . . .

```
000 000 000 001 111 000 110 100
111 010 000 010 110 010 001 100
                .
                .
                .
                .
011 010 000 000 110 111 010 101
```

which can then be used by the computer.

GE-PAC 4020

Figure 4. Simplified Compilation and Assembly Procedure

PROGRAMMING IN PAL

GENERAL RULES

PAL programs are written on the GE-PAC Language Statement Coding Form shown in Figure 5. One instruction is written on each line of this form. Once a program is written, it is keypunched onto data processing cards and processed by the GE-PAC 4020 PAL assembler program. The output of this program is a list of binary-coded instructions and data which are executed when the program is run.

Uses of the various fields of the coding form are described below.

LOCATION FIELD (Columns 1-6) - The location field associates a name with the address of the instruction or data written on that line. Names used in the location field may consist of up to six alphanumeric characters starting in column one. The first character must be alphabetic. Numbers, letters, and decimal points are the only characters normally used in a location name. A name may be defined only once in the location field of each program.

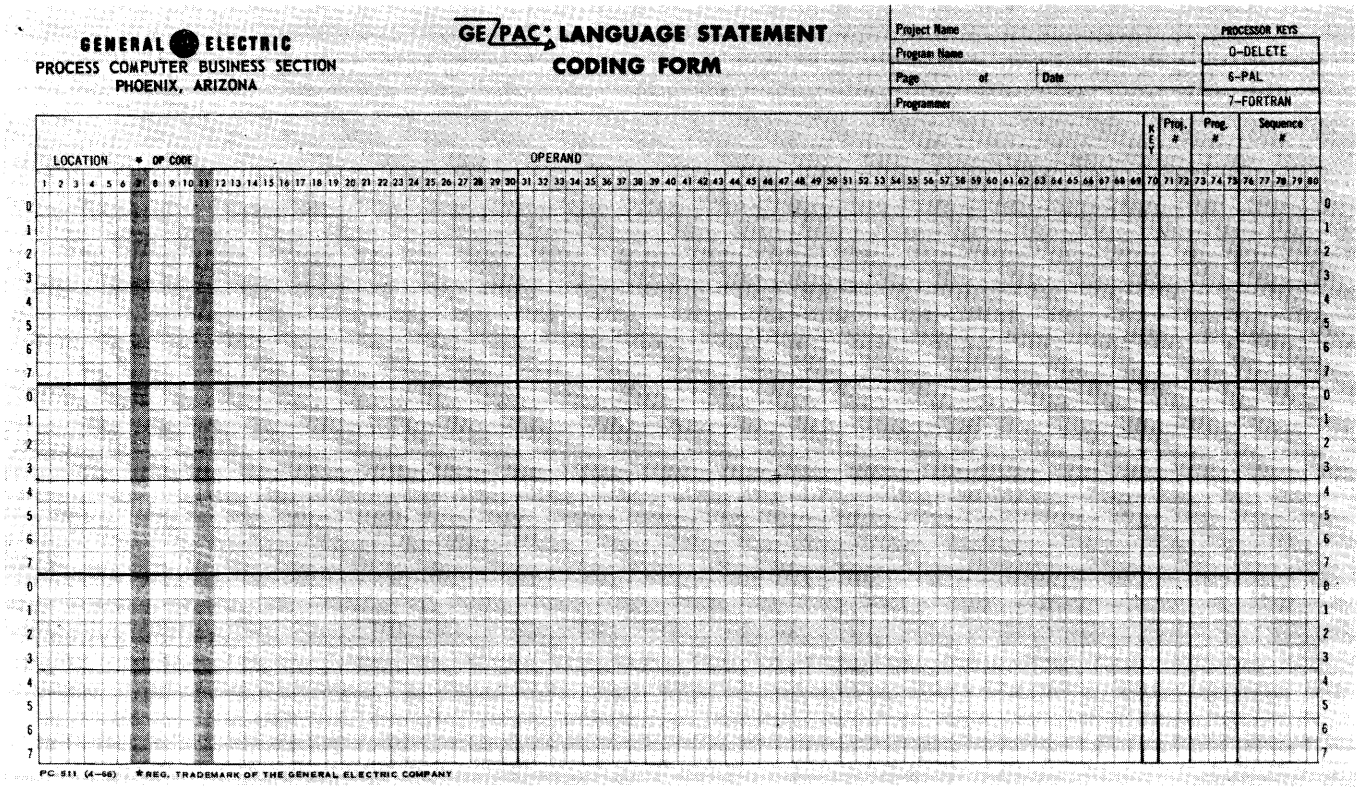
An asterisk (*) in column one of the coding form indicates that the rest of the line is a comment by the programmer and will not be interpreted by the assembler.

Example:

```
*    SCANNER VALIDITY CHECK
C317
PT04
```

LOCATION CLASSIFICATION (Column 7) - If this column is blank, the name in the location field is assigned a relative value; if Column 7 contains a minus sign (-), the name is assigned a specific, absolute value.

If an asterisk (*) appears in column seven, the name in the location field is assigned an absolute value, and the name is added to the common system symbol table.



GENERAL ELECTRIC
PROCESS COMPUTER BUSINESS SECTION
PHOENIX, ARIZONA

GE-PAC* LANGUAGE STATEMENT CODING FORM

Project Name
Program Name
Page of Date
Programmer

PROCESSOR KEYS
0-DELETE
6-PAL
7-FORTRAN

LOCATION * OP CODE OPERAND KEY Proj. # Prog. # Sequence #

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7

PC 511 (4-66) *REG. TRADEMARK OF THE GENERAL ELECTRIC COMPANY

Figure 5. GE-PAC Coding Form

OP CODE FIELD (Columns 8 - 10) - This field contains the two or three character instruction code which identifies the operation to be performed.

Example:

```
LDA
AKA
SUB
```

OPERAND FIELD (Columns 12 - 69) - The operand field contains the information required by the instruction in the OP CODE field. The operand field may contain any of the following types of parameters:

LABEL - Same as permitted in location field
DECIMAL - A decimal integer value
OCTAL - An octal integer value, preceded by a slash (/).

EXPRESSIONS - An operand may be composed of one or a combination of the parameters described above. These are combined using the following operators: + add, * multiply, - subtract, / divide. Expressions are evaluated by the PAL assembly program.

The meaning of asterisk or slash depends upon its relationship to the other parameters. For example, an asterisk represents multiplication only if it is connecting two parameters. Otherwise, it indicates the address of the instruction in which it appears, for use in relative addressing. The slash indicates an octal value when it precedes a numeric parameter; otherwise it is a division sign.

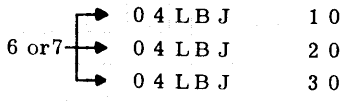
The first blank space in the operand field terminates the assembly of the instruction. Characters appearing after the first blank space are treated as comments.

Example:

```
TEMP1       }  LABELS
PTØZZ       }
999         }  DECIMAL
0           }  INTEGERS
/77776      }  OCTAL
/24         }  INTEGERS
PØINT+10    }
NEXT-1/2    }  EXPRESSIONS
VALUE+1     }
```

KEY (Language Identification - Column 70) - must contain a seven if the statement is written in GE-PAC Process FORTRAN; a six if it is a PAL statement.

PROJ. #, PROG. #, SEQUENCE # - may be used for identification or left blank.



PAL PSEUDO-INSTRUCTIONS

Before a PAL program is run it must be translated into binary machine language by the assembler. It is frequently necessary to tell the assembler to define constants, build storage areas, etc. This is done with pseudo-instructions. Pseudo-instructions are executed only when the program is assembled.

The following pseudo-instructions are commonly used.

SINGLE-WORD CONSTANTS

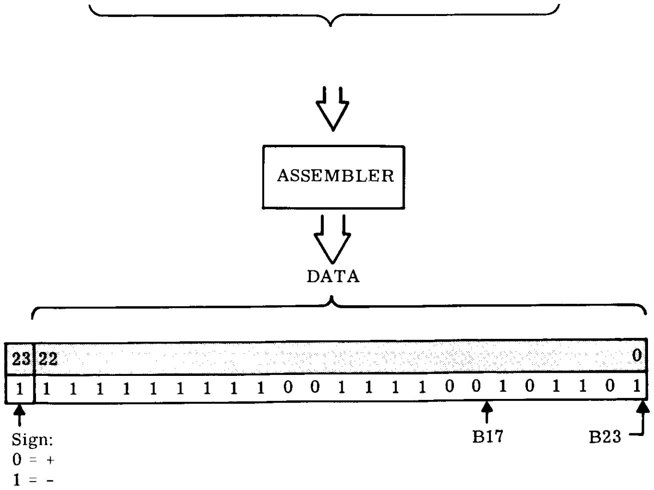
Single-word decimal constants are created in a program with the following pseudo-instruction.

CON D, (decimal number) (scale factor)

FIXED POINT DECIMAL CONSTANT - The number specified in this statement is converted by the assembler into a binary fixed point number in the word format shown below. The CON D pseudo-instruction is removed from the program after the conversion has been made. The resulting data are stored in its place. Negative numbers are represented in two's complement form.

Example:

```
CØN   D, -195.3B17
```



The B factor, or scale factor*, indicates where the data are to be positioned within the word. If B is not specified it is assumed to be B23.

This conversion is done only once, when the assembler executes a CON D pseudo-instruction. The constant generated by the instruction is the only thing used by the computer when it runs the program.

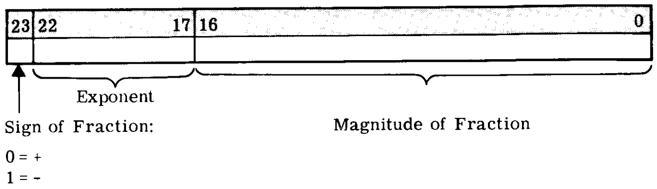
*See page 22 for discussion of scaling.

Examples:

```
CØN  D,3975
CØN  D,35.04B17
```

CON F, (decimal number)

FLOATING CONSTANT - The number specified is converted to a binary floating-point number and placed in a format as shown below.



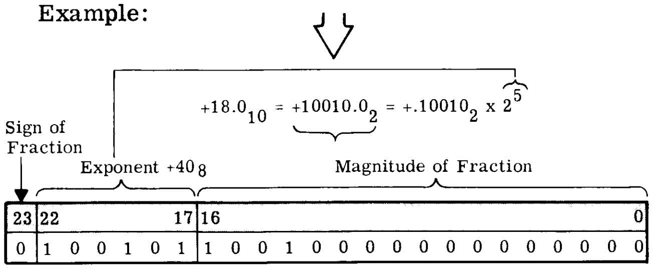
The fraction field contains the normalized (left justified) fraction in binary form.

The exponent field contains the binary exponent of the fraction. Since this field has no sign bit, 40_8 is considered to be an exponent of zero; 41_8 through 77_8 represent positive exponents; and 0_8 through 37_8 represent negative exponents. This is illustrated in the table below. Zero in floating point is represented by all zeroes.

EXPONENT	VALUE IN EXPONENT FIELD
:	:
2	42_8
1	41_8
0	40_8
-1	37_8
-2	36_8
.	

Example:

```
CØN   F, 18
```



Since most decimal/binary conversion is performed by the assembler and RTMOS, the programmer uses it only in those few instances when he must examine numbers at the binary level.

Examples:

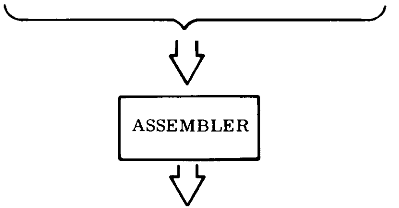
```
CØN  F, . 12E5    (.12E5 = .12 x 10+5)
CØN  F, 9
CØN  F, -8. 14E-2
```

The following pseudo-instruction is frequently used to set up a desired bit pattern in a word.

CON O, (octal number)

OCTAL CONSTANT - The octal number specified is converted to binary by the assembler and placed in the format shown below. Octal constants must be unsigned octal integers. A maximum of eight octal digits (one 24-bit word) may be defined by a single CON O.

```
CØN  Ø, 6712
```



23																							0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	1	0	1	0

DATA

(There is no sign)

Examples:

```
CØN  Ø, 6
CØN  Ø, 31
CØN  Ø, 77777777
```

It is often necessary to insert alphanumeric constants into your program to explain information output from the computer. To have the computer type out 92.3 is meaningless. It needs to be explained with a statement. When the computer types out "TEMPERATURE OUTSIDE IS 92.3 DEGREES" the number becomes meaningful. These explanatory statements are supplied with alphanumeric constants by the pseudo-instruction described below. The use of this pseudo-instruction is explained in the RTMOS manual.

CON A, (no. of characters), (characters)

ALPHANUMERIC CONSTANT - The assembler translates these characters into the American Standard Code for Information Interchange (ASCII) and packs them three per 24-bit word.

The format for packing characters is shown below.

23		15		7	0
0		0		0	

Character Codes

Since a maximum of three characters may be packed into a single word, a single CON A may generate several words of information. Always specify the number of characters to be represented.
A maximum of 51 characters is allowed in a single CON A statement. However, several CON A statements may be strung together to form longer messages.

Example:

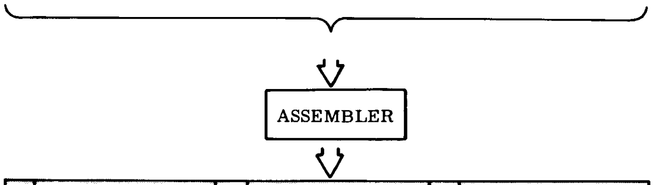
CON A, 16, VALVE(7) IS OPEN

Number of characters (including spaces)

Alphanumeric characters

To accommodate this statement the computer generates six sequential words of data as shown below. Incomplete words are filled out with delete codes.

```
CØN  A, 16, VALVE(7)  IS  ØPEN
```



23		15		7	0
0	V	0	A	0	L
0	V	0	E	0	(
0	7	0	)	0	SPACE
0	I	0	S	0	SPACE
0	Ø	0	P	0	E
0	N	0	DELETE CODE	0	DELETE CODE

EQUIVALENCE

Since many programmers find it easier to remember symbolic names than numbers, PAL includes the following pseudo-instruction to assign a value to a symbol without having to define it as an instruction label.

(symbol) EQL (integer number)

ASSIGN A SYMBOLIC EQUIVALENCE - Whenever the assembler finds that symbol in the program, it uses the numeric value assigned to it in place of the symbol. A symbol may be equated to another symbol as long as the symbol on the right is pre-defined in the program.

Examples:

```
START  EQL 20
A      EQL /732
PØINT  EQL /6320
```

GENERATING BLOCK STORAGE

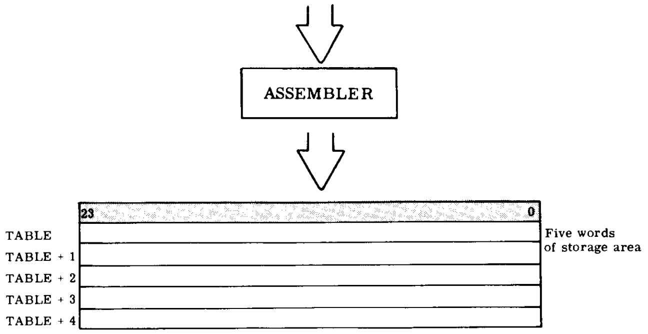
Programs frequently generate data which must be stored somewhere. The following pseudo-instruction instructs the assembler to generate this storage area.

BSS K

BLOCK STORAGE RESERVATION - This pseudo-instruction causes the assembler to reserve K words of storage area beginning at the location of the BSS word.

Example:

```
TABLE  BSS  5
```



SPECIFYING PROGRAM STARTING ADDRESSES

GE-PAC 4020 programs may be immediately preceded by a pseudo-instruction indicating where the program is initially to be placed. All-core systems use the ORG pseudo-instruction to do this. On systems with a drum or disc the DCW pseudo-instruction is normally used. These are explained below.

ORG (starting address)

PROGRAM STARTING ADDRESS - This pseudo-instruction causes the assembler to specify for the loader program the starting core address of the program.

DCW (bulk address), 0

PROGRAM STARTING ADDRESS - In a bulk/core system programs are usually loaded onto the drum or disc and brought into core just before they are run. This pseudo-instruction causes the assembler to specify for the loader program the beginning drum or disc address where the program is to be stored.

ENDING PAL PROGRAMS

END

END OF PROGRAM - Tells the assembler to stop. There must be an END statement after the last word of a PAL program, in columns 8 - 10 of the coding form.

OTHER PSEUDO-INSTRUCTIONS

The following pseudo-instructions are described in the Appendix.

DCN

DOUBLE WORD CONSTANT - Generates double word decimal, octal, or floating-point constants. The GENERAL constant is also discussed.

SLW

SLEW PRINTER PAGE - Shifts printer to next page when the assembly listing is printed.

DEF

DEFINE - Used to define new operations.

GEN

GENERATE DUPLICATES - Generates duplicate PAL instructions to save repetitive coding.

PAL FUNCTIONAL INSTRUCTIONS

LOADING AND STORING

Most PAL instructions operate on data located in the A register and/or the Q location. A and Q also hold the results of these operations. The following instructions are used to transfer information from core memory to A and/or Q before an operation, and to store the results back into core afterward.

LDA Y

LOAD A WITH C(Y) – The contents of core location Y, C(Y), are copied into the A register. The C(Y) are not changed.

STA Y

STORE C(A) IN Y – The contents of the A register are cqpied into core location Y. The C(A) are not changed.

LDQ Y

LOAD Q WITH C(Y) – The contents of core location Y are copied into location Q. The C(Y) are not changed.

STQ Y

STORE C(Q) IN Y– The contents of location Q are copied into location Y. The C(Q) are not changed.

Greater precision in arithmetic operations may be obtained using double-word operations. In these cases both the A and Q are combined to make up a 47-bit double-word. The following two instructions are used for the transfer of double-words to and from core memory.

DLD Y

DOUBLE LENGTH LOAD – The contents of core locations Y and Y+1 are copied into A and Q respectively. The C(Y) and the C(Y+1) do not change.

DST Y

DOUBLE LENGTH STORE – The contents of A and Q are copied into core locations Y and Y+1. The C(A) and the C(Q) do not change.

MEMORY ADDRESSING TECHNIQUES

Each 24-bit word in core memory has a unique address. In order to bring a word into a register to operate upon its contents it is necessary to specify its address. For example, if 'Y' specifies the exact address of a word in core memory, we may bring that word into the A Register with the following instruction:

LDA Y

This technique is called direct, or absolute, addressing. It is used primarily for communication with RTMOS, and other areas of permanent core memory. Besides direct addressing there are three other methods of addressing core: relative addressing, which is used within functional programs; automatic index modification, which is used to access elements of a table; and indirect addressing, which is rarely used.

Direct Addressing

In direct addressing of core memory, the number, symbol or expression in the operand field of a PAL statement refers directly to the address of the desired 24-bit word.

It is possible to directly address the first 16,384 words of core memory. Since the RTMOS resides at fixed locations in this area, all communications with it use direct addressing. Direct addressing is also used in program intercommunication, described on page 55.

Examples:

```
        LDA  /14372        C (/14372)→A Register

        LDA  VALVE-1   ⎫
                       ⎬   C (/1000) →A Register
VALVE   EQL  /1001     ⎭
```

Relative Addressing

In relative addressing, the number, symbol, or expression in the operand field specifies a core location relative to the address of the instruction being executed.

Relative addressing permits the addressing of 8191 locations above and 8192 locations below the point in core memory at which the instruction using it is located. It may be explicitly called for by placing an asterisk in the operand field of an instruction (normally in column 12). In this case, the number, symbol, or expression in the operand field is added at execution time to the address of the instruction, so that the result specifies the desired core address.

Examples:

```
        LDA  *+2      ——   This operand references..
        ---
        ---           ←——  The word at this location


        ---           ←——      this location ←
        ---
        LDA  *-2      ——→  This operand references..
```

Most functional programs use relative addressing for addresses within themselves, so that they may run correctly from any area of core memory. This capability is integral to the power of the RTMOS.

To maximize the use of relative addressing, the PAL assembler program generates relative addresses whenever possible.

Example:

The assembler will interpret this coding . . .

```
         LDA  ITEM
         ---
ITEM     CØN  F,6
```

as if we had written this . . .

```
         LDA  *+2
         ---
         CØN  F,6
```

Automatic Address Modification

The GE-PAC 4020 Computer uses seven index locations called "X" locations which are core locations one through seven. The first two are used for special purposes; the other five are available for general use.

In automatic address modification, the content of an X location, specified at the end of the statement, is added to the location specified by the number, symbol, or expression in the operand field. The sum of these specify the address of the desired word. Only one X location may be specified in a single statement.

Examples: Where C(4) = /10, and the LDA instruction is at core address /10000

```
         LDA  /20000,4          C(/20010)→A Register

         LDA  /16700,4          C(/16710)→A Register

         LDA  *+2,4             C(/10012)→A Register
```

A list of instructions that may use automatic address modification is included in the GE-PAC 4000 Instruction Reference Manual.

The primary use for automatic address modification is in looping through tables of data. This is discussed on page 43. There are also many other uses for indexed addressing and X locations that will become apparent later on.

The following instructions may be used to load and store the contents of X locations.

LDX Y,X

LOAD INDEX LOCATION X WITH THE C(Y) — The contents of core location Y are copied into the index location specified by X. The C(Y) are unchanged.

STX Y,X

STORE C(X) INTO Y — The contents of the index location specified are copied into core location Y. The C(X) are unchanged.

Indirect Addressing

Indirect addressing is seldom necessary in programming the GE-PAC 4020 computer because of its extensive direct and relative addressing capabilities. When indirect addressing is desirable, the following two instructions are used.

LDI Y

LOAD A INDIRECTLY—The rightmost 15-bits of the word in core location Y refer to a core memory location Z. The contents of core location Z are copied into the A register. The C(Y) and C(Z) are unchanged. Indexing and relative addressing, if used in the LDI instruction, are completed prior to fetching C(Y). If bit 18 of the word at location Y is set to one, the address specified within Y is relative to location Y.

STI Y

STORE A INDIRECTLY—The first 15-bits of the word in core location Y refer to a core location Z. The contents of this core location are replaced by the contents of the A register. The C(Y) and C(A) are unchanged. Indexing and relative addressing, when used in the LDI instruction, are completed prior to fetching Y. If bit 18 of the word at location Y is set to one, the address specified within Y is relative to location Y.

Indirect addressing is usually used to access areas of core memory above 16K. To do this the programmer specifies the location he wants in one core location and then addresses it indirectly. Since other addressing techniques are as powerful and more simply applied, indirect addressing is seldom used.

ARITHMETIC OPERATIONS

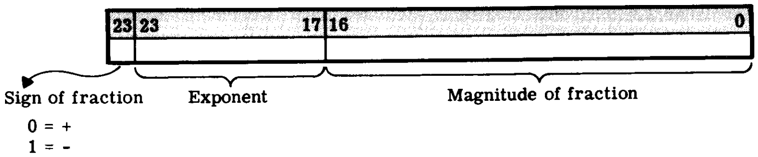
Add, subtract, multiply, and divide are the four arithmetic instructions used by the GE-PAC 4020 computer. They may be performed in either floating-point or fixed-point arithmetic. Because it is the simplest to use, floating-point is discussed first.

Floating-Point (Single-Word Precision)

Floating-point numbers are expressed as fractions raised to a power of ten, as opposed to fixed-point numbers which are expressed as decimals or integers.

Fixed Point		Floating Point
+18346	=	+.18346 X 10^{+5}
+18.346	=	+.18346 x 10^{+2}
-.0018346	=	-.18346 x 10^{-2}

At assembly time, decimal floating-point numbers are converted to binary floating-point form and stored in the format shown below.



A detailed explanation of this format is shown on page 11. However, since most decimal/binary conversion is done by the assembler and RTMOS, knowledge of formats is necessary only in those few instances when the programmer must examine numbers at the binary level.

Single-word floating-point arithmetic allows expression of numbers in a range of ±2.15 x $10^{\pm 9}$ with five decimal place precision.

The following instructions are used in floating-point arithmetic calculations. All numbers used in floating-point calculations must be in floating-point form.

FAD Y

FLOATING ADD — The contents of location Y are added to the contents of the A register. The result is placed in the A register. The C(Y) are not changed.

FSU Y

FLOATING SUBTRACT — The contents of location Y are subtracted from the contents of the A register. The result is placed in the A register. The C(Y) are not changed.

FMP Y

FLOATING MULTIPLY — The contents of location Y are multiplied by the contents of the A register. The result is placed in the A register. The C(Y) are not changed.

FDV Y

FLOATING DIVIDE — The contents of the A register are divided by the contents of location Y. The result is placed in the A register. The C(Y) are not changed.

Example:

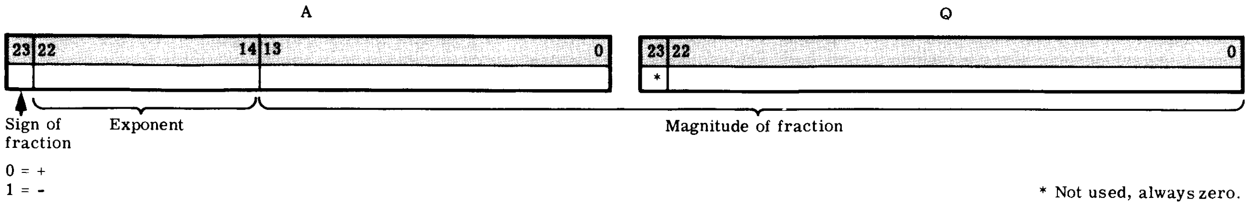
```
          LDA   FC5          5 ──→A reg
          FSU   FC3          5-3  =2 ──→A  reg
          FAD   FC8          2+8  =10 ──→A  reg
          FMP   FC6         10*6  =60 ──→A  reg
          FDV   FC3         60/3  =20 ──→A  reg
          STA   TEMP         C(Areg) ──→TEMP

FC5       CØN   F,5       ⎫
FC3       CØN   F,3       ⎪  DEFINITION OF
FC8       CØN   F,8       ⎬       CONSTANTS
FC6       CØN   F,6       ⎪         AND
TEMP      BSS   1         ⎭       STORAGE
```

Floating-Point (Double-Word Precision)

Greater arithmetic precision may be obtained using double-word floating-point operations. Double-word operations permit expression of numbers over a range of ±5.8 x $10^{\pm 76}$, with eleven decimal place precision.

Double-word computations use the same operation code as single-word. Selecting the mode (single- or double-precision) of floating-point arithmetic operations is discussed on page 20. To provide double-word precision the Q location is used as an auxiliary to the A register. The binary format for double-word instructions is shown below. Remember that all data used in double-word floating-point computations must be in double-word floating-point form.



* Not used, always zero.

The following describes the execution of the floating-point instructions in the double-precision mode.

FAD Y

FLOATING ADD — The double-word expressed in the 47 bits of core locations Y and Y + 1 is added to the double-word expressed by the combination of A and Q. The result is placed in A and Q.

FSU Y

FLOATING SUBTRACT — The double-word expressed in the 47 bits of core locations Y and Y + 1 is subtracted from the double-word expressed by the combination of A and Q. The resulting double-word is placed in A and Q.

FMP Y

FLOATING MULTIPLY — The double-word expressed in the 47 bits of core locations Y and Y + 1 is multiplied by the double-word expressed by the combination of A and Q. The result is placed in A and Q.

FDV Y

FLOATING DIVIDE — The double-word expressed in A and Q is divided by the double-word expressed in core locations Y and Y + 1. The result is placed in A and Q.

Shifting Between Single- and Double-Precision Operations

Since the same instructions are used to perform single and double precision floating-point operations, the following instructions indicate which mode is to be used when executing a floating-point instruction.

FMS 1

FLOATING MODE SHIFT TO SINGLE-PRECISION — All floating-point instructions following this instruction will be performed in single-word precision.

FMS 2

FLOATING MODE SHIFT TO DOUBLE-PRECISION - All floating-point instructions following this instruction will be performed in double-word precision. Any even number in the operand field causes a shift to double-word precision.

Example:

```
   :
FMS  1
LDA  RDNG1
FAD  CNST    ←—— Executed in single-word precision
STA  VALVE1
   :
FMS  2
DLD  RDNG2
FMP  DBLCØN  ←—— Executed in double-word precision
DST  VALVE2
   :
```

It makes no difference where the floating mode shift instruction is used. Floating-point instructions will be performed in the mode last specified. Be sure that the data used agrees with the mode specified. It is good practice to set the desired floating mode at the beginning of each program using floating-point instructions

Overflow

An occasional source of grief to the programmer is a condition called "arithmetic overflow." This occurs when the result of an arithmetic operation is too large to fit in the data portion of the A register, and a carry is propagated into the sign bit.

For example, the maximum number that can be stored in a single floating-point word is 2.15 x 10^9. If we multiply 2.0 x 10^5 by 2.0 x 10^5 we will obtain 4.0 x 10^{10} which is too large to be held in the A register. Overflow will occur. If we were working in double-word precision 2.0 x 10^{40} by 2.0 x 10^{37} would also give us an overflow. The GE-PAC 4020 computer detects overflow with an overflow flip-flop, named "OVRF." If overflow occurs OVRF is set to one. If no overflow occurs OVRF is unchanged. The following instruction will test the state of OVRF.

JNO

JUMP IF NO OVERFLOW — JNO tests the condition of the overflow flip-flop, OVRF. If it is set (overflow occurred), the next instruction in the program is executed and OVRF is reset. If OVRF is reset (no overflow) the computer jumps to the second sequential location.

Example:

```
LDA  RDNG
FMP  CØNST1
FAD  Y
FDV  CØNST2
JNØ
BRU  ØVRERR   ←—— Executed next if OVRF set
STA  Z        ←—— Executed next if OVRF reset
```

If Overflow Occurred — The instruction that immediately follows JNO is executed. Any instruction may be used here, but typically it is a branch to some program to correct, or at least note, the error.

If No Overflow — The computer skips over the first sequential instruction and continues on in the program from the second.

Overflow conditions are detected after the execution of FAD, FSU, FMP, FDV, ADD, SUB, MPY, DVD, AKA, SKA, FIX, FLO, SLA, and DLA instructions. These are the only instructions that may cause arithmetic overflow.

The JNO instruction is most helpful in debugging programs. To perform spot checks for possible overflow simply insert a JNO after each section of questionable arithmetic operations. Once any errors have been corrected, the JNO's may be removed. To check for overflow in debugged and running programs, reset OVRF at the beginning of the program and insert a single JNO at the end.

The procedure for resetting OVRF follows:

```
JNØ
NØP     ←—— No OPeration - Indicates that no instruction
             is to be executed. Computer moves to next
             instruction.
```

Overflow problems can be a headache unless they are caught in the debugging stage. Although there is no standard treatment for overflow, taking the following action during the debugging stages will eliminate most occurrences.

- Rearrange arithmetic instructions and rescale the data.
- Check to be sure the data source (analog scanner, etc.) is giving correct data. Data reasonability checks should be built into programs to help spot bad devices while a program is running.

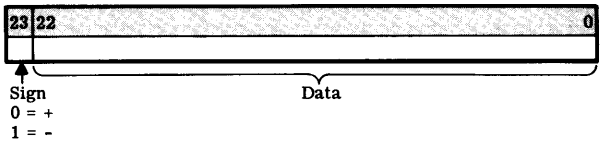
This should cure most overflow problems. If overflow still occurs occasionally, take one or both of the following actions:

- Print a message identifying the error and where it occurred.
- Recalculate, using the last valid data obtained.

Another, but less bothersome, problem is arithmetic underflow. This occurs when the result of an arithmetic operation is too small to be represented in a 24-bit register. To test for underflow, test the accumulator register for all zeroes (see page 39).

Fixed Point (Single-Word Precision)

Fixed-point numbers do not use exponents to indicate their magnitude. The following format is used to represent a fixed-point number in a GE-PAC 24-bit word. If a number is negative, the number is represented in two's complement form and bit 23 will be a one.



Decimal numbers are converted to binary within the computer and are stored in the data portion of the word. The exact position of the data within the word is determined by the B factor, or binary scale factor, which is specified by the programmer.

Fixed-point arithmetic offers two advantages over floating point. First, it is faster (see Figure 6).

Operation	Floating Point	Fixed Point
ADD	203.0 µs	3.2 µs
SUBTRACT	208.0 µs	3.2 µs
MULTIPLY	151.0 µs	8.9 - 12.1 µs
DIVIDE	182.0 µs	13.7 µs

Figure 6. Comparison of Execution Times

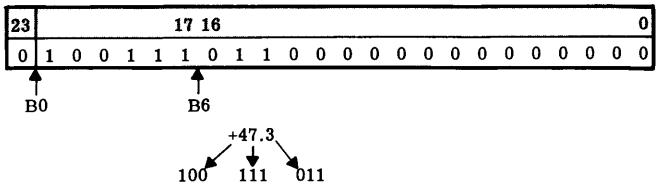
Second, it provides an extra 6 bits of precision — 17 bits for floating point vs. 23 for fixed point.

There are two drawbacks to using fixed-point arithmetic. First, it does not easily permit double-precision multiply and divide. Second, it is difficult to use. Because of this, programmers normally use it only when computer time is critical.

Fixed-point arithmetic requires that the programmer keep track of the decimal point, just as he must when using an adding machine, desk calculator, or paper and pencil.

Fixed-point quantities are represented in a GE-PAC computer by a sign bit and 23 bits of data. Positioning this data within a word is called "scaling." The number of bit positions between a number's binary point and the machine's binary point is called its scale factor, abbreviated to "B". The machine's binary point (B0) is located between the sign bit and the adjacent data bit.

Shown below is what 47.3_8 would look like if it were represented in a GE-PAC register at a scale factor (B) of 6:



It happens that B6 is the smallest scale factor that could be used, since putting this same number at B5 would have changed the sign bit to a one, erroneously suggesting a negative number. The maximum scale factor would be B20 in this case, and anywhere between B6 and B20 would be acceptable. Choice of the best scale factor for a number depends upon its possible range, and the calculations it will be subjected to. If a number is positioned at the minimum scale factor at which it is correctly represented, it is said to be left justified.

To add or subtract two fixed-point numbers, they each must have the same scale factor, or B. If they are different, one or the other must be shifted to put both at the same scale factor. The arithmetic shift commands which are explained later on will do this.

In multiplying, the B of the product is the sum of the B's of the factors; in dividing, the B of the quotient is the difference of the B's of the dividend and the divisor.

To preserve maximum precision, it is desirable to keep numbers as far to the left as possible without pushing data into the sign bit and causing overflow.

Negative B's occur when representing fractions with one or more significant zeroes. A negative B represents the distance to the left between B0 and the binary point of the fraction.

Positive (or negative) B's greater than 23 are also possible. Positive B's above 23 indicate use of only the 23 most significant data bits of the number (or less, if the number is not left justified).

Scale factors are not represented within the computer; there is no binary point or any other indication of their existence. The B factor simply describes how a binary number should be positioned within a word. The programmer must keep track of this positioning by associating a B with every fixed-point number. By comparing the B factors and the range of the numbers he can determine whether or not the result of an arithmetic operation will cause overflow. The computer will tell him if an operation does cause overflow, but the programmer himself must correct for it.

The following operation instructions are used to perform single precision fixed-point operations.

ADD Y

FIXED-POINT ADD — The contents of core location Y are added to the contents of the A register. The C(Y) are not changed. The scale factors of both numbers must be the same.

SUB Y

FIXED-POINT SUBTRACT — The contents of core location Y are subtracted from the contents of the A register. The C(Y) are not changed. The scale factors of both numbers must be the same.

MPY Y

FIXED-POINT MULTIPLY — The contents of core location Y are multiplied by the contents of the Q location. The result is placed in A and Q with the most significant half in A. The C(Y) are not changed. The scale factor of the result is the sum of the scale factors of the numbers being multiplied. Bit 23 of Q is not used and is always set to zero.

NOTE: As a part of the fixed-point multiply instruction the original contents of A are algebraically added to Q. Therefore, the program should normally load zeroes into the A register before multiplying. While this feature can be useful in computing equations such are ab+c, it is omitted from the example for clarity.

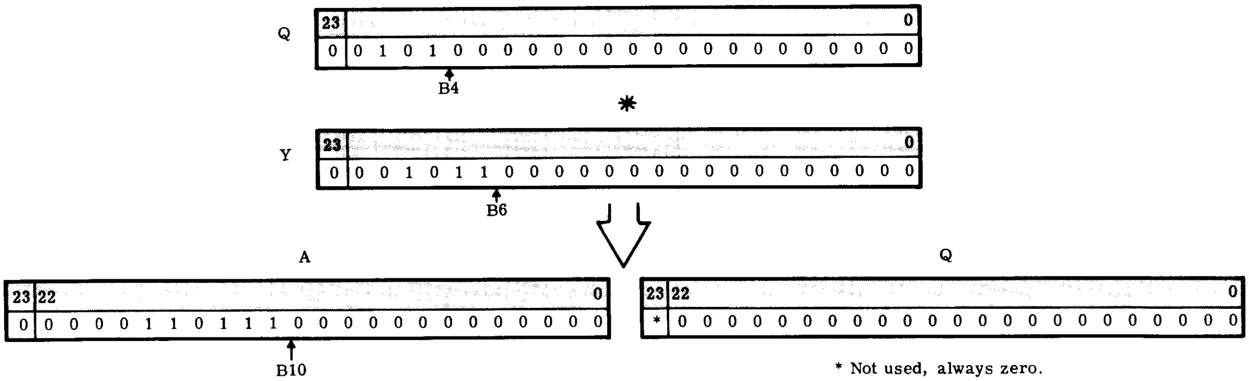
Example:

5 * 11

5 = /5 and resides in Q @ B4

11 = /13 and resides in Y @ B6

C(A Reg) = 0



The answer is /67 or 55, which appears in A and Q @ B10.

DVD Y

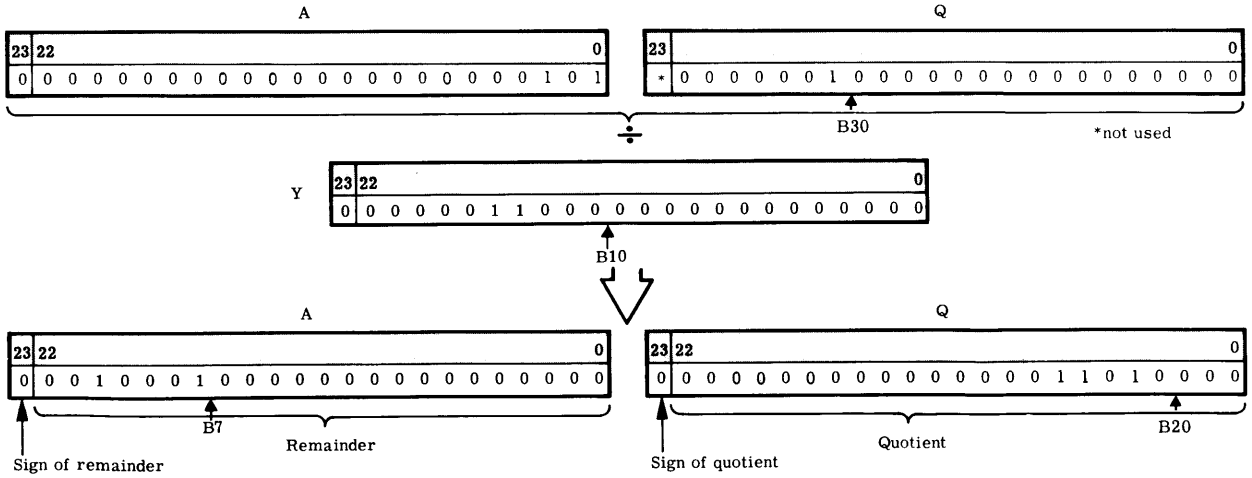
FIXED-POINT DIVIDE — The contents of the A and Q are taken as a single number and divided by the contents of core location Y. The quotient is placed in Q; the remainder in A. The binary scale factor of the quotient is equal to the scale factor of C(A and Q) minus the scale factor of the C(Y). The remainder carries the same scale factor as the dividend less 23. Its sign is the same as that of the divisor.

Example:

641 ÷ 24

641 = /1201 and resides in A and Q @ B30

24 = /30 and resides in Y @ B10



The answer is /32 with a remainder of /21, or 26 with a remainder of 17.

Use of Fixed Point Instructions

Example:

Compute: PRSRE = C1*RDNG + C2 where: $0 \leq$ RDNG ≤ 4000 @ B17

also determine B for C1 and C2 where: C1 = 2 B?

Working backward we first determine the B necessary to hold the result by computing: C2 = 43 B?

$$\text{PRSRE}_{\text{max}} = 2*4000 + 43 = 8043_{10}$$

$8043_{10} = 17553_8$ which requires B13. C1 may be scaled at B2. To add C2 to the results of C1*RDNG it must be at B19.

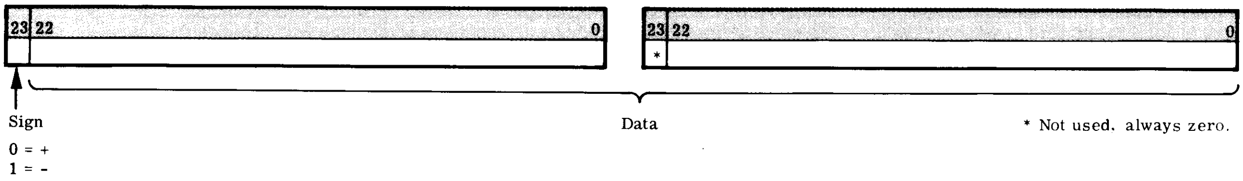
Assume RDNG is placed in the program @ B17 by some other program.

Example:

```
         LDA  ZERO      ZEROES→A  reg
         LDQ  RDNG        RDNG→Q  reg  @B17
         MPY  C1        C1*RDNG→A&Q  @B19
         ADD  C2          @B19
         STA  PRSRE     C1*RDNG+C2→PRSRE@B19
          :
ZERØ     CØN  D,0
C1       CØN  D,2B2
C2       CØN  D,43B19
PRSRE    BSS  1
RDNG     BSS  1
          :
```

Fixed-Point (Double-Word Precision)

Fixed-point double-precision arithmetic uses A and Q together to form a single word 47 bits long. This allows expression of numbers ranging between plus and minus $70,368,744,177,663_{10}$. The format for representing a double-word fixed-point number is as follows:



The double word add and subtract instructions are explained below. There are no double-word fixed-point multiply and divide instructions in PAL.

DAD Y

DOUBLE-LENGTH ADD — The contents of core locations Y and Y+1 are algebraically added to A and Q together. The result is placed in A and Q. The C(Y) and C(Y+1) are not changed.

DSU Y

DOUBLE-LENGTH SUBTRACT — The contents of core locations Y and Y+1 are algebraically subtracted from A and Q. The results are stored in A and Q. The C(Y) and C(Y+1) are not changed.

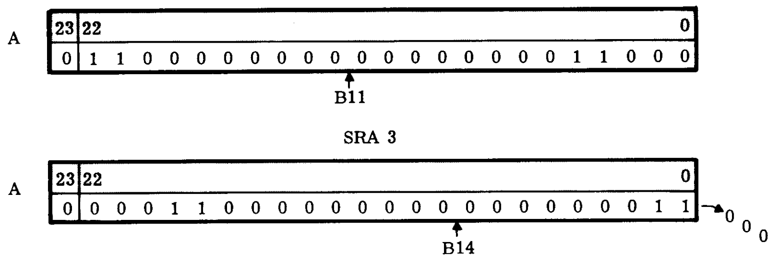
Scale Modification By Arithmetic Shifts

Since fixed-point arithmetic requires the continual juggling of scale factors, it is helpful to have a means of shifting data one way or the other within a register. The following instructions shift data in the A register, or A and Q together.

SRA K

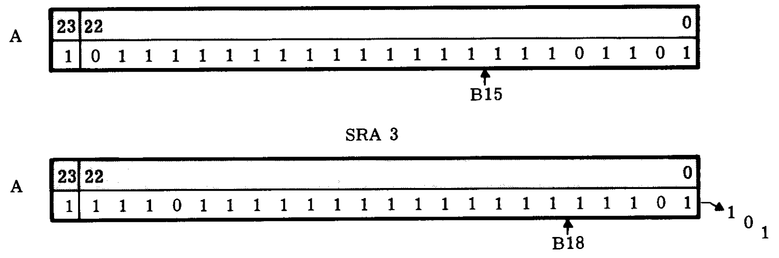
SINGLE RIGHT ARITHMETIC — The contents of the A register, with the exception of the sign bit are shifted right K places. If the sign is positive (0), zeroes are inserted into the vacated positions. If the sign is negative (1), ones are inserted into the vacated positions. Bits shifted out of the right end are lost.

Example:



Sign (0) is propagated to the right. Bits are lost out of the right side of A. There is no overflow indication.

Example:

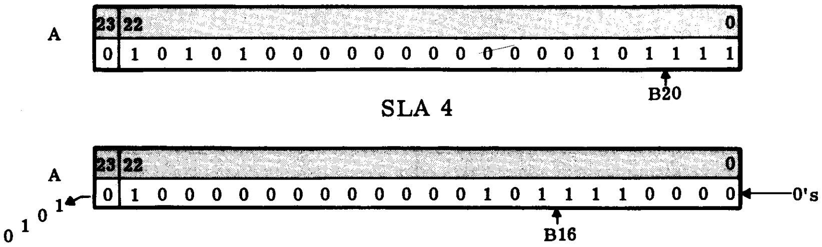


Sign (1) is propagated to the right. Notice that the propagation of ones when shifting a negative number does not affect its value.

SLA K

SINGLE LEFT ARITHMETIC — The contents of the A register are shifted left K places. Vacated positions are filled with zeroes. If the original sign of A was positive (0) and a one is shifted into bit 23, OVRF will be set. If the original sign of A was negative (1) and a zero is shifted into bit 23, OVRF will also be set.

Example:

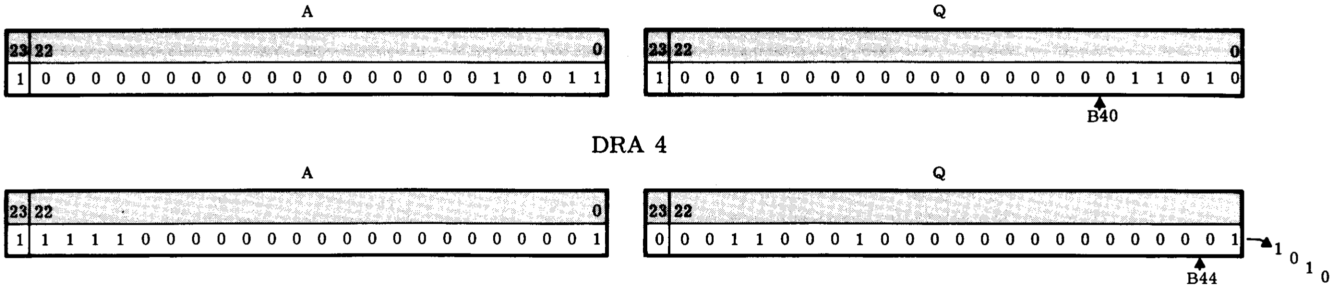


Bits are lost out of the left side of A. OVRF is set, since original sign was positive (0) and ones were shifted through bit 23.

DRA K

DOUBLE RIGHT ARITHMETIC – The contents of A and Q with the exception of their sign bits, are shifted K places to the right. Bits shifted out of A are loaded into Q beginning at bit 22. Bits shifted out of Q are lost. The sign of A is propagated into the vacated positions. The sign of Q is set to zero. A single DRA instruction may shift up to 31 positions.

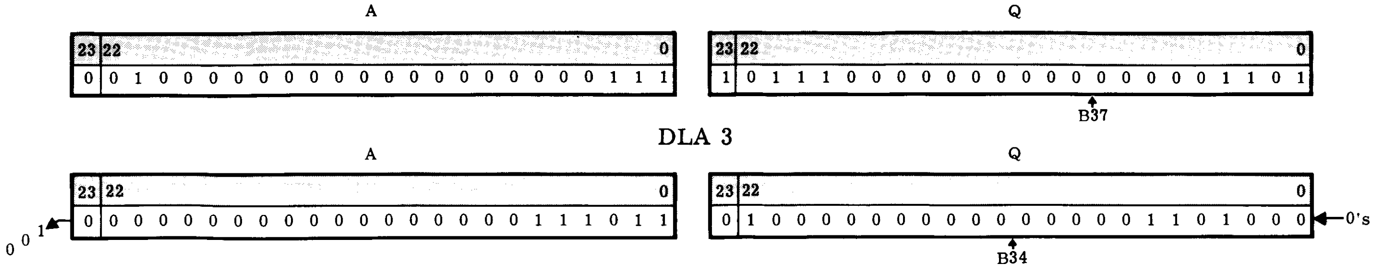
Example:



DLA K

DOUBLE LEFT ARITHMETIC — The contents of A and Q with the exception of the sign of Q are shifted left K places. Bits shifted out of the left end of Q go into the right end of A. Bits shifted out of A are lost. Zeroes are shifted into the right end of Q. If the sign of A was positive (0) and ones are shifted into that position, OVRF will be set. If the sign of A was negative (1) and zeroes are shifted into that position, the OVRF will be set. Bit 23 of Q is always reset to zero. A single DLA instruction may shift up to 31 positions.

Example:



Since a one shifted through bit 23 of A and bit 23 was originally zero, OVRF is set.

Shifting to adjust scale factors is the most common use for arithmetic shift instructions — either to make scale factors equal before an add or subtract instruction, or to prevent possible overflow while obtaining maximum precision in calculations.

Example: Y = X + ZU Where constants are defined as shown:

```
              :
             LDA  ZERØ
             LDQ  Z           Z——→Q@B6
             MPY  U           U*Z——→AandQ@B12
             STA  Y           RESULT——→Y@B12
             LDA  X           X——→A@B11
             SRA  1           B11——→B12
             ADD  Y           X+Y——→A@B12
             STA  Y           RESULT——→Y@B12
              :
Y            BSS  1
X            CØN  D,1000B11
Z            CØN  D,50B6
U            CØN  D,52B6
ZERO         CØN  D,0
              :
```

Why not SLA 1 to B11 instead? Overflow would occur.

Example: Y = C + D Where constants are defined as shown:

```
              :
             LDA  D          D ——→A reg
             SRA  4          B6——→B10
             ADD  C         C+D——→A reg@B11
             STA  Y         RESULT——→Y@B11
              :
D            CØN  D,1000B10 \ CONSTANTS
C            CØN  D,50B6    /
Y            BSS  1           VARIABLE STORAGE
              :
```

Other Fixed-Point Instructions

Besides the basic add, subtract, multiply, and divide instructions used in fixed-point arithmetic, the following instructions provide an added degree of computational flexibility and efficiency.

AKA K

ADD K TO A – The integer K is added to the contents of the A register. OVRF will be set if overflow occurs.

SKA K

SUBTRACT K FROM A — The integer K is subtracted from the contents of the A register. OVRF will be set if overflow occurs.

LDK K

LOAD A WITH K — The integer K replaces the contents of the A register.

Example: Y = 5 + 7 - 2

```
        LDK  5        5 ──► A  reg
        AKA  7        5+7=12 ──► A  reg
        SKA  2        12-2=10 ──► A  reg
```

K is always loaded, added, or subtracted at B23.

General Utility Instructions

The following instruction is particularly prior to performing a fixed-point multiply. It shifts the contents of the A register to Q, thus setting up registers for a multiply instruction, and then loads zeroes into A.

MAQ

MOVE A TO Q — The contents of the A register replace the contents of Q. Zeroes are loaded into A.

Example: Y = 10*PICØN

```
         :
        LDK  10          10 ──► A  reg
        MAQ              C(A reg) ──► Q, ZEROES ──► A  reg
        MPY  PICØN       10* PICØN ──► AandQ @ B28
        STQ  Y           RESULTS ──► Y @ B5
         :
PICØN   CØN  D,3.1416B5
Y       BSS  1
         :
```

When a program is first written, it is helpful to occasionally insert dummy instructions to make future addition of instructions easier. It is also helpful to have dummy instructions to insert in place of instructions that must be removed. The following instruction is used as a dummy.

NOP

NO OPERATION — No operation is performed, program control moves to the next instruction.

Occasionally, when available core area is scarce, but computer time is plentiful, the following instruction may be useful.

OOM Y,

OPERATE ON MEMORY – Core location Y serves as the A register for the first instruction following OOM. The C(Q) are destroyed.

Example: Add one to C(CNTVAL). Would look like this if coded conventionally . . .

```
        STA  TEMP        SAVE A  reg
        LDA  CNTVAL   \
        ADD  ØNE       } PERFORM  OPERATION
        STA  CNTVAL   /
        LDA  TEMP        RESTORE A  reg
         :
ØNE     CØN  D,1
         :
```

Required core locations - 5 words, execution time - 16 μs or like this, using OOM:

```
         :
         ØØM  CNTVAL
         ADD  ØNE            OPERATION PERFORMED
         :                    WITHOUT DISTURBING A
ØNE      CØN  D,-1
         :
```

Required core locations - 2 words, execution time - 65.2μ

The use of OOM increases execution time but saves core. Consult the Instruction Reference Manual for instructions that may be OOM'ed.

Changing Arithmetic Modes

Fixed-point operations may be performed only with fixed-point numbers. Floating-point operations may be performed only with floating-point numbers. For example, we cannot add fixed point 12.4 to floating point .5 x 10^2. To work with these numbers we must either convert .5 x 10^2 into a fixed-point number and ADD it to 12.4, or convert 12.4 into a floating-point number and FAD it to .5 x 10^2.

To convert numbers from one mode to another use the following instructions.

FIX K

FIX A FLOATING POINT NUMBER — The contents of the A register are converted from a floating-point number to a fixed-point number with a scale factor of K.

FLO K

FLOAT A FIXED POINT NUMBER — The contents of the A register are converted from a fixed-point number with a scale factor of K to a normalized floating-point number.

NOTE: The instructions above describe the operation as it occurs in single-word precision (FMS 1). In double-word precision (FMS 2) these instructions will act upon the double word expressed in A and Q.

Example: Y = D + F

D = fixed point number @ B17

F = floating-point number

Performing the operation in floating point;

```
          :
          LDA  D               D——►A reg
          FLØ  17              D CONVERTED TO FLOATING POINT
          FAD  F               D+F———►A reg
          STA  Y               RESULTS———►Y
          :
D         CØN  D,50.0B17
F         CØN  F,.25E2
Y         BSS  1
          :
```

or, in fixed point;

```
            :
            LDA  F               F——→Areg
            FIX  17              F CONVERTED TO FIXED POINT@B17
            ADD  D               F+D——→Areg
            STA  Y               RESULTS——→Y
            :
D           CØN  D,50.0B17
F           CØN  F,.25E2
Y           BSS  1
            :
```

In conjunction with the FMS instruction, FIX and FLØ may also be used for the conversion of double-word numbers to single-word and vice-versa.

Example: FXNØ = double-word fixed-point number @ B17

FLØNØ = single-word floating-point number that could be represented in fixed point @ B12

Convert: FXNØ to a single-word floating-point number

FLØNØ to a double-word fixed-point number @ B12

```
            :
            DLD  FXNØ            FXNØ——→A, FXNØ+1——→Q
            FMS  1               SHIFT TO SINGLE-WORD PRECISION
            FLØ  17              FLOAT C(Areg)@B17
            STA  FXNØ            RESULTS——→FXNØ
            LDZ                  ZEROES——→Areg
            MAQ                  ZEROES——→Q
            LDA  FLØNØ           FLØNØ ——→Areg
            FMS  2               SHIFT TO DOUBLE-WORD PRECISION
            FIX  12              FIX C(Areg)@B12
            DST  FLØNØ           RESULTS——→FLØNØ and FLØNØ+1
            :
```

LOGICAL OPERATIONS

A major feature of the GE-PAC 4020 computer is the number and variety of its logical operations. The following sections cover the PAL repertoire of bit manipulation, word logical, masking, logical shifting, and bit counting instructions.

Bit Manipulation Instructions

Instructions that manipulate individual bits of a 24-bit word are often useful. With them it is possible to dictate the sequencing of a series of process operations, set bit flags for inter/intra-program communication, or create any desired bit pattern within a word. These instructions are explained below.

SBK K

SET BIT K — Bit K of the word in the A register is set to one. All other bits are unchanged.

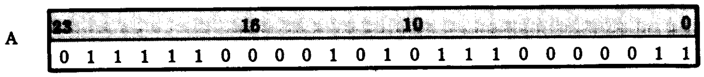
RBK K

RESET BIT K — Bit K of the word in the A register is reset to zero. All other bits are unchanged.

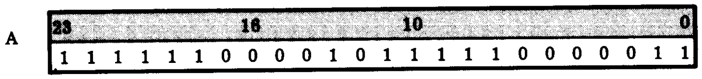
CBK K

CHANGE BIT K – If Bit K was set to one, it is then reset to zero. If it was reset to zero, it is then set to one. All other bits are unchanged.

Examples of SBK, RBK and CBK:



```
SBK 23
RBK 16
CBK 10
```



IBK K

ISOLATE BIT K – Bit K of the word in the A register is unchanged. All other bits are reset to zero.

LBM K

LOAD BIT MASK – Bit K of the word in the A register is reset to zero. All other bits are set to one.

LDO K

LOAD ONE INTO BIT K OF A – Bit K of the A register is set to one. All other bits are reset to zero.

ADO K

ADD ONE TO BIT K OF A – Plus one is algebraically added to the number represented by bits 23-K of the A register. This instruction will not affect OVRF.

Register Manipulation

These instructions set the A register to all zeroes, all ones, or take the one's or the two's complement of its contents.

LDZ

LOAD A WITH ZEROES – All bits of the A register are reset to zero.

LMO

LOAD A WITH MINUS ONE – All bits of the A register are set to one.

CPL

ONE'S COMPLEMENT OF C(A) — Each bit of the A register is inverted. Ones are replaced by zeroes; zeroes by ones.

NEG

NEGATE C(A) — Each bit of the A register is inverted and one is added to bit position zero, thus forming the two's complement (negative value) of the original number.

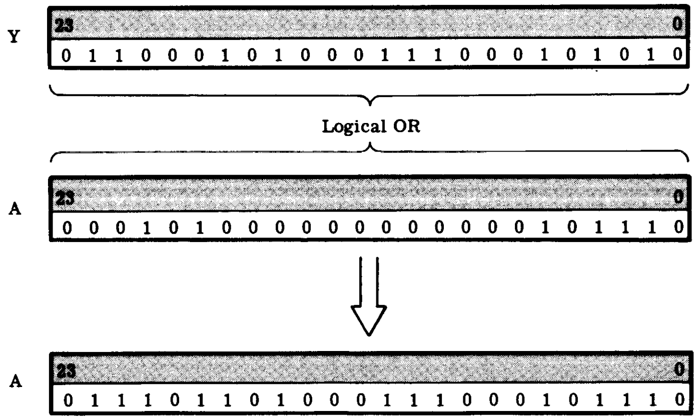
Word Logical Operations

In process control programming it is frequently helpful to look at particular parts of a word that contain information concerning particular areas of your process. Word logical masking techniques make this possible.

ORA Y

OR C(Y) WITH C(A) — Each bit of the contents of core location Y is compared with the corresponding bit of the A register. If either or both contain a one, that bit of A is set to one. Otherwise, it is reset to zero. The C(Y) are unchanged.

Example:



Y: 0 1 1 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0

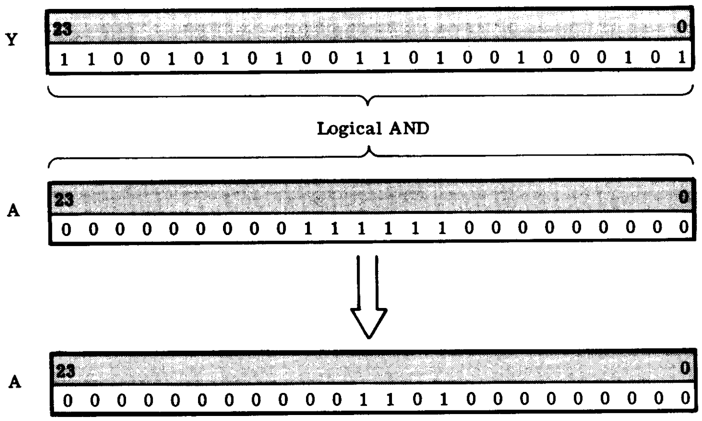
A: 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0

A: 0 1 1 1 0 1 1 0 1 0 0 0 1 1 1 0 0 0 1 0 1 1 1 0

ANA Y

AND C(Y) WITH C(A) — The corresponding bits of core location Y and the contents of the A register are compared. If corresponding bits in A and Y are ones, that bit of the A register is set. If either or both of the corresponding bits are reset to zero, that bit of the A register is reset. The C(Y) are unchanged.

Example:

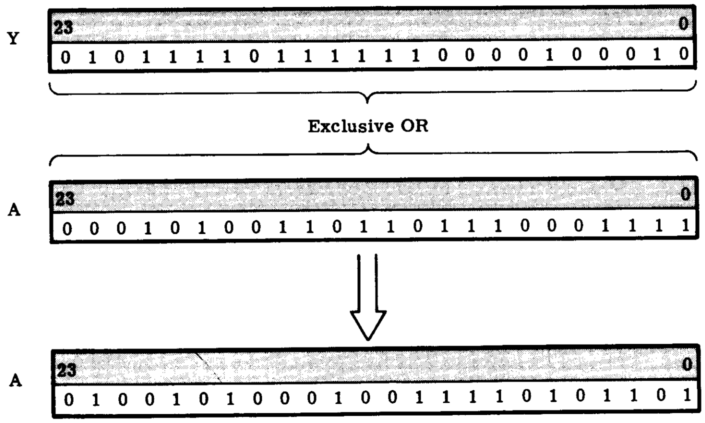


Notice that by setting certain bits in A it is possible to determine which segment of Y will be copied into A.

ERA Y

EXCLUSIVE OR OF C(Y) WITH C(A REG) — If corresponding bits of Y and the A register are alike, a zero is placed in that position in the A register. If they are unlike, that position is set to one. The C(Y) are unchanged.

Example:



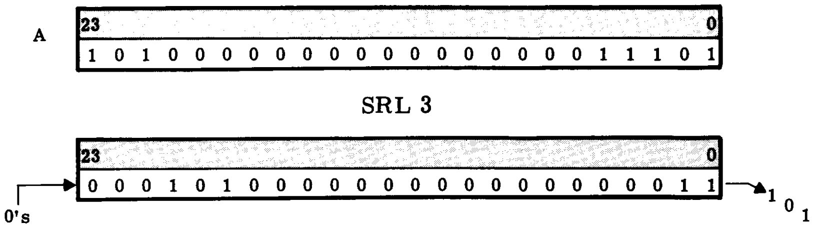
Logical Shifting

Logical shifts differ from arithmetic shifts in that they make no effort to preserve the sign bit or set OVRF when overflow occurs. Logical shift instructions are described below.

SRL K

SINGLE RIGHT LOGICAL — The contents of the A register are shifted K places to the right. Zeroes are shifted into A; the bits shifted out are lost. A single SRL instruction may shift up to 23 places.

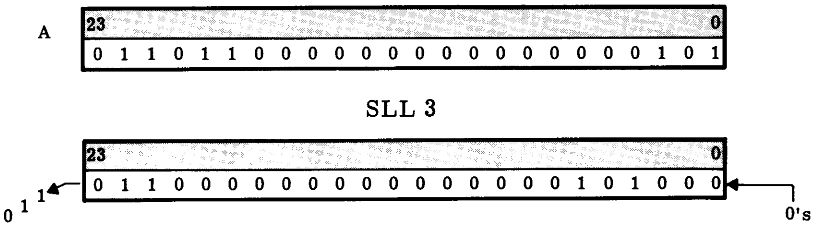
Example:



SLL K

SINGLE LEFT LOGICAL — The contents of the A register are shifted K places to the left. Zeroes are shifted into A; the bits shifted out are lost. A single SLL instruction may shift up to 23 places.

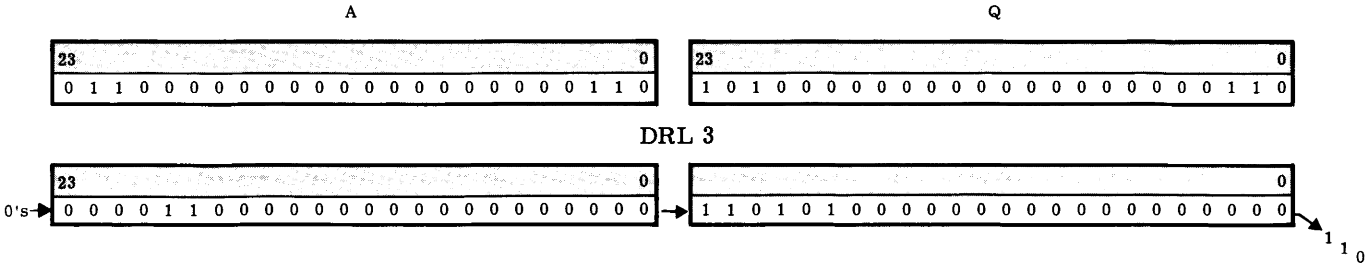
Example:



DRL K

DOUBLE RIGHT LOGICAL — The contents of A and Q together are shifted K places to the right. The bits shifted out of A are loaded into Q. The bits shifted out of Q are lost. Zeroes are loaded into A. A single DRL instruction may shift up to 31 places.

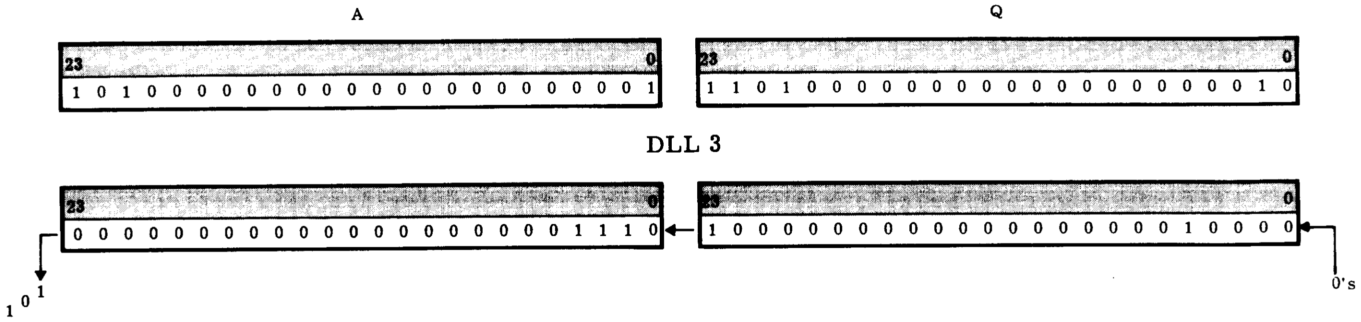
Example:



DLL K

DOUBLE LEFT LOGICAL — The contents of A and Q are shifted K places to the left. The bits shifted out of Q are loaded into A. The bits shifted out of A are lost. Zeroes are loaded into Q. A single DLL shift may shift up to 31 places.

Example:



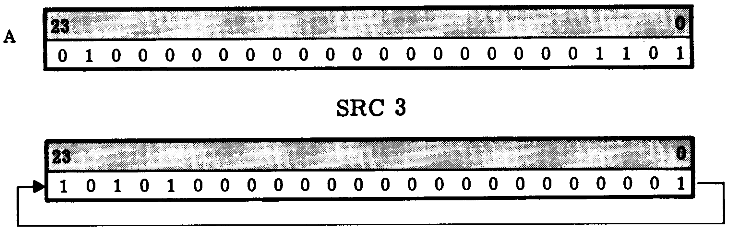
Circular Shifting

Circular shifts allow repositioning data without losing any of it. They are frequently used for packing input data and unpacking data for output. Circular shift instructions are described below.

SRC K

SHIFT RIGHT CIRCULAR — The contents of the A register are shifted right K places. The bits leaving the right end of the register are loaded back into the left end.

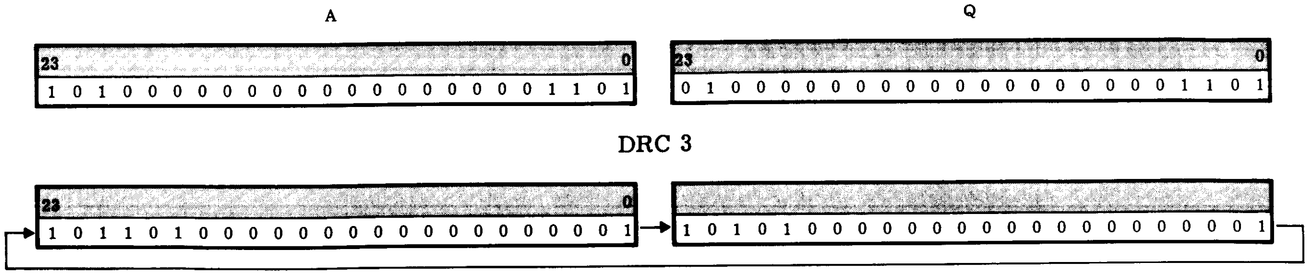
Example:



DRC K

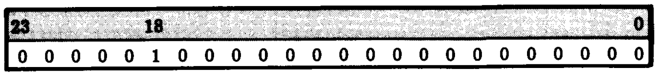
DOUBLE RIGHT CIRCULAR — The contents of A and Q are shifted together K places to the right. Bits shifted out of A are loaded into Q. Those shifted out of Q are loaded into the left end of A. A single DRC instruction may shift up to 31 places.

Example:



Bit Counting

Process alarm conditions are often represented by the condition of a particular bit in a data word. The diagram below shows a typical alarm word. Each bit in the word that is set to one represents the alarm condition of a particular device. If bit 18 is set when a certain temperature is out of limits, we may use the following bit counting instructions to isolate that alarm condition (see example on page 42).



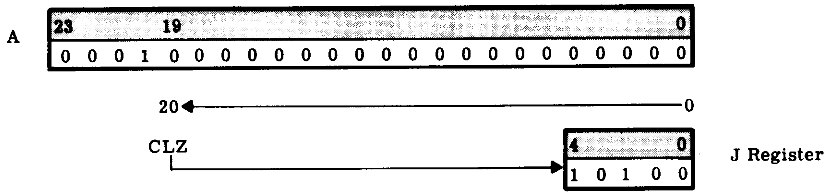
Alarm Data Word

Bit counting instructions accumulate their total count in a five-bit register called the J register. After executing a counting instruction, the J register is interrogated to find the total count.

CLZ

COUNT LEAST SIGNIFICANT ZEROES – The number of least significant zeroes in the A register is placed in the J register.

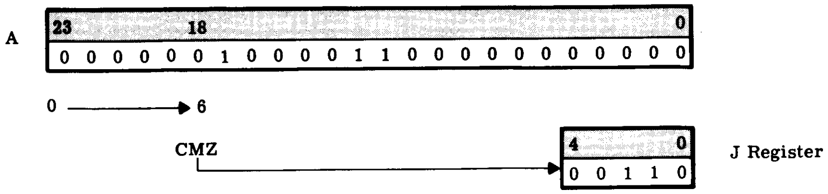
Example:



CMZ

COUNT MOST SIGNIFICANT ZEROES – The number of most significant zeroes in the A register is placed in the J register.

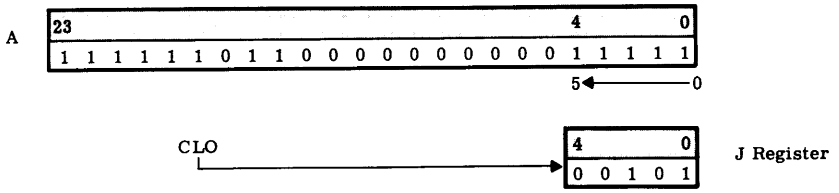
Example:



CLO

COUNT LEAST SIGNIFICANT ONES – The number of least significant ones in the A register is placed in the J register.

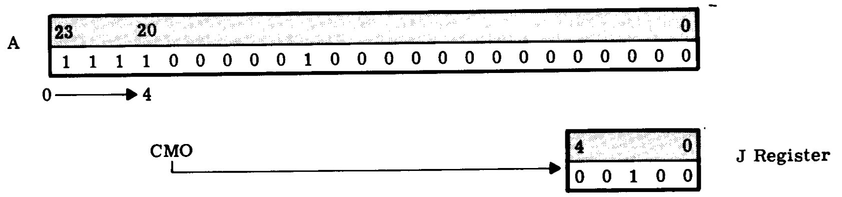
Example:



CMO

COUNT MOST SIGNIFICANT ONES — The number of most significant ones in the A register is placed in the J register.

Example:



LXC X

LOAD X REGISTER WITH C(J) — The contents of the J register are placed in the rightmost five bits of the index location specified. The other 19 bits of that index location are reset to zero. An LXC should immediately follow any count instruction, to avoid loss of the count.

Example:

```
 ⋮
LDA  Y
CMZ          COUNTS ——→ J Register
LXC  3       C(J) ——→ index location 3
 ⋮
```

TEST INSTRUCTIONS

Every process computer program does a great deal of testing — testing for alarm conditions, testing to determine the size of numbers, testing to determine the number of times an event has occurred or a program loop has been completed, testing to determine what a previous program did or a future program should do, and testing for a myriad of other reasons. To fill these needs, the GE-PAC 4020 computer has an unparalleled group of word and bit test instructions.

The result of a test instruction may be thought of as either a true, or a false condition. A test flip-flop named TSTF records the result of a test. If the result of a test is true, TSTF is set to one; if false TSTF is reset to zero. There are also tests that affect TSTF for one condition but leave it unchanged for the other. These and other test instructions are explained below.

Setting and Resetting the TSTF Directly

These instructions are used to set TSTF to a known condition.

SET

SET TSTF — The test flip-flop, TSTF, is set.

RST

RESET TSTF — The test flip-flop, TSTF, is reset.

Word Tests

These tests operate on the entire A register.

TZE

TEST A EQUAL ZERO — TSTF is set if all bits in the A register are zero. TSTF is reset if any bit is a one.

TNZ

TEST A NOT ZERO — TSTF is set if any bit in the A register is a one. TSTF is reset if all bits are zero.

RNZ

RESET IF A NOT ZERO — TSTF is reset if any bit in the A register is a one. TSTF is unchanged if all bits are zero.

SNZ

SET IF A NOT ZERO — TSTF is set if any bit in the A register is a one. TSTF is unchanged if all bits are zero.

TNM

TEST A NOT MINUS ONE — TSTF is set if any bit in the A register is a zero. TSTF is reset if all bits in A register are ones (minus one).

TZC

TEST A ZERO AND COMPLEMENT — TSTF is set if all bits in the A register are zeroes. If any bit is a one TSTF is reset and the contents of the A register are replaced by its one's complement (all bits are inverted).

TSC K

SHIFT RIGHT CIRCULAR AND TEST FOR K ZEROES — The contents of the A register are shifted right circular K places. Bits shifted out of the right end of A are loaded back into the left end. If all K bits shifted out of A are zeroes TSTF is set. If any of the K bits is a one, TSTF is reset.

Bit Tests

The following tests are performed on a specific bit in the A register.

TEV K

TEST BIT K EVEN – TSTF is set if bit K of the A register is a zero. TSTF is reset if it is a one.

TOD K

TEST BIT K ODD – TSTF is set if bit K of the A register is a one. TSTF is reset if it is a zero.

SEV K

SET TSTF IF BIT K EVEN – TSTF is set if bit K of the A register is a zero. If it is a one, TSTF remains unchanged.

REV K

RESET TSTF IF BIT K EVEN – TSTF is reset if bit K of the A register is a zero. If it is a one, TSTF remains unchanged.

SOD K

SET TSTF IF BIT K ODD – TSTF is set if bit K of the A register is a one. If it is a zero, TSTF remains unchanged.

ROD K

RESET TSTF IF BIT K ODD – TSTF is reset if bit K of the A register is a one. If it is a zero, TSTF remains unchanged.

TES K

TEST EVEN AND SET BIT K – If bit K of the A register is a one, TSTF is reset. If bit K is a zero TSTF is set and bit K is set to one.

TER K

TEST EVEN AND RESET BIT K – If bit K of the A register is a zero, TSTF is set. If it is a one, TSTF is reset and bit K is changed to a zero.

TOS K

TEST ODD AND SET BIT K – If bit K of the A register is a zero, TSTF is reset and bit K is changed to a one. If it is a one, TSTF is set.

TOR K

TEST ODD AND RESET BIT K – If bit K of the A register is a one, TSTF is set and bit K is changed to a zero. If it is a zero, TSTF is reset.

BRANCH INSTRUCTIONS

After executing a test instruction it is usually necessary to do one thing if TSTF is set, and another if it is reset. The conditional branch instructions described below will jump to another part of a program for a given condition of TSTF. The unconditional branch instruction described always transfers program control when executed. Program control is the logic that determines which instruction will be executed next.

BTS Y

BRANCH IF TSTF SET — If TSTF is set, program control will branch and begin executing instructions starting at location Y. If reset, it will execute the instruction immediately following the BTS. TSTF is unchanged.

BTR Y

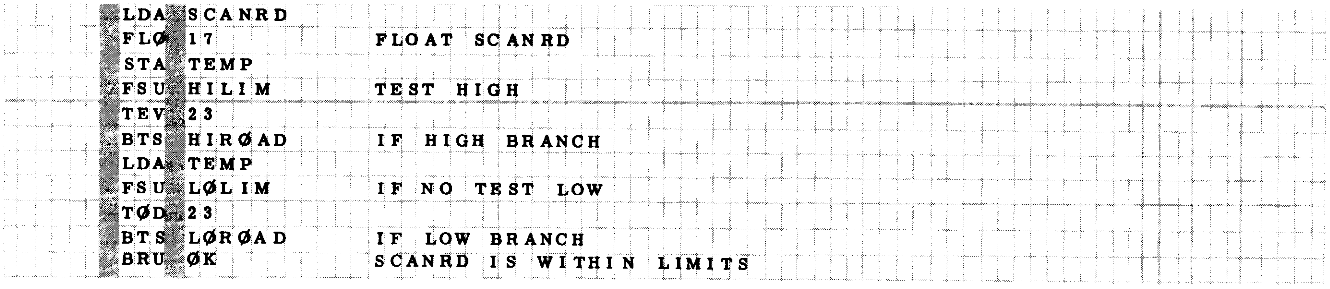
BRANCH IF TSTF RESET — If TSTF is reset, program control will branch and begin executing instructions starting at location Y. If set, it will execute the instruction immediately following the BTR. TSTF is unchanged.

BRU Y

UNCONDITIONAL BRANCH — Program control will always branch and being executing instructions starting at location Y.

Examples:

It is necessary to know if an analog scanner reading is within limits. The high limit is contained in core location HILIM, the low limit is at LOLIM, the scanner reading is at SCANRD. These limits are in floating point. The reading is in fixed point at B17. If the reading is high branch to HIROAD, if low branch to LOROAD, if within limits branch to OKROAD.



```
LDA SCANRD
FLØ 17           FLOAT SCANRD
STA TEMP
FSU HILIM        TEST HIGH
TEV 23
BTS HIRØAD       IF HIGH BRANCH
LDA TEMP
FSU LØLIM        IF NO TEST LOW
TØD 23
BTS LØRØAD       IF LOW BRANCH
BRU ØK           SCANRD IS WITHIN LIMITS
```

Assume that one step in a startup procedure consists of determining that one and only one of two pumps is on, that its valve is open, and a main valve is open. The condition of these motors and valves are held in five bits of a word at location GROUP 1. If a bit is set to one the on, or open condition, exists. Bit assignments are shown below.

Bit 1 — Motor for pump #1
Bit 2 — Valve for pump #1
Bit 3 — Motor for pump #2
Bit 4 — Valve for pump #2
Bit 5 — Main valve

Write a routine to check that the system is ready for startup. If it isn't ready, branch to ALARM.

```
           :
        LDA  GRØUP1
        TØD  BITA           PUMP #1 MOTOR ON?
        BTR  *+5            IF NO, BRANCH
        REV  BITB              If any of these
        RØD  BITC              conditions are not met TSTF
        RØD  BITD              is reset, indicating alarm
        BRU  *+5            BRANCH TO TEST E
        SET                 SET TSTF
        RØD  BITB              If any of these conditions are not
        REV  BITC              met, TSTF is reset
        REV  BITD              indicating alarm
        REV  BITE
        BTS  ØK             If TSTF still set — everything OK
        BRU  ALARM          If not, an alarm exists
           :
           :
BITA    EQL  1
BITB    EQL  2
BITC    EQL  3
BITD    EQL  4
BITE    EQL  5
           :
```

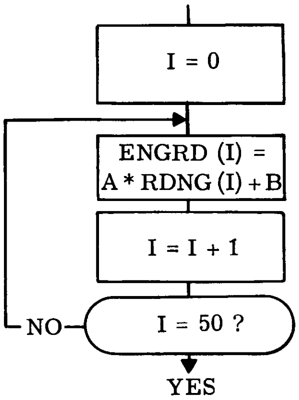
A scan and limit check program finds alarms and indicates them in a word at location ALM. Each of the 24 bits indicate a unique alarm condition when set. Write the coding to isolate any alarm conditions.

```
           :
NEXT    LDA  ALM            ALM ——→ Areg
        CLZ                 COUNTS ——→ Jreg
        LXC  3              C(J) ——→ 3
        TXH  24,3           ANY ALARMS ?
        BTS  NØALM          NO
        RBK  0,3            YES, Reset bit
        STA  ALM               and service
           :                   alarm condition.
```

After an alarm is serviced the program may go back to NEXT and determine if more alarms exist.

LOOPING

If a table of fifty values must be converted to engineering units with the equation, ENGRD = A*RDNG + B, we certainly wouldn't want to write or provide storage for fifty programs to solve the equation. Looping permits writing the program once and cycling through it fifty times. In flow chart form the procedure would look like this:



Where A and B are constants, ENGRD and RDNG are fifty word tables, and I is an index to indicate which reading is being converted.

The following additional instructions are used in writing loops in PAL.

LXK K, X

LOAD INDEX LOCATION X WITH K — The constant K replaces the contents of the index location specified.

INX K, X

INCREMENT C(X) BY K — The constant K is added to the contents of the index location specified.

TXH K, X

TEST C(X) HIGH OR EQUAL TO K — If the contents of bits zero through 13 of index location X are equal to or greater than K TSTF is set. If not, it is reset. The C(X) are not changed. Note: K is always represented in its two's complement (negative) form within the computer.

DMT Y

DECREMENT MEMORY AND TEST — One is subtracted from the contents of memory location Y. If the original value was zero, TSTF is reset. If the original value was non-zero, TSTF is set. When a DMT is the first instruction following an automatic program interrupt, TSTF is not affected.

There are two methods of looping in PAL; incrementing, which begins with the first of a group of numbers and loops through to the last; and decrementing, which begins with the last number and works back to the first.

Example:

Fifty scanner readings in a table beginning at core location 17000$_8$ must be converted to engineering units by the equation(80 * RDNG)/4000 and stored in a table beginning at core location 17500$_8$. The scanner readings are in fixed-point form at B17. Perform the calculations in floating point.

/17000	---
/17001	---
.	.
.	.
.	.
.	.
.	.
/17061	---

RDNG Table

/17500	---
/17501	---
.	.
.	.
.	.
.	.
.	.
/17561	---

ENGRD Table

By incrementing:

```
        FMS   1
        LXK   0, 5            Initialize Loop
        LDA   RDNG, 5
        FLØ   17              Convert and store
        FMP   A
        FDV   B
        STA   ENGRD, 5
        INX   1, 5            Count readings processed
        TXH   50, 5           Have all Readings been Processed?
        BTR   * - 7           If no, go back; if yes, continue
          :
A       CØN   F, 80           Constants
B       CØN   F, 4000
RDNG    EQL   /17000
ENGRD   EQL   /17500
          :
```

By decrementing

```
          :
        FMS   1
        LXK   49. 5           Initialize Loop
        LDA   RDNG, 5
        FLØ   17
        FMP   A               Convert & Store
        FDV   B
        STA   ENGRD, 5
        DMT   5               Decrement; If C (5) = -1 TSTF is Reset
        BTS   * - 6           If TSTF Set go back, If Reset continue
          :
A       CØN   F, 80           Constants
B       CØN   F, 4000
RDNG    EQL   /17000
ENGRD   EQL   /17500
          :
```

CHAPTER 8

AUTOMATIC PROGRAM INTERRUPTS

Within the GE-PAC 4020 computer there are a group of flip-flops called automatic program interrupts, or API's. There may be up to 128 API's in a system. These make the GE-PAC 4020 system responsive to process disturbances, permit timekeeping, and monitor the operation of peripherals.

There are two types of interrupts; inhibitable, and non-inhibitable. Non-inhibitable interrupts usually indicate relatively high-priority demands. Lower-priority demands are indicated by the occurrence of inhibitable interrupts.

These interrupts set a flag for RTMOS, causing it to perform whatever function that interrupt requires. In the case of non-inhibitable interrupts this is usually the execution of a single instruction, such as a DMT. After the interrupt has been serviced RTMOS usually continues running the program that was interrupted.

Inhibitable interrupts frequently require more extensive service such as the running of another program. In this case, RTMOS may run the program required by the interrupt and finish the interrupted program later on. Usually this is acceptable, since information critical to the interrupted program is saved and the program will be completed a few seconds later. However, there are times, in critical programs, or in critical steps of a program, when interruption is undesirable. In these cases the servicing of inhibitable API's may be regulated by manipulating the Permit Automatic Interrupt flip-flop, called PAIF.

The PAIF may be set to a one or reset to zero. If reset, inhibitable interrupts will not be serviced until PAIF is set.

Programs may permit or inhibit inhibitable API's with the following two instructions.

PAI

PERMIT AUTOMATIC INTERRUPT — PAIF is set to one, permitting interruption by either an inhibitable or non-inhibitable interrupt. Programs are normally run with interrupts permitted.

IAI

INHIBIT AUTOMATIC INTERRUPTS — PAIF is reset to zero, permitting interruption only by non-inhibitable interrupts. Inhibitable interrupts will be recorded, but not serviced until the PAIF is set to one.

There is also an instruction, IAI2, that will inhibit both inhibitable and non-inhibitable interrupts. It is used in RTMOS I/O routines.

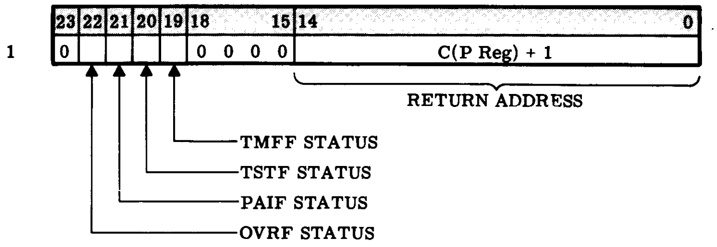
SUBROUTINES

Various process control programs require that the same function be performed several times at different points within a program. In such cases repetitive coding may be avoided by writing the function as a subroutine for the program. Besides serving one program, a subroutine may also be shared by several different programs.

The following instructions are used in writing subroutines:

SPB Y

SAVE PLACE AND BRANCH — Inhibit interrupts and branch to a subroutine located at Y. Before branching to the subroutine, the following information is stored in index location one.



OVRF STATUS — If OVRF is set when the branch occurs a one is placed in bit 22.

PAIF STATUS — If automatic program interrupts are permitted while the main program is running, PAIF is set to one. The condition of this flip-flop at the time the branch occurs is recorded in bit 21.

TSTF STATUS — If TSTF is set when the branch occurs a one is placed in bit 20.

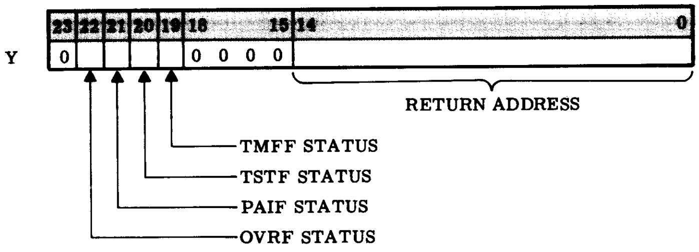
TMFF STATUS - TMFF is the name of the Trapping Mode flip-flop. If a program is running under Quadritect memory protection, TMFF will be set to one; otherwise it is reset to zero.

RETURN ADDRESS — The return address is usually the contents of the P register plus one. If an SPB occurs as the result of an automatic program interrupt one is not added to the C(P).

The SPB instruction always inhibits API's by resetting PAIF before the branch takes place.

LPR Y

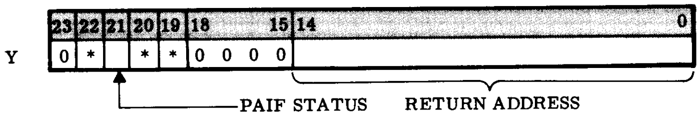
LOAD PLACE AND RESTORE — This instruction is normally used to return to the main program from a subroutine. Usually Y refers to the address of the word built when the SPB instruction was executed.



Program control will go to the address specified in bits 0 - 14 and restore the various flip-flops fo the status indicated. The return address and the status of the flip-flops may be modified within the subroutine.

LDP Y

LOAD PLACE — This is an alternate to LDR. The conditions of OVRF, TMFF and TSTF are not restored. Program control will go to the address specified and set PAIF as specified in the word at location Y. Usually this word is the same one built when the SPB instruction was executed.



*not used

XEC Y

EXECUTE — The instruction at core location Y is executed. Program control does not change unless the XEC'd instruction is a branch.

General procedure for writing PAL subroutines:

a. Locate necessary input variables where the subroutine expects to find them.

b. SPB to subroutine.

c. Inside the subroutine immediately:

   1. Save index location 1 which contains the return address.

   2. Save the contents of A, Q, and any index locations used in the subroutine which contain information critical to the main program.

d. Write the subroutine, bringing in the input variables as required.

e. Locate output variables where the main program expects to find them.

f. Restore all the registers saved in step c. 2.

g. Return to main program via LPR or LDP instruction.

Example:

Since a conversion equation, VALVRD = A*Y*Y + B must be solved several times at different places in a program it is desirable to write it as a subroutine. The subroutine will look for the input variable, Y, in the A Register. It will also return the result in the A register. All data are in floating point form.

```
            :
           LDA   Y              Position Input Variable     \  SEGMENT OF
           SPB   CØNVR          Branch                       } MAIN
           STA   VALVRD   <---  Return made here            /  PROGRAM
            :
CØNVR      STX   TEMP,1         Save Return Address         \
           STA   TEMP+1                                      |
           FMS   1           \                               |
           FMP   TEMP+1       } Calculate                    | SUBROUTINE
           FMP   A            |                              }
           FAD   B           /                               |
           LPR   TEMP           Return                       |
A          CØN   F,5.3                                       |
B          CØN   F,11.2                                      |
TEMP       BSS   2                                          /
            :
```

NOTE: Saving and restoring the contents of Q and index locations 3-7 was not necessary since they were not used.

CIRCULAR LISTS

Circular lists are used for output queues and for transferring data between programs.

Shown below is a diagram of a typical circular list.

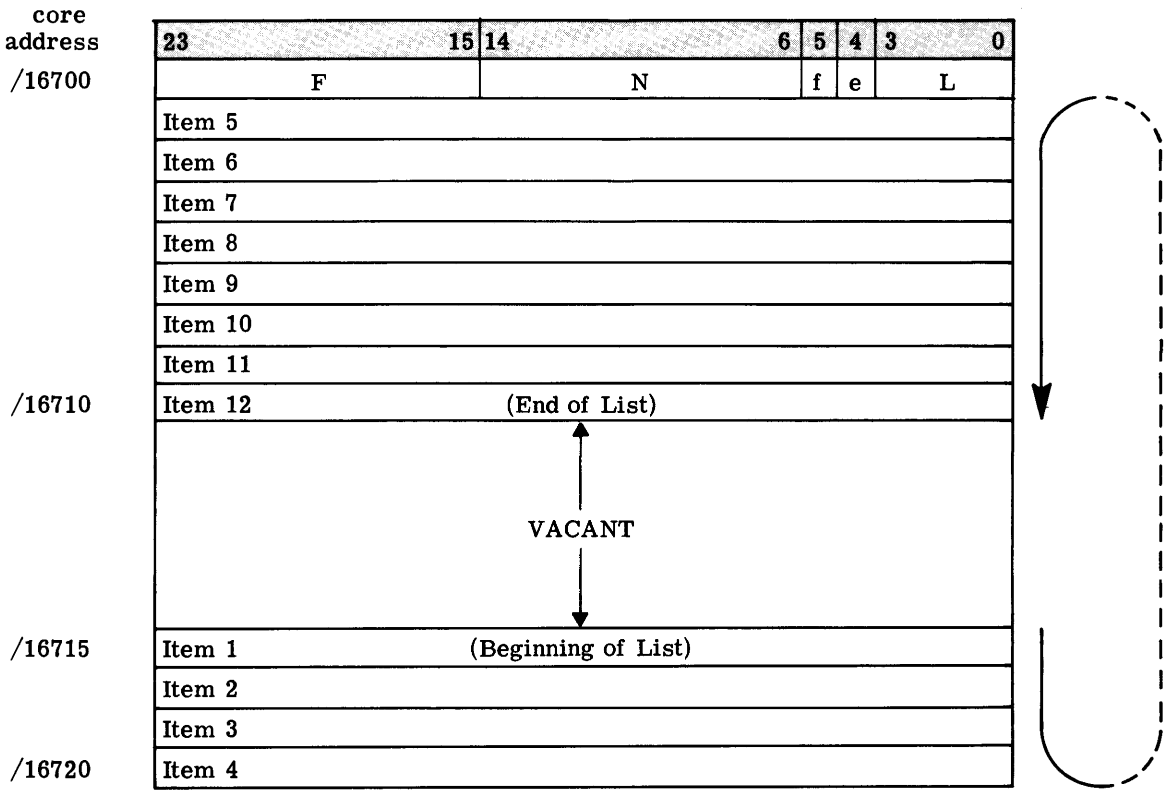


Figure 7. Circular List

The first word of a circular list is the list control word. It specifies the length of the list, the number of items currently in it, the address of the next beginning item, and whether or not the list is full or empty. The description below tells how this information is stored within the word.

L — The length of the list is 2^L. L may vary from one through eight. Therefore, the list may vary in length from two to 256 words. In the above example L is four.

F — This field specifies the location of the next beginning item in the list relative to the first word following the list control word. In the above example F is /13.

N — This field contains the number of items in the list. If the list is either full or empty N is zero. In the above example N is /14.

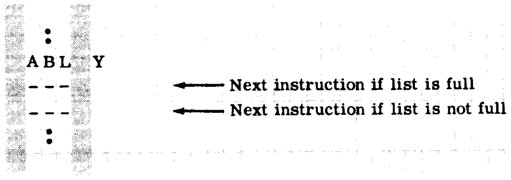
e — This bit is set to one if the list is empty; zero if it is not.

f — This bit is set to one if the list is full; zero if it is not.

The following instructions are used with circular lists.

ABL Y

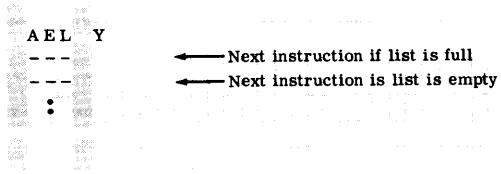
APPEND ITEM TO BEGINNING OF LIST - The list control word at core location Y is checked. If the list is not full (f=0) the contents of the A register are appended to the beginning of the list (Figure 8), the list control word is updated, and program control advances to the second sequential location. The C(A) becomes the new beginning item of the list. If the list is full the instruction is ignored and program control advances to the first sequential location.


:
ABL Y

:
Next instruction if list is full
Next instruction if list is not full

AEL Y

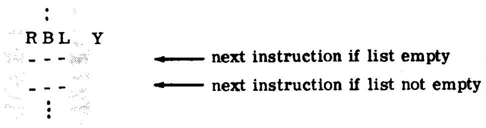
APPEND ITEM TO END OF LIST - The list control word at core location Y is checked. If the list is not full (f=0) the contents of the A Register are appended to the end of the list (Figure 8), the list control word is updated, and program control advances to the second sequential location. The C(A) becomes the new ending item of the list. If the list is full the instruction is ignored and program control advances to the first sequential location.


AEL Y

:
Next instruction if list is full
Next instruction is list is empty

RBL Y

REMOVE BEGINNING ITEM FROM LIST — The list control word at core location Y is checked. If the list is not empty (e=0) the beginning item of the list is extracted (Figure 8) and replaces the contents of the A register. The list control word is then updated, and program control advances to the second sequential location. If the list is empty the instruction is ignored and program control advances to the first sequential location.


:
RBL Y

:
next instruction if list empty
next instruction if list not empty

REL Y

REMOVE ENDING ITEM FROM LIST – The list control word at core location Y is checked. If the list is not empty (e=0) the ending item of the list is extracted (Figure 8) and replaces the contents of the A register. The list control word is then updated, and program control advances to the second sequential location. If the list is empty the instruction is ignored and program control advances to the first sequential location.

```
  :
 REL  Y
 ---        ←—— next instruction if list empty
 ---        ←—— next instruction if list not empty
  :
```

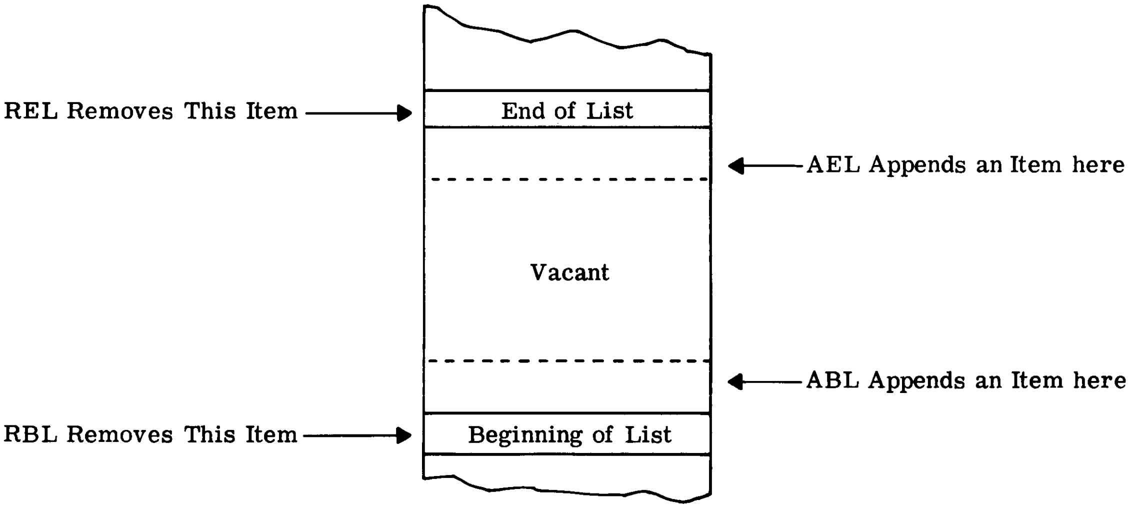


Figure 8. List Instructions

Example:

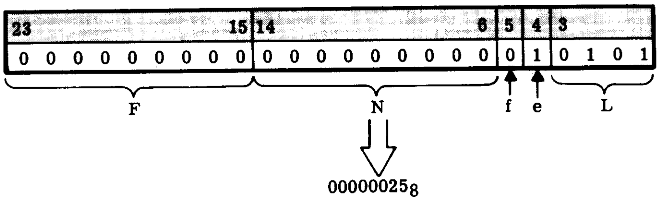
LISTA is the address of a sixty-four word list that is full. It is necessary to transfer its contents to two, empty, thirty-two word lists at locations LISTB and LISTC.

```
          :
NEXWRD   RBL   LISTA       Beginning item——→A
         BRU   EXIT        Branch out when transfer complete
         AEL   LISTB       C(A)——→End of LISTB
         BRU   * + 2       When List B full fill LISTC
         BRU   NEXWRD      Fetch next item from LISTA
         AEL   LISTC       C(A)——→End of LISTC
         NØP               Will never be executed
         BRU   NEXWRD      Fetch next item from LISTA
          :
```

BUILDING A LIST — To build a list requires definition of a list control word and reservation of an appropriate area to hold its contents. To build a list control word determine the contents of each field in binary, convert the resulting word to octal, and define it in the program with a CON O pseudo-instruction. Reserve space with a BSS pseudo-instruction.

Example:

Define an empty list thirty-two words long. Develop the list control word;



and then place it in the program with adequate block storage.

```
         :
LISTWD  CØN  Ø,25
        BSS  32
         :
```

When filling the list simply refer to LISTWD. With some further manipulation of the list control word and the addition of constants following it, a list may be partially or entirely filled at the time it is defined.

One of the benefits of the DEF pseudo-instruction described in the Appendix is that it can be used to define an operation which will set up list control words.

PROGRAM INTERCOMMUNICATION

GE-PAC 4020 core memory is divided into two parts; working core and permanent core. Working core is used to run functional programs which are moved into core as required. Permanent core includes the RTMOS and an area called common core.

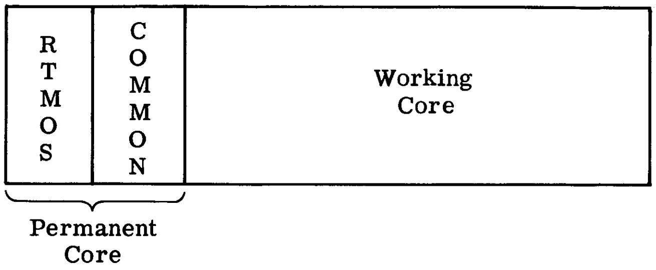


Figure 9. Division of Core Memory

Common core contains data, subroutines, etc. that are frequently used by two or more system programs.

The filling of common core is normally begun during the earlier stages of programming a process control system and continued throughout the project. It contains constants, variables, and the like which are labeled and appended to a common system symbol (equals) table. Each label is written on the coding form with an asterisk (*) in column 7. The assembler program will equate each label with an absolute address, and add that label and its address to its common symbol table. Thereafter, programs that are assembled against the common symbol table may reference any common system symbol by simply calling its name.

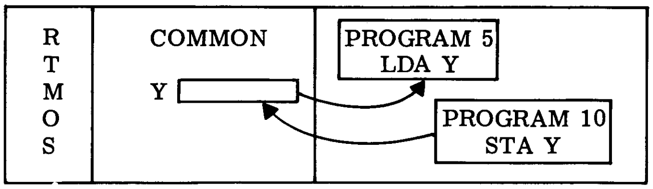


Fig. 10. Communicating variables through common core

Constants, subroutines, circular lists and tables of data that are frequently used by several system programs should be stored in common core.

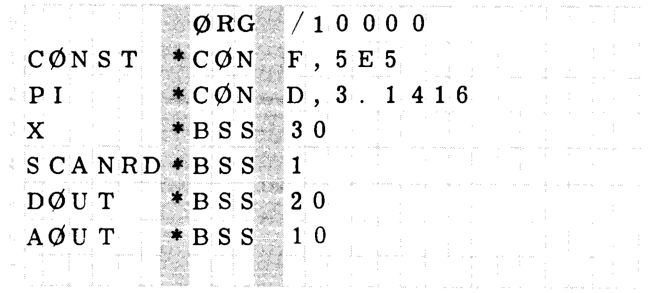
Single variables, tables of variables, and logical data that must be passed between programs should also be stored in common core.

Bulk storage is organized in much the same way as core memory. There is an area for RTMOS, common data storage, and functional program storage. Constants, variables, subroutines, circular lists, and tables that are infrequently used by several programs should be stored in the common area of bulk storage. When a running program requires information resident only in common bulk storage it will request the RTMOS to transfer that information into core.

Example: Build a common core area between /10000 and /10100. The following information is to be loaded into this common area:

CONST	=	5×10^5	(Floating Point)
PI	=	3.1416	(Fixed Point)
X	=	Table of 30 variables to be filled later	
SCANRD	=	To be determined later	
DOUT	=	Table of 20 variables to be filled later	
AOUT	=	Table of 10 variables to be filled later	

The program written to do the job might look like this:



```
       ØRG  /10000
CØNST *CØN  F,5E5
PI    *CØN  D,3.1416
X     *BSS  30
SCANRD*BSS  1
DØUT  *BSS  20
AØUT  *BSS  10
```

The asterisk (*) in column 7 designates those labels as common system symbols which may be used by any system program. An asterisk does not automatically cause the data to be placed in common core, however. To be placed in common core, data must be loaded into that area.

The remaining locations of common in this example are spares. After this program is assembled the resulting information will be fed into the computer, and loaded directly into CORE memory. For backup protection an additional copy of this data may also be placed on bulk. Future programs that use these common system symbols must be assembled with an 'equals table' that specifies the location of each common system symbol.

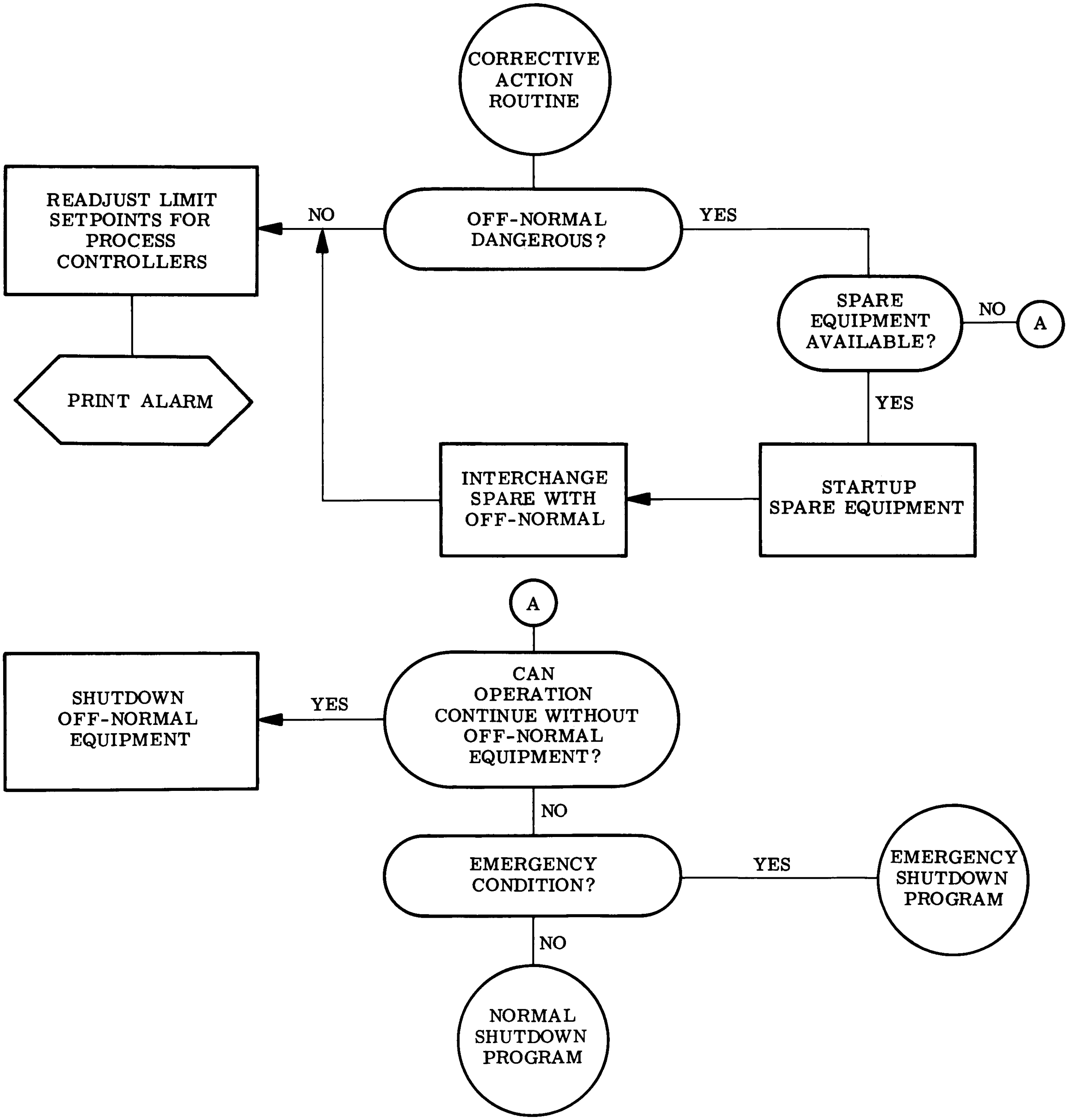
All input to and output from the GE-PAC 4020 computer is normally handled by the Real-Time Multiprogramming Operating System. Complete instructions for its use are in the RTMOS manual.

SPECIAL DISCUSSIONS

FLOW CHARTING

Before writing a program, organize the task using a flow chart. A flow chart logically describes a series of operations to be performed, and permits finding major logic flaws before wasting any programming effort. An example of a flow chart is shown below.

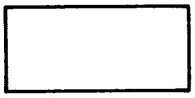
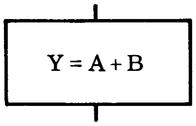
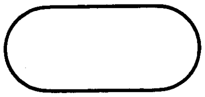
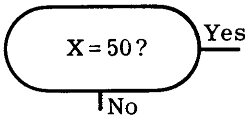
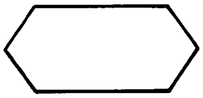
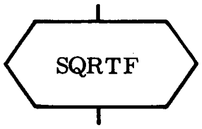
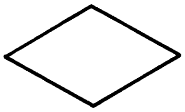
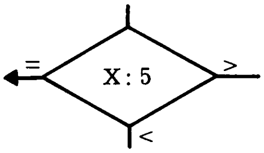
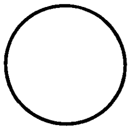
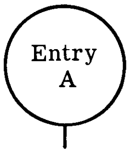
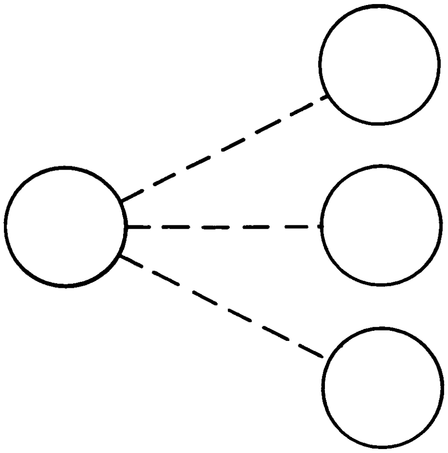
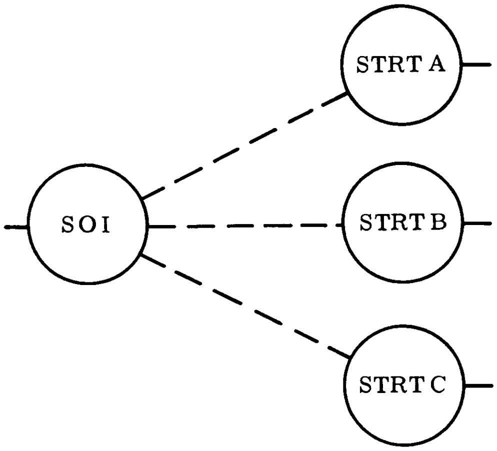
Example: Flow chart for a corrective action routine.


CORRECTIVE ACTION ROUTINE
READJUST LIMIT SETPOINTS FOR PROCESS CONTROLLERS
NO
OFF-NORMAL DANGEROUS?
YES
PRINT ALARM
SPARE EQUIPMENT AVAILABLE?
NO
A
YES
INTERCHANGE SPARE WITH OFF-NORMAL
STARTUP SPARE EQUIPMENT
A
SHUTDOWN OFF-NORMAL EQUIPMENT
YES
CAN OPERATION CONTINUE WITHOUT OFF-NORMAL EQUIPMENT?
NO
EMERGENCY CONDITION?
YES
EMERGENCY SHUTDOWN PROGRAM
NO
NORMAL SHUTDOWN PROGRAM

The general rules governing the use of flow charts are:

1. Always flow chart a program before writing it.
2. Flow charts should contain only a general description of the operations to be performed.

The following symbols are typically used in flowcharting problems for the GE-PAC 4020 process computer system.

Symbol	Meaning	Example
	Operation	
	Decision (2 way)	
	Subroutine	
	Comparison	
	Connector	
	Multiple Switch	

BINARY ARITHMETIC

Probably because he has ten fingers, man has grown accustomed to a decimal (base ten) number system. Similarly, because a computer's logical components such as transistors and relays have two modes of operation, on or off, digital computers can easily use a binary number system. Since it is difficult to remember and recognize binary numbers, a third number system, octal, is used as an aid to interpretation.

The following table shows how to count in binary numbers. Where 1_2 means "one to the base two" or one represented in binary numbers, and

BINARY		DECIMAL EQUIVALENT
0_2	=	0_{10}
1_2	=	1_{10}
10_2	=	2_{10}
11_2	=	3_{10}
100_2	=	4_{10}
101_2	=	5_{10}
110_2	=	6_{10}
111_2	=	7_{10}
1000_2	=	8_{10}
1001_2	=	9_{10}
1010_2	=	10_{10}

1_{10} means "one to the base 10" or one represented in decimal numbers.

The difference between the two is that in the decimal system a carry does not occur until we pass nine, whereas in a binary system a carry occurs when we pass one.

Binary numbers are difficult to read. Since it is alien to think in a system of ones and zeroes, we naturally look at a binary number and try to convert it to decimal in our head to more easily comprehend its meaning. However, the job of converting from binary directly to decimal is extremely difficult for most people.

Realizing this problem someone developed a simple method of interpretation, the use of a base eight, or octal number system. Octal numbers are very well adapted to representing binary numbers and are close enough to the decimal system to be easily comprehendable. The example below shows the relationship between binary and octal numbers. Notice that in the last step both systems had a carry.

BINARY		OCTAL
0_2	=	0_8
1_2	=	1_8
10_2	=	2_8
11_2	=	3_8
100_2	=	4_8
101_2	=	5_8
110_2	=	6_8
111_2	=	7_8
1000_2	=	10_8

We can see the significance of this now when we try to interpret the contents of a 24-bit GE-PAC word. Now we find that we can simply interpret the word three bits or binary numbers at a time. The result gives us a much easier method of representing 24 bits of information.

Example:

23							0
0 0 1	1 0 1	0 0 0	1 1 0	1 1 1	0 1 0	0 1 0	0 1 1
1	5	0	6	7	2	2	3

The general rule is to start at the binary point and interpret in groups of three to the left to find the whole number. Interpret to the right to find the fraction.

Example:

Binary Point

10	101	100	001	000.	110	111	01	Binary Number
2	5	4	1	0 .	6	7	2	Octal Equivalent

Assume that leading and trailing digits are zeroes when not shown. Remember that a binary number and its octal representation both represent the same numeric value.

Binary/octal translation is used in stepping through a program at the computer console to find an error. Through its console the computer will display patterns of lights on (ones) and off (zeroes) representing binary instruction codes, data, and core addresses, which are normally converted into octal numbers to work with.

Fortunately it is seldom necessary to make this conversion. When dealing with quantities, as distinguished from arbitrary codes and addresses, the computer nearly always performs the translation automatically.

Octal/Decimal Conversion

It is occasionally necessary to convert octal numbers to decimal numbers, and vice versa. The easiest way to do this is to refer to the tables listed in the Appendix. However, for reasons of size or desired accuracy it may be necessary to convert numbers not listed in the table. To do this we recommend the two methods described below.

- DECIMAL TO OCTAL -

To convert from decimal integers to octal integers, successively divide the remaining integer portions of any decimal integer by eight. The remainders, in inverse order, are the digits of the octal equivalent.

Example: Convert 7631_{10} to its octal equivalent.

1. Set up the following diagram.

7631	

2. Divide 7631 by 8. Put the remainder on the right side of the line as shown below.

953	7
7631	

3. Divide 953 by 8

119	1
953	7
7631	

4. Divide 119 by 8

14	7
119	1
953	7
7631	

5. Divide 14 by 8

1	6
14	7
119	1
953	7
7631	

6. Divide 1 by 8

0	1
1	6
14	7
119	1
953	7
7631	

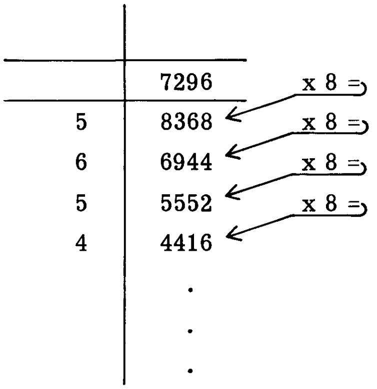
When there is a zero on the left side of the vertical line the conversion is complete.

The equivalent octal number is the string of remainders, read from top to bottom:

$$7631_{10} = 16717_8$$

To convert decimal fractions to octal fractions, successively multiply the fractional parts of the successive products by eight. Now the digits coming into the integer column, in descending order, form the digits of the equivalent octal fraction.

Example: Convert 0.7296_{10} to octal



	7296	x 8
5	8368	x 8
6	6944	x 8
5	5552	x 8
4	4416	
	.	
	.	
	.	

The conversion can be continued as long as it is practical.

The most significant digits are placed in the integer column. They are not used in the next multiplication. These digits, read from top to bottom, form the octal fraction. Thus: $0.7296_{10} = 0.5654_8$

The laws of significant digits and the rough similarity of the sizes of the bases suggest that the conversion stop when you have generated the same number of digits you were given.

- OCTAL TO DECIMAL -

Just as successive digits in a decimal number represent coefficients of successive powers of ten, so do successive octal digits represent coefficients of successive powers of eight. Thus the decimal equivalent of 16717_8 equals the sum of these terms:

$$\begin{aligned} 1 \times 8^4 &= 4096_{10} \\ +6 \times 8^3 &= 3072_{10} \\ +7 \times 8^2 &= 448_{10} \\ +1 \times 8^1 &= 8_{10} \\ +7 \times 8^0 &= 7_{10} \\ \hline & \quad 7631_{10} \end{aligned}$$

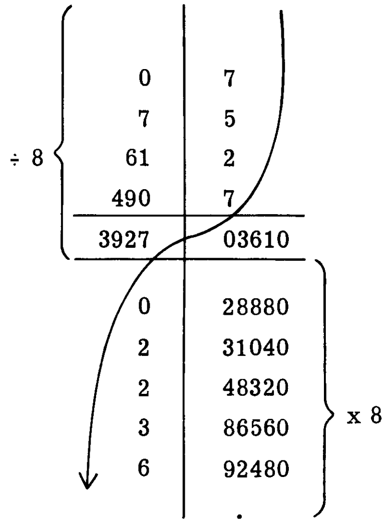
Using the same principle we can convert octal fractions to decimal fractions. For example the decimal equivalent of 0.5654_8 equals the sums of these terms:

$$\begin{aligned} 5 \times 8^{-1} &= .6250_{10} \\ +6 \times 8^{-2} &= .0936_{10} \\ +5 \times 8^{-3} &= .0100_{10} \\ +4 \times 8^{-4} &= .0010_{10} \\ \hline & \quad .7296_{10} \end{aligned}$$

The powers of eight may be obtained from the powers of two table in the Appendix.

In summary, to convert from decimal to octal use the formula shown in the following example and read off the octals in descending order.

Example:



÷ 8	0	7
	7	5
	61	2
	490	7
	3927	03610
x 8	0	28880
	2	31040
	2	48320
	3	86560
	6	92480
		.

Thus: $3927.03610_{10} = 7527.02236_8$

To convert from octal to decimal use a power of eight table as shown in the example below:

Example:

$$\begin{aligned} 1 \times 8^4 &= 4096._{10} \\ +6 \times 8^3 &= 3072._{10} \\ +7 \times 8^2 &= 448._{10} \\ +1 \times 8^1 &= 8._{10} \\ +7 \times 8^0 &= 7._{10} \\ +5 \times 8^{-1} &= .6250_{10} \\ +6 \times 8^{-2} &= .0936_{10} \\ +5 \times 8^{-3} &= .0100_{10} \\ +4 \times 8^{-4} &= .0010_{10} \\ \hline & \quad 7631.7296_{10} \end{aligned}$$

Negative Binary Numbers (Fixed Point)

All fixed-point negative numbers are represented in their two's complement form. The two's complement form of a given number is obtained by changing all zeroes to ones and all ones to zeroes and adding one as shown below.

```
Sign bit ⤻ 0 010  111  011  100.     = 2734.₈
0 = +      1 101  000  100  011.     Reversing all
1 = -                                digits
                              +1.    Add one
           1 101  000  100  100.     = -2734.₈
```

The one in the sign bit indicates that the data is negative and is represented in two's complement form. It turns out that by adding the two's complement of a number to given quantity you effectively subtract from that quantity.

Example:

```
            0001101011111      + 1537₈
                  +
            1111010100001      -  537₈
            _____________      _______
 lost   ←   0001000000000      + 1000₈
 (1)        ↓
            + 1  0  0  0₈
```

Occasionally you may be faced with either expressing a negative number in two's complement form, or interpreting a number that is written in two's complement. When converting either way follow these two steps.

1. Change all zeroes to ones and ones to zeroes.

2. Add one to the least significant digit.

APPENDIX I

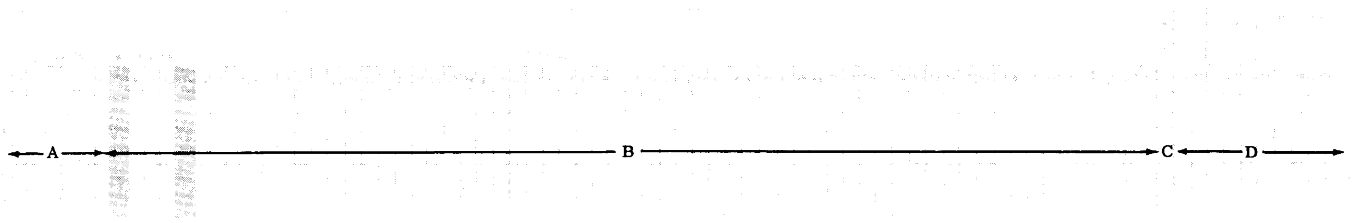
MIXED PROGRAMS

It is possible to freely mix PAL and Process FORTRAN statements in a GE-PAC 4020 program. Also, Process FORTRAN library subroutines may be used by PAL programs.

Using the Coding Sheet

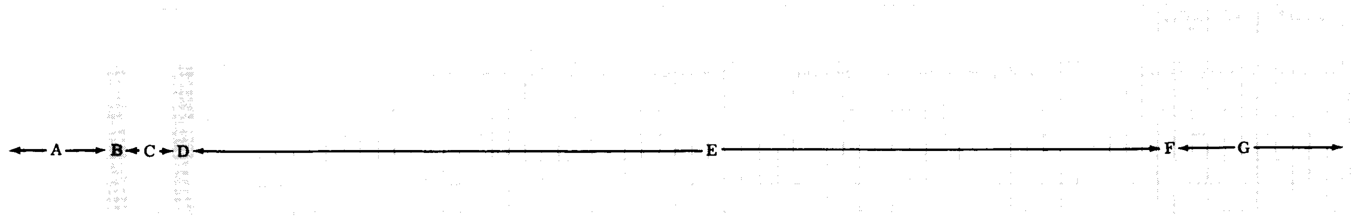
PAL and Process FORTRAN statements may be written together on the standard coding sheet. Use of the fields vary somewhat as shown below.

Writing statements in Process FORTRAN:



A - Statement Number (C in column 1 indicates a comment; any non-zero character in column 6 indicates that the line is a continuation of the previous line.)
B - Process FORTRAN Statement.
C - Column 70 must contain a 7.
D - May be used for program identification.

Writing in PAL:



A - Location name.
B - Location classification.
C - Instruction name.
D - Not used.
E - Operand.
F - Column 70 must contain a 6.
G - May be used for program identification.

Sharing Variables

Variables shared between the PAL and Process FORTRAN portions of a mixed program must be defined only once; either in PAL or in Process FORTRAN. The programmer defines variables in PAL by allocating storage areas for them either within the program, or in common core.

The compiler defines variables in Process FORTRAN by automatically allocating storage areas for them within the program. Therefore, if a variable is defined in PAL and later used in a Process FORTRAN statement, the programmer must tell the compiler not to re-define it. This is done with the Process FORTRAN DEFINE Statement described below.

DEFINE $Fv_1(Pv_1)$, $Fv_2(Pv_2)$, . . .

where: $Fv_1, Fv_2, \ldots$ are the names of the Process FORTRAN variables

$Pv_1, Pv_2, \ldots$ are the names of the corresponding PAL variables

When a Process FORTRAN variable is named in a DEFINE statement the compiler will not allocate storage area for it. Instead the compiler will use the storage area generated by the programmer for the PAL variable named within the parentheses.

The names of the Process FORTRAN variable and the PAL variable may be the same. If a number is used in place of a PAL name, the Process FORTRAN variable will use the core location referenced by that number as its storage area. The DEFINE statement should be placed just prior to the program's END statement.

Follow these general rules for treatment of variables in a mixed program:

1. If a variable is used only in the PAL portion of the program, allocate storage for it with a PAL pseudo-instruction.

2. If a variable is used only in the Process FORTRAN portion of the program, the compiler will automatically generate the necessary storage area.

3. If a variable which has been defined as a common system symbol is to be shared between the PAL and Process FORTRAN portions of a program, the PAL and Process FORTRAN names for the variable must be mentioned in a DEFINE statement.

4. If the variable which will be defined within the program is to be used both in PAL and Process FORTRAN either:

   a. Allocate storage for it in PAL and mention it in a DEFINE statement, or,

   b. Let the compiler allocate the required storage area.

Referencing Statements

PAL and Process FORTRAN statements may be freely interspersed. However, the following rules should be followed when a PAL statement is referred to by a Process FORTRAN statement and vice versa.

To reference a PAL statement from a FORTRAN control statement, place an appropriately numbered CONTINUE statement immediately ahead of the PAL statement.

To reference a FORTRAN statement from a PAL branch instruction, place an appropriately labeled "BSS O" instruction immediately ahead of the FORTRAN statement.

Examples: Referencing Process FORTRAN from PAL:

```
          :
         BTS  CALC
          :
CALC     BSS  0
        GØØDRD  =(Z*Z)+R*2
          :
```

Referencing PAL from Process FORTRAN:

```
          :
        IF (B-1000) 1,2,2
1       TR2 = A+TR2*B
          :
          :
2       CØNTINUE
         SPB  ALARM
          :
```

Saving Registers

When transferring from PAL into Process FORTRAN store the contents of A, Q, and index locations if they contain information that must be preserved. Process FORTRAN uses all of these registers and makes no effort to save and restore their contents.

APPENDIX II

OTHER PSEUDO-INSTRUCTIONS

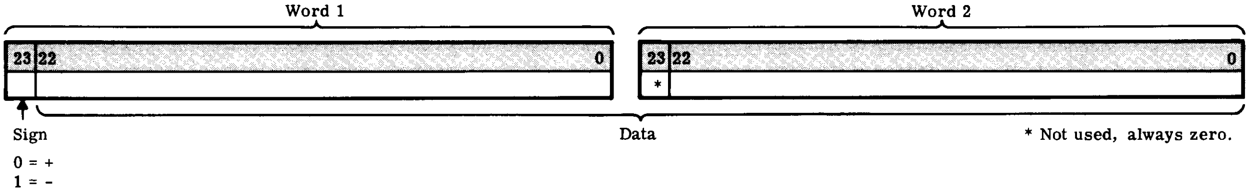
The pseudo-instructions described below are extensions of the ones described in chapter 6.

Double-Word Constants

If, for reasons of greater precision, it is necessary to define decimal, floating point, or octal constants that cannot be expressed adequately in a single GE-PAC 4020 word, the pseudo-instructions described below may be used. The rules for their usage are essentially the same as those used to define their single-word counterparts.

DCN D,(Decimal Number)(Scale Factor)

DOUBLE-WORD FIXED-POINT DECIMAL CONSTANT – The decimal number specified is converted by the assembler into a binary number and stored in the double-word format shown below. The binary point is relative to the sign bit of the first word. The sign bit of the second word is set to zero and not used.



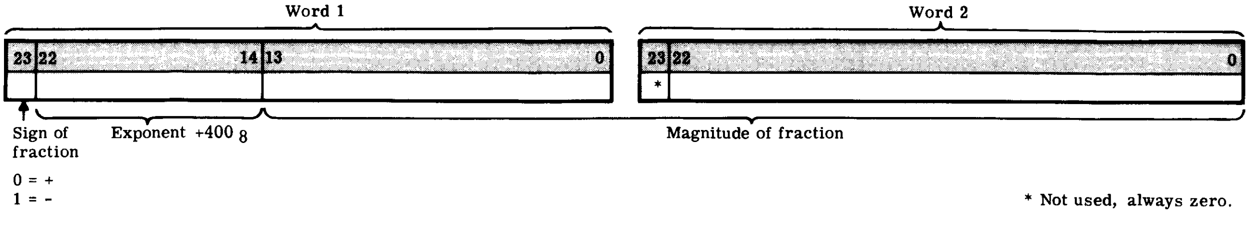
Since Bit 23 of word two is not used, Bit 22 of word two and Bit zero of word one are contiguous. Therefore, B23 is between Bit zero of word one and Bit 22 of word two. If B is not specified, B46 is assumed.

Example:

```
DCN  D,22413.796B30
DCN  D,4918724
```

DCN F,(Decimal Number)

DOUBLE-WORD FLOATING-POINT CONSTANT – The number specified is converted to binary by the assembler and stored in the format shown below.



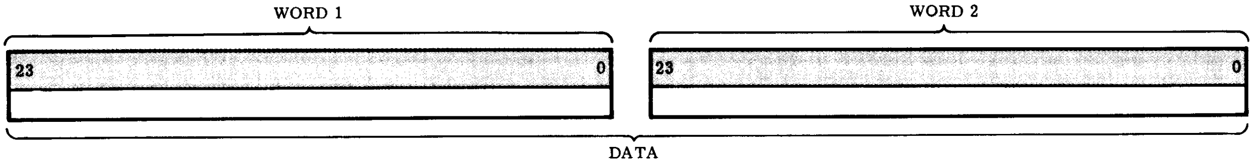
A binary exponent of zero is represented as 400_8. Numbers below 400_8 represent negative exponents; numbers above represent positive ones.

Examples:

```
DCN F,-232764.5
DCN F,49712.341E4
```

DCN O,(Octal Integer)

DOUBLE-WORD OCTAL CONSTANT – The octal integer specified is converted to binary and entered, right justified, into the double-word format shown below. All 48 bits of the two words are used for data, there is no sign.



Examples:

```
DCN Ø,77777777774
DCN Ø,324017521
```

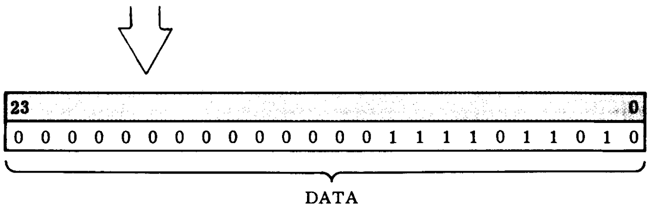
General Constant

CON G,(Label or Integer)

GENERAL CONSTANT – If a label is specified it must be defined as an integer value either in the program where the CON G is used or in the common system symbol table. The CON G causes the assembler to insert the integer or label value, right justified, into the program in place of the pseudo-instruction.

Example:

```
CØN G,WØRDA     Where WORDA =/1732
```



Generating Duplicate Instructions

GEN K

GENERATE DUPLICATES – Specifies to the assembler that the next instruction must appear K times. Symbols must be predefined.

Example:

```
GEN  3
CØN  Ø, 4
```

This coding . . .

```
CØN  Ø, 4
CØN  Ø, 4
CØN  Ø, 4
```

is equivalent to this.

NOTE: The GEN instruction cannot be used to duplicate any of the following PAL instructions:

ORG
BSS
DCW
CON A
GEN
EQL
DEF
SLW
END
LIB
IDN

Page Positioning

SLW

SLEW PRINTER PAGE – Causes the assembler to position the listing at the top of the next page.

Defining a New Operation

Using the CON O pseudo-instruction it is possible to set up a fixed bit pattern within a word. Occasionally, however, it would be convenient to be able to build a word with fixed fields but still be able to insert variable data within those fields. For example, this capability would be handy in setting up list control words for circular lists. The format for a list control word is always the same as shown below it contains five operand fields, each with a unique position and a fixed length. The data within the word, however, may change as we specify different lists.

23 15	14 6	5	4	3 0
F	N	f	e	L

LIST CONTROL WORD

The DEF pseudo-instruction permits us to define an operation that will insert data into specific operand fields within a word. The following procedure is used to define a new operation:

A. Set up the base octal.

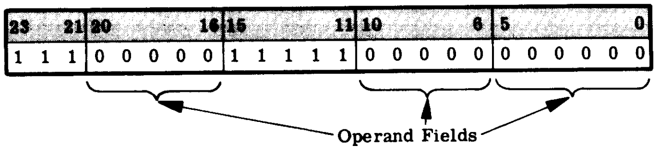
B. Assign audit codes to operand fields.

C. Define the new operation.

This procedure is described in detail below.

A. Set up the base octal.

The base octal for an operation is normally determined by defining the contents of the word which are not specified as operand fields. The contents of the operand fields themselves are usually set to zero. For example, we wish to define a new operation which will construct words with three operands as shown below; the bits not specified in operands must be set to one.



23 21	20 16	15 11	10 6	5 0
1 1 1	0 0 0 0 0	1 1 1 1 1	0 0 0 0 0	0 0 0 0 0 0

The resulting base octal for this word would be 70174000. In the case of defining an operation to set up a list control word the base octal would be 00000000, since all 24 bits of the word are included in operand fields.

B. Assign audit codes to operand fields.

Audit codes are used by the assembler program to determine operand field width and position within a word. There are 64 audit codes numbered from zero to sixty-three – the first 51 of these plus audit code 63 are already defined for use by the assembler and the RTMOS; the rest may be used by the programmer to define new operations.

In defining a new operation, each operand field width and its position must be assigned an audit code in the following manner.

(AUDIT CODE) DEF (OPERAND FIELD WIDTH),(ANCHOR BIT)

- (AUDIT CODE) - any decimal number from 51 up to and including 62.
- DEF - pseudo-instruction which assigns a field width and anchor bit to an audit code.
- (OPERAND FIELD WIDTH) - specifies the maximum width of an operand field within the word. This width is expressed as the largest octal number the field can contain; i.e., a five-bit operand width is indicated by 37, a two-bit operand width is indicated by 3.
- (ANCHOR BIT) - specifies the placement of the operand within the word. It is the number of the rightmost bit in that operand field.

Example: In the process of defining an operation to set up list control words, the following audit code assignment is necessary.

```
62      DEF  /777 , 15  ←—— specifies the ' F" operand field
61      DEF  /777 , 6   ←—— specifies the ' N" operand field
60      DEF  /1 , 5     ←—— specifies the ' f" operand field
59      DEF  /1 , 4     ←—— specifies the ' e" operand field
58      DEF  /17 , 0    ←—— specifies the ' L" operand field
```

C. Define the new operation.

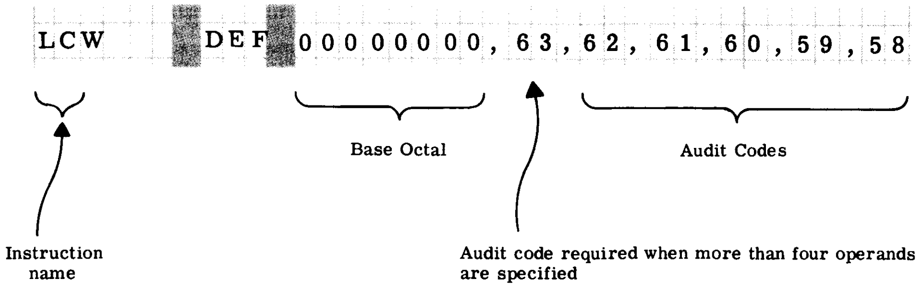
To do this we use the following general form.

(OPERATION NAME) DEF (BASE OCTAL),(FIRST AUDIT CODE),(SECOND AUDIT CODE),ETC.

The operation name may be any three letter combination not already used for a PAL instruction or pseudo-instruction.

If more than four audit codes are specified, they must be preceded by audit code No. 63. A maximum of 12 operand fields are permissible.

Example: With the information previously determined in steps one and two, the operation to build list control words may be defined. Assume that the name of the new operation will be LCW.



```
LCW     DEF 00000000,63,62,61,60,59,58
```

Base Octal
Audit Codes
Instruction name
Audit code required when more than four operands are specified

After this new operation has been defined it may be used to build any list control word.

Example: The new operation to build list control words has been previously defined and it is necessary to define an empty list thirty-two words long. The following parameters for the word are necessary.

F = 0 since beginning item is at list location zero.

N = 0 since there are no items currently in the list.

f = 0 since list is not full.

e = 1 since list is empty.

L = 5 since list size is 32.

With these parameters, the list control word may be specified.

```
LISTA  LCW 0,0,0,1,5
```

When this is read by the assembler it will generate the appropriate octal word (00000025) and insert it in the place of LCW, and associate the name LISTA with that word.

We have used the definition of list control words as an example to show how an operation is defined. New operations may also be defined for many other purposes — specifying scanner control words, packed tables, and even defining instructions to be executed when the program is run.

The following general rules should be observed when defining new operations.

1. DEF may not be used to substitute new definitions for standard PAL instructions or pseudo-instructions.

2. Operands of the DEF instruction itself may be written in decimal, octal, or symbolic. However, all symbols must be previously defined. All operands are considered absolute.

3. Operands for newly defined op codes follow the usual rules for translation as absolute or relative symbolics.

4. Audit codes must be numeric.

5. Audit codes 51-62 may be defined. It is recommended that the programmer start with 62 and number in reverse.

6. Audit codes and op codes not reserved for PAL may be redefined within a program as often as the programmer desires.

7. New op codes, audit codes, and their audit code definitions are made a part of the common system symbol table which may be preserved for subsequent assemblies.

Error conditions — When an illegal attempt is made to define an op code or audit code the assembler will ignore the attempt and print a location error flag (L) on the listing.

APPENDIX III

INSTRUCTION FORMATS

Instructions, like data, must be represented in standard formats. It is helpful to know the formats for these instructions when machine level decoding is necessary. The formats and instruction codes for specific instructions are described in detail in the Instruction Reference Manual. There are four general types of hardware instructions in the GE-PAC 4020 computer; full operand, GEN 1, GEN 2, and GEN 3.

Full Operand Instructions

Full operand instructions are used to perform arithmetic operations, data transfers, bit counting, masking, etc.

23 18	17 15	14	13 0
Op Code	X	*	K or Y

OP - Operation code for the instruction

X - Index location number, if used

* - If set, relative addressing occurs

K or Y - Operand

GEN 1 Instructions

GEN 1 instructions are used for bit manipulation of the A register; shifting right, bit counting, masking, etc.

23 18	17 15	14 5	4 0
Op Code	X	G	K

OP- Operation code for all GEN 1 instructions is 05_8

X - Index location number, if used

G - Micro-coding designating the function of the command

K - Operand

GEN 2 Instructions

GEN 2 instructions are used for certain forms of input and output.

23 18	17 15	14 12	11 0
Op Code	X	G	D

OP- Operation code for all GEN 2 instructions is 25_8

X - Index location number, if used

G - Subcommand to the computer or the I/O device

D - I/O device address

GEN 3 Instructions

GEN 3 instructions are used for shifting A and Q in either direction or shifting A to the left.

23 18	17 15	14 5	4 0
Op Code	X	G	K

OP- Operation code for all GEN 3 instructions is 45_8

X - Index location number, if used

G - Microcoding designating the function of the command

K - Operand

Examples: Where Y = /500

INSTRUCTION	OCTAL REPRESENTATION
LDA Y	00 00 05 00
STA Y + 1	32 00 05 01
STA Y+1,3	32 30 05 01
BRU *+2	14 04 00 02
LDA *Y+1,5	00 54 05 01
RBK 5	05 04 50 05
SRA 7,3	05 31 40 47
SOD 14	05 00 45 16
SLA 10,5	45 50 20 52
MAQ	45 00 43 30

Quasi Instructions

A feature of the GE-PAC 4020 computer is its ability to use quasi instructions. Most instructions are executed by hardware. Quasi's are implemented by software; instead of a hardware operation, they call for the execution of a subroutine to perform their function. The Instruction Reference Manual points out the various quasi's implemented on the GE-PAC 4020 computer.

APPENDIX IV

ASSEMBLER ERROR FLAGS

The PAL assembler performs validity tests on each instruction. When errors or suspected errors are detected, one of the following indicators will appear on the output listing.

FLAG	DEFINITION	CAUSE
L	Location Field Error	1. First character of the label is not alphabetic. 2. Using the DEF pseudo-instruction when: a. The operation name assigned is a GE-PAC machine operation. b. Requesting "extra operands" definition when operation name has been previously defined as a machine operation. c. There is an illegal audit code number. 3. Location field is blank when a symbol is required. 4. Location field contains a symbol when not allowed. 5. Label not found in the table, probably due to overflow of the table on the first pass.
O	Operation Field Error	1. The op code not part of the language or was not added to the table through the DEF pseudo-instruction. This often occurs when definition was attempted but was illegal. Consequently, it was not added to the operation table. 2. This op code cannot be GENerated.
I	Illegal Operand	1. Blank operand when an operand is required. 2. Operand not blank when it should have been. 3. One or more required operands missing. 4. Too many operands. 5. Operand value too large. 6. Negative operand value in an instruction that will not accept one. 7. Illegal constant.
X	Index Word Error	1. Index word 1 or 2 specified. 2. Required index missing. 3. Specified index word is greater than seven.

ASSEMBLY ERROR FLAGS (CONT.)

FLAG	DEFINITION	CAUSE
U	Undefined Symbol	Occurs only when a symbol appears in the operand field and: 1. It never appeared in the location field or on the common symbol tape. 2. It appeared in the location field, but the symbol table was full at the time.
C	Illegal Character	An illegal character was found in the location, op code, or operand field.
M	Multiply-defined Symbol	1. A symbol in location field was flagged because: a. It has appeared in the location field of a previous statement. b. It appeared on the requested EQL tape with a value unequal to the one being assigned. c. It was saved from a previous assembly with a value unequal to the one being assigned. 2. Any record which references a multiply-defined symbol in the operand field will also be flagged.
2	Second Pass Definition of Symbol Different from First Pass	1. Symbol was not defined prior to use in operand of a BSS, EQL, or GEN pseudo-instruction. 2. Occasionally the result of a Multiply-defined Symbol.
R	Relative Operand Error	Operand value was relative and should normally be absolute for this operation.
F	Tables Full	Assembler has room to store a fixed number of symbols. This flag occurs when a program, symbols exceed the capacity of the assembler.

APPENDIX V

OCTAL/DECIMAL CONVERSION TABLE

There are two tables in this section, one for integers, and one for fractions. To convert a number from octal to decimal, or vice versa:

1. Obtain the integer portion from the integer table.
2. Obtain the fractional portion from the fraction table.

Example:

Find the octal equivalent of 1794.3613_{10}

From the integer table on page 77.

Octal	0	1	2	3
3400	1792	1793	1794	
3410	1800	1801		

$1794_{10} = 3400_8 + 2_8 = 3402_8$

From the fraction table on page 82.

Octal	Decimal
.270	.359375
.271	.361328
.272	.363281
.273	.365234

$.361_{10} = .271_8$

The accuracy of the conversion could be improved by interpolation. Putting the integer and fractional portions together:

$$1794.361_{10} = 3402.271_8$$

When converting numbers greater than 4095_{10} or 7777_8 use the block at the top of integer tables to break out the largest portion of the number. Evaluate the rest using the integer tables.

Example: Find the decimal equivalent of 40741_8

Octal	10000	20000	30000	40000
Decimal	4096	8192	12288	16384

$$\begin{aligned} 40000_8 &= 16384_{10} \\ \underline{741_8} &= \underline{481_{10}} \\ 40741_8 &= 16865_{10} \end{aligned}$$

OCTAL DECIMAL CONVERSION TABLES

Octal-Decimal Integer Conversion Table

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	0000 to 0377
Decimal	0000 to 0255

Octal	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

Octal	0400 to 0777
Decimal	0256 to 0511

Octal	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

Octal	1000 to 1377
Decimal	0512 to 0767

Octal	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0149	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

Octal	1400 to 1777
Decimal	0768 to 1023

Octal	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

Octal-Decimal Integer Conversion Table

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	2000 to 2377
Decimal	1024 to 1279

Octal	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

Octal	2400 to 2777
Decimal	1280 to 1535

Octal	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

Octal	3000 to 3377
Decimal	1356 to 1791

Octal	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729.	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

Octal	3400 to 3777
Decimal	1792 to 2047

Octal	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

Octal-Decimal Integer ConversionTable

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	4000 to 4377
Decimal	2048 to 2303

Octal	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

Octal	4400 to 4777
Decimal	2304 to 2559

Octal	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	3537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

Octal	5000 to 5377
Decimal	2560 to 2815

Octal	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

Octal	5400 to 5777
Decimal	2816 to 3071

Octal	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

Octal-Decimal Integer Conversion Table

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	6000 to 6377
Decimal	3072 to 3327

Octal	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	2354	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

Octal	7000 to 7377
Decimal	3584 to 3839

Octal	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

Octal	6400 to 6777
Decimal	3328 to 3583

Octal	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

Octal	7400 to 7777
Decimal	3840 to 4095

Octal	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

Octal-Decimal Fraction Conversion Table

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

Octal-Decimal Fraction Conversion Table

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

Octal-Decimal Fraction Conversion Table

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

Table of Pcwers of 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

INDEX TO INSTRUCTIONS AND PSEUDO-INSTRUCTIONS

Progress Is Our Most Important Product

GENERAL ELECTRIC

PROCESS COMPUTER BUSINESS SECTION
PHOENIX, ARIZONA