JOSS: A DESIGNER'S VIEW OF AN EXPERIMENTAL ON-LINE COMPUTING SYSTEM

J. C. Shaw

August 1964

P-2922

Copyright © 1964
THE RAND CORPORATION

JOSS: A DESIGNER'S VIEW OF AN EXPERIMENTAL ON-LINE COMPUTING SYSTEM

J. C. Shaw*

The RAND Corporation, Santa Monica, California

ABSTRACT

JOSS (JOHNNIAC Open-Shop System) is an experimental on-line, time-shared computing service. It is in daily use by staff members of The RAND Corporation for the solution of small numerical problems. The users compose stored programs and interact with JOSS through remote typewriter consoles by using a single, high-level language.

The system is described with emphasis on those features which have led users to accept it as a convenient new tool. JOSS provides use of familiar typewriters, exact input/output, decimal arithmetic, high-level algebraic language with English punctuation rules, easy modification and repair of programs, and report-quality formatted output.

---

*Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views or opinions of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

This paper was prepared for presentation at the 1964 Fall Joint Computer Conference, sponsored by the American Federation of Information Processing Societies, October 27-29, 1964, at San Francisco, California.

INTRODUCTION

The JOHNNIAC Open-Shop System (JOSS) is an experimental, on-line, time-shared computing system[1] which has been in daily use by staff members of The RAND Corporation since January 1964.* It was designed to give the individual scientist or engineer an easy, direct way of solving his small numerical problems without a large investment in learning to use an operating system, a compiler, and debugging tools, or in explaining his problems to a professional computer programmer and in checking the latter's results. The ease and directness of JOSS is attributable to an interpretive routine in the JOHNNIAC computer which responds quickly to instructions expressed in a simple language and transmitted over telephone lines from convenient remote electric-typewriter consoles. An evaluation of the system has shown that in spite of severe constraints on speed and size of programs, and the use of an aging machine of the vacuum-tube era, JOSS provides a valuable service for computational needs which cannot be satisfied by conventional, closed-shop practice.

This paper concentrates on the numerous, small, hardware and software design decisions which have influenced the acceptance of the system by its intended users. Several figures, produced on-line, are included, providing readable examples of features of the JOSS language.

---

*An austere version of the system saw limited use during most of 1963.

BACKGROUND

From the earliest days of construction of the JOHNNIAC computer, a Princeton-class machine built at The RAND Corporation in 1950-53, it has been the author's dream to have an economical, personal, remote communication station for on-line control and programming of a computer. With so much to be learned about programming and operating large, general-purpose computers, it isn't surprising that the additional investment in communications equipment, remote stations, and corresponding software was postponed.

In its early days, JOHNNIAC served well as a production machine. Then, because it has only a 4096-word core memory, a slow 12,288-word drum, slow copy-logic for card I/O and printing, no tapes, and a very austere order code, production computing was gradually shifted to more modern IBM equipment. Yet, the very accessibility to this unsaturated second machine made JOHNNIAC attractive as the basis for simplified programming systems for small, open-shop problems and for experimental work in heuristic programming, new software systems, and hardware for better interaction with a computer. In November 1960, after years of discussion of personal remote consoles with T. O. Ellis, I proposed to the management of RAND's Computer Sciences Department that JOHNNIAC be committed full time to providing a modest computing service to the open-shop via remote typewriters.

The purpose of the JOSS experiment was not to make JOHNNIAC machine language available, but rather to provide a service through a new, machine-independent language which had to be designed specifically for the purpose. It was to be an experiment with the goal of demonstrating the value

of on-line access to a computer via an appropriate language, and was intended to contribute to a project with the long-range goal of a sophisticated information processor. T. O. Ellis, I. Nehama, A. Newell, and K. W. Uncapher were the other participants in that project.

In 1961-62, Ellis and M. R. Davis designed and directed the construction of the required multiple typewriter communication system adjunct to JOHNNIAC. The hardware was ready well in advance of the first version of the system program, and only a few select users were subjected to this very limited system. Their feedback, including encouraging remarks on the usefulness of JOSS, helped shape the full version.*

COMPARISON

Other on-line, time-shared computing systems have become operational in recent years.(2-6) All are pioneering efforts. By comparison, JOSS is special-purpose, even though it encompasses a wider class of problems than one might guess at first reading. Most of the others provide the user with access to machine language. F. J. Corbató has aptly described them as <u>open</u> systems and JOSS as a <u>closed</u> system. In the open systems, an executive routine is prepared to help the user at the machine-language level or to pass control to one of several subsystems providing adaptations of pre-existing programming systems. JOSS,

---

*We wanted to do a controlled evaluation of the system at the time of the introduction of the full version of JOSS, but the new users taught others so quickly that we had to resort to after-the-fact questionnaires!

however, was designed with on-line interaction in mind, and resources were devoted to making it smooth and easy to use. The future lies with the open systems, but it remains to be seen whether the open-system executive will absorb JOSS-like systems simply as additional subsystems, or whether JOSS-like systems will absorb the executive function and thus serve as the user's computing aide and single contact with the computer.

HARDWARE COMPONENTS OF JOSS

Physically, JOSS consists of the JOHNNIAC computer, ten remote consoles, and a multiple typewriter communication system to mediate between JOHNNIAC and the consoles.

JOHNNIAC

RAND, as did several universities and research institutions in the early 1950s, constructed a computer (called "JOHNNIAC" for John von Neumann) more or less on the pattern of the machine of the Institute for Advanced Study at Princeton. JOHNNIAC was upgraded in 1954 with the replacement of the original 256-word Selectron memory with a 4096-word magnetic core memory. The word length is 40 bits. Because JOHNNIAC was ill-equipped to handle the message traffic required in JOSS service, a special-purpose buffering system was built to process characters within messages and to monitor the remote stations. The alternative of modifying the main frame to handle the message traffic directly would have required a major rework of the JOHNNIAC control and would still have yielded

degraded performance in JOSS service. Thus, JOHNNIAC remains a very primitive machine with no indexing, no indirect addressing, no floating point, no error checking, no memory protect, no interrupts, no channels, no compare, no zero test, a miserable format of two single-address instructions per word, and a 50-μs add time.

The JOSS system program runs about 6000 words, the low-frequency portions residing on drum and overlaying each other in core when called in for execution. A large part of the JOSS system program resides permanently in core. It was a considerable challenge to compress it sufficiently to leave room for the processing of a user's block in core. More than once I regretted the lack of an adequate subroutine linkage operation; it would have saved much space in this deeply hierarchical program.

The 12,288-word JOHNNIAC drum is divided into three sections, accessed by moving heads at a rate not quite so fast as a modern disk unit unless the heads are luckily in the correct position. Average swap time (i.e., the time to write one user's block of information out onto drum and read a second user's block into core for processing) is, therefore, quite slow at about half a second.

COMMUNICATION SYSTEM

The multiple typewriter communication system provides sixteen line-buffers, controls the states of all ten remote consoles, and registers signals from them. The limit is 81 consoles--well beyond our needs and our budget. The JOSS system program in JOHNNIAC commands block transfers between core and the line buffers. It also commands the

communication system to enable or disable a console, request or relinquish control of a console, clear a line buffer, assign a line buffer to a console, or transmit a line buffer to a console. It also commands the communication system to report any signals from consoles indicating a carriage return, a page ejection, or the depression of one of the console control keys.

REMOTE CONSOLE (Fig. 1)

Lights and switches in a small box augment the IBM model 868 typewriter to indicate the status and to control the functions of the local communication terminal electronics. The switches are: a POWER switch; an ON switch to connect the terminal to JOSS; an OFF switch to disconnect; a READY switch to reactivate the typewriter after inserting a fresh supply of paper; an IN switch to request control of the typewriter for input; and an OUT switch to relinquish control for output. Indicators are provided as follows: a POWER light; an ENABLE light showing that JOSS service is available; a READY light showing that output is acceptable at the typewriter; a red light to show that JOSS controls the typewriter; a green light to show that the user controls it; an IN REQUEST light to show that the user has depressed the IN button for control but JOSS hasn't yet responded; and an OUT REQUEST light to inform the user that JOSS has an administrative message for him (such as "Shutting down at 2330.").

The READY light goes out if the paper supply is exhausted or if the paper jumps the sprockets. The user may also switch the READY light off any time he wants to

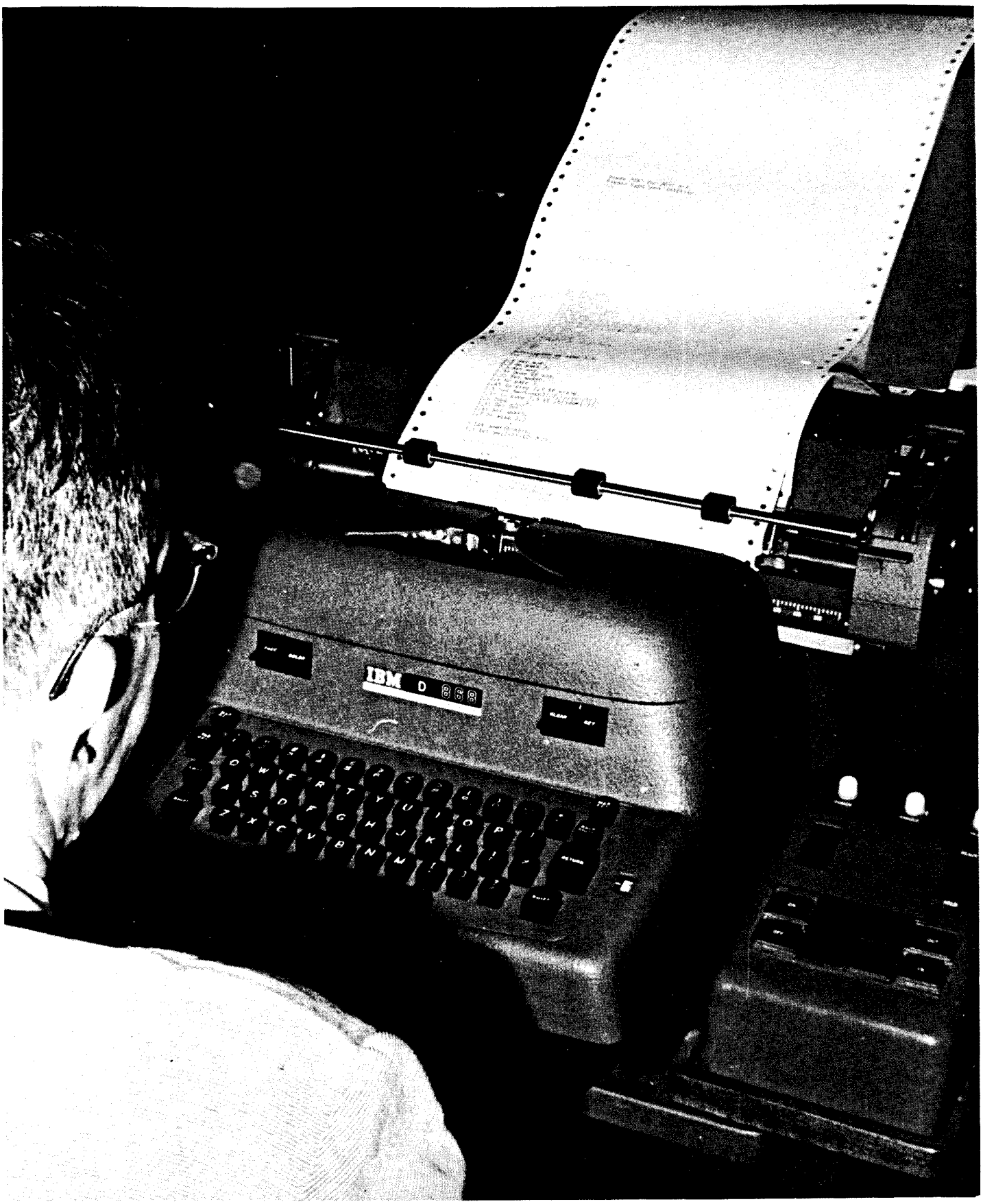

Fig. 1—JOSS console and station local control box

hold up output momentarily. To continue with the output, the READY light is turned back on; no information is lost. The philosophy is one of exclusive control of the typewriter. When JOSS has control, the red light is on, the keyboard is locked, and the typewriter ribbon color is black. As JOSS turns control of the typewriter back to the user, the light changes to green, the keyboard unlocks, the ribbon color changes to green, and a soft gong rings. These visual, tactile, and audible signals leave no doubt as to who controls the station.

If a remote typewriter console is to be a personal instrument, it must also serve as a simple typewriter. This consideration dictated that the console allow for off-line use and that the character set include all the normal punctuation of a typewriter. The sprockets and paging mechanisms restrict the stations from being entirely satisfactory as personal typewriters, because of the problem of changing paper and the excessive noise. However, the hard copy produced is excellent--quite acceptable for reports without further transcription and chance for error.

KEYBOARD (Fig. 2)

The choice of character set and key positions for any on-line keyboard input device isn't to be taken lightly, especially if one hopes to encourage senior technical people to use the keyboard in the direct solution of their problems. It is customary for these people to pay others to drive teletypewriters, keypunches, and even typewriters. For the JOSS remote typewriters, we left the comma, period, semicolon, colon, slash, question mark, quotes, space sign,

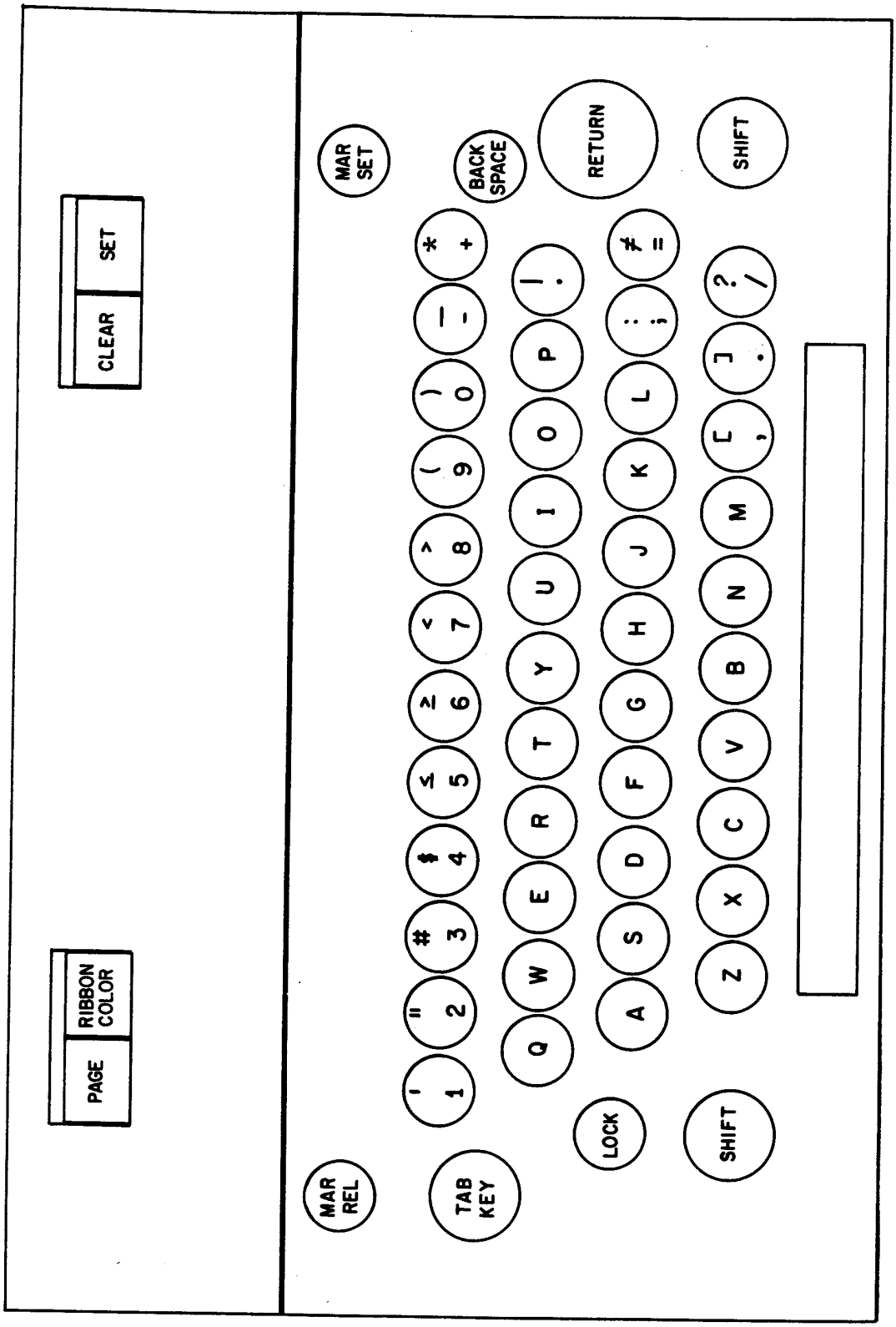

Fig. 2—JOSS typewriter keyboard

dollar sign, parentheses, and hyphen in their customary positions. The less essential characters of standard sets were sacrificed in order to make room for all six numerical relation symbols.

To linearize numerical expressions requires an explicit sign for exponentiation, for which we chose a five-pointed, upward-pointing, slightly elevated asterisk in upper case (all of which contribute proper associations for exponentiation). The plus, minus (hyphen), centered dot (for multiplication), slash (for division), and equals sign are all in lower case. Left and right brackets were included, in place of the upper-case comma and period, in order to improve readability of linearized expressions (otherwise, such expressions tend to become cluttered with parentheses). The absolute value bar also contributes somewhat to readability. Parentheses and brackets are interchangeable in pairs for all grouping functions: subexpressions, arguments for functions, indices, and interval size in iteration expressions.

The punctuation and capitalization rules for JOSS are quite conventional, but three symbols are used in very special ways. The space sign (#) is used as a strikeover character, causing a character already in the input line buffer to be replaced by a space. This is needed since, if the typed line is to reflect the contents of the input buffer, the space bar and backspace key must never <u>enter</u> any character into the buffer (although they do control position). The asterisk (*) at either end of an instruction input line leads JOSS to ignore the line and thus provides a device for annotating one's work and for cancelling lines. (For most errors, however, the process of

simply backspacing and striking over is adequate.) The dollar sign ($) may be used in any expression and carries a numerical value equal to the line number (from 1 to 54) of the typewriter's current position on the page. This value is updated by JOSS so the user can easily control format on the page. If no format is specified, JOSS supplies an automatic one-inch margin at the top and bottom of each page.

The tab may be used to speed output typing by skipping instead of spacing over blank positions. The typing speed of ten characters per second, less shifting time, has not been a source of dissatisfaction. A key interlock, intended to insure that only one key at a time was depressed, was abandoned because of the frustrating effect on the user. The action of the keys without the interlock is admirable, but it is possible to hit two keys at once, superimposing their character codes for transmission. This risk is more acceptable than the interlock, but it means that JOSS must be prepared to receive any 7-bit character code--not just the legal ones. The keyboard lock (not to be confused with the now-abandoned key interlock) is incomplete in that it locks only the typing keys, and even then it can be overriden. This deficiency has not been a problem however since, at most, the user can spoil only his own output by trying to use the typewriter out of turn.

SOFTWARE

JOSS services the requests of users at the remote consoles in such a way that the users' activities are

logically independent of one another. Up to eight of the ten stations may be served concurrently by time-sharing techniques. In addition to administering input/output and swaps of user blocks, JOSS interprets and executes both direct and indirect (i.e., stored-program) instructions couched in a readable and easily learned language.

TIME SHARING

The basic JOHNNIAC computer provides no parallel processing. The multiple typewriter communication system does provide for parallel activity at many consoles by high-speed line-scanning and time-shared use of the logic circuits. JOSS takes advantage of this independent parallel processing in the communication system by time sharing--i.e., by switching its attention rapidly from one user to another to give adequate service to all active users. Each active user is represented by a block of information which resides on the drum, except when JOSS is actually processing it in core.

First priority for JOSS' attention goes to the servicing of signals from the consoles: carriage return, page, on, off, in, out, and end-of-transmission. JOSS looks for these signals in the communication system when idling, and between interpretive steps when executing a user's program. An end-of-transmission signal requires only that JOSS record that the line buffer is available, and direct the transmission of the next line of output to the same station if one is ready. JOSS then continues with its previous activity. A carriage return, however, like several other signals, requires that JOSS break off its current activity, move the current user's block out to drum, move the signaling

user's block into core, and, finally, interpret and act on the line of input just released by the carriage return.

Second priority is given to users who have given JOSS output-limited tasks which have been set aside until the typewriters have nearly caught up. Third priority is given to users with unfinished tasks, on which JOSS works for two seconds apiece in round-robin fashion. A user's priority changes dynamically according to this discipline, which successfully exploits the parallel processing of the communication system. Under a typical load, JOSS responds to simple requests in a fraction of a second and rarely in as long as three seconds. Users who are skilled in typing can maintain impressive rates of interaction with JOSS.

LANGUAGE

The reader will observe that throughout this section on software, the term "JOSS" is used to refer to that single active agent at the computer end of the telephone line connecting the user's remote console. It is convenient to consider JOSS to be a "computing aide" interacting with the user by means of a simple language. The reader should now read the examples (Figs. 3a-3i) before continuing in this section. The examples fall short of being an adequate instruction manual for the system, but they suggest the readability of the language, the high degree of interaction, and the power of expression.

A striking feature of the system is that the user commands JOSS directly in the same language that he uses to define procedures for JOSS to carry out indirectly. A

```
U:  Type 2+2.
J:       2+2 =       4

U:  Set x=3.
    Type x.
J:         x =       3
U:  Type x+2, x-2, 2·x, x/2, x*2.
J:       x+2 =       5
         x-2 =       1
         2·x =       6
         x/2 =       1.5
         x*2 =       9

U:  Type [(|x-5|·3+4)·2-15]·3+10.
J:  [(|x-5|·3+4)·2-15]·3+10 = 25
```

U - Denotes inputs of the JOSS user; normally typed in green.
J - Denotes outputs from JOSS; normally typed in black.

(3a)

Fig. 3—Samples of JOSS language and interaction

```
U:  Type sqrt(3), sqrt(4).
J:   sqrt(3) =      1.73205081
     sqrt(4) =      2

U:  Type sqrt(-1).
J:  Error above:  Negative argument for sqrt.

U:  Set e=2.71828183.
    Type log(1), log(2), log(e).
J:    log(1) =      0
      log(2) =       .69314718
      log(e) =      1

U:  Type exp(0), exp(.5), exp(1).
J:    exp(0) =      1
     exp(.5) =      1.64872127
      exp(1) =      2.71828183

U:  Type sin(.5), cos(.5), sin(.5)*2+cos(.5)*2.
J:   sin(.5) =       .479425539
     cos(.5) =       .877582562
    sin(.5)*2+cos(.5)*2 = 1

U:  Type arg(1,0), arg(0,1), arg(-1,0), 4·arg(3,3).
J:  arg(1,0) =      0
    arg(0,1) =      1.57079633
    arg(-1,0) =     3.14159265
    4·arg(3,3) =    3.14159265
```

(3b)

```
U:  Set y = 123.456.
    Type y, ip(y), fp(y), dp(y), xp(y).
J:          y =    123.456
         ip(y) =    123
         fp(y) =        .456
         dp(y) =      1.23456
         xp(y) =      2
U:  Type 1.23456·10*2.
J:  1.23456·10*2 = 123.456

U:  Type sgn(-3.5), sgn(0), sgn(3.5).
J:  sgn(-3.5) =    -1
      sgn(0) =     0
    sgn(3.5) =     1

U:  Type max(1,2,3), min(1,2,3).
J:  max(1,2,3) =    3
    min(1,2,3) =    1
```

(3c)

U:	Delete all. 1.1 Do part 2 for b=1(1)a. 2.1 Set c = sqrt(a*2 + b*2). 2.2 Type a, b, c in form 1. Form 1: a = __ b = __ c = __.___
J:	Do part 1 for a=1(1)3. a = 1 b = 1 c = 1.414 a = 2 b = 1 c = 2.236 a = 2 b = 2 c = 2.828 a = 3 b = 1 c = 3.162 a = 3 b = 2 c = 3.606 a = 3 b = 3 c = 4.243
U:	Type all steps.
J:	1.1 Do part 2 for b=1(1)a. 2.1 Set c = sqrt(a*2 + b*2). 2.2 Type a, b, c in form 1.
U:	Type all forms.
J:	Form 1: a = __ b = __ c = __.___
U:	Type all.
J:	1.1 Do part 2 for b=1(1)a. 2.1 Set c = sqrt(a*2 + b*2). 2.2 Type a, b, c in form 1. Form 1: a = __ b = __ c = __.___ a = 3 b = 3 c = 4.24264069

```
U:  2.15 Line if fp(c)=0.
    2.2 Type a, b, c in form 1 if fp(c)=0.
    Type part 2.
J:  2.1 Set c = sqrt(a*2 + b*2).
    2.15 Line if fp(c)=0.
    2.2 Type a, b, c in form 1 if fp(c)=0.
U:  Do part 1 for a=1(1)15.

J:  a =  4   b =  3   c =  5.000

    a =  8   b =  6   c = 10.000

    a = 12   b =  5   c = 13.000

    a = 12   b =  9   c = 15.000

    a = 15   b =  8   c = 17.000

U:  Delete part 2.
    Type part 2.
J:  Error above:  No such part.
U:  Type all values.

J:        a =     15
          b =     15
          c =     21.2132034

U:  Delete all.
```

(3e)

```
U:  3.1 Type x, sqrt(x), log(x), exp(x), __.
    Do step 3.1 for x=1,2,3,100.
J:          x =      1
     sqrt(x) =      1
      log(x) =      0
      exp(x) =      2.71828183

            x =      2
     sqrt(x) =      1.41421356
      log(x) =       .69314718
      exp(x) =      7.3890561

            x =      3
     sqrt(x) =      1.73205081
      log(x) =      1.09861229
      exp(x) =     20.0855369

            x =    100
     sqrt(x) =     10
      log(x) =      4.60517019
      exp(x) =      2.68811714·10*43
```

(3f)

```
U:  3.1 Type x, sqrt(x), log(x), exp(x) in form 3.
    Form 3:
    _.__   _.___   _.____   .........

    Do step 3.1 for x = 8.50(.01)8.54(.02)8.60, 9, 9.5.
J:  8.50   2.915   2.1401    4.915 03
    8.51   2.917   2.1412    4.964 03
    8.52   2.919   2.1424    5.014 03
    8.53   2.921   2.1436    5.064 03
    8.54   2.922   2.1448    5.115 03
    8.56   2.926   2.1471    5.219 03
    8.58   2.929   2.1494    5.324 03
    8.60   2.933   2.1518    5.432 03
    9.00   3.000   2.1972    8.103 03
    9.50   3.082   2.2513    1.336 04
```

(3g)

```
U:    4.1 Demand b(i).
      4.2 Set s = s+b(i).

      Set s=0.
      Do part 4 for i=1(1)4.
J/U:      b(1) = 543
          b(2) = 237
          b(3) = 411
          b(4) = 733
U:    Type s.
J:            s =   1924
U:    5.1 Set b(i) = b(i)/s.
      Do part 5 for i=1(1)4.
      Type b.
J:        b(1) =        .282224532
          b(2) =        .123180873
          b(3) =        .213617464
          b(4) =        .380977131
U:    Set s=0.
      Do step 4.2 for i=1(1)4.
      Type s.
J:            s =      1
U:    Delete all.
```

(3h)

```
U:  6.1 Type form 61.
    6.2 Line.
    6.3 Do part 7 for t=0(h)4.

    7.1 Type t, a in form 62 if fp(t/.5)=0.
    7.2 Do part 8 for x = t+h·(.5-k), t+h·(.5+k).

    8.1 Set y = c·exp(-x*2/2).
    8.2 Set a = a+.5·h·y.

    Form 61:
      t       Integral
    Form 62:
    _·__      _·_____

    Set a=0.
    Set k=1/sqrt(12).
    Set h=.1.
    Set c=1/sqrt(2·3.14159265).

    Do part 6.
J:    t       Integral

     .00        .0000
     .50        .1915
    1.00        .3413
    1.50        .4332
    2.00        .4772
    2.50        .4938
    3.00        .4987
    3.50        .4998
    4.00        .5000
```

(3i)

numeric label as a prefix to a step is an implied command to JOSS to store the step in sequence according to the numerical value of the label. JOSS differs from other on-line systems by requiring the user to supply his own step numbers on all steps of his stored program. This seems appropriate, for the user always has the option of typing a direct command or an indirect step, without having to explicitly call for another mode to get the desired option. The numeric label determines whether an indirect step is an addition, an insertion, or a replacement for another step. The step numbers really do pay their way. Elsewhere, the language is very explicit. For example, it requires full words, in conjunction with numerical expressions, to denote steps, parts, or forms. This too contributes to readability. A step is limited to a single line, and a line is limited to a single step, neither being much of a constraint. As a result, a step number serves to identify not only the logical step but the stored string and typographical line as well. Arbitrarily complex expressions may be used everywhere, except as step label prefixes which must be explicit decimal numerals. The 52 upper- and lower-case letters are the only identifiers to which the user can assign numerical values. (If pressed, he can extend the set by indexing letters, but indexing is normally used in the customary way--to identify elements of vectors or matrices.) Again, generality of expression, single-letter identifiers, and two sets of groupers all contribute to readability. (For an experienced typist, readability implies writeability as well--text is easy, expressions take time but can be mastered, highly encoded implicit material is difficult.)

JOSS represents all numbers internally in scientific notation--nine decimal digits of significance and a base-ten scale factor with an integer exponent in the range -99 through +99. JOSS presents an exact input interface, familiar decimal arithmetic internally, and an exact output interface. Addition, subtraction, multiplication, division, and square root are carried out by JOSS to give true results rounded to nine significant decimal digits (except on overflow which yields an error message, or on underflow for which zero is substituted). The decimal nature of JOSS gives the user easy control over exact calculations that would require especially careful attention in a binary system.

The functions in the language include a set of logical functions which, together with the numerical relations and and and or, lead to powerful direct expressions of conditions which can be attached to any step. Care has been taken in a basic set of elementary functions to hit certain "magic" values on the nose and to provide reasonably full significance of results. The general exponential routine to compute a*b, for example, factors out error situations and the special cases of $b = 0$, $a = 0$, $b = 1$, b an integer and a an integer power of 10, $b = .5$, $b = -.5$, and b an integer with $2 \leq b \leq 29$, before resorting to $\exp[b \cdot \log(a)]$.

The interpretive technique on which JOSS is based enables the user to edit his stored program freely and quickly--even when JOSS interrupts at the user's request or suspends work on a task to report an error. Inserting and replacing steps or forms is implicit in the treatment of any new line of input. Deleting and typing are called for explicitly and the language provides "handles" at various

levels of aggregation so the user isn't forced to do his editing piecemeal at the level of individual steps, forms, and values. Steps are organized into parts according to the integer parts of the step numbers. Parts then become units that can be typed or deleted, as well as natural units for specifying procedures in hierarchical fashion. Values, too, may be organized into vectors and arrays if indexed letters are used, and the letters by themselves may be used to refer to entire arrays for purposes of typing or deleting. Still higher aggregates may be typed or deleted by using the expressions: all steps, all parts, all forms, all values, and all.

JOSS and the user take turns controlling the typewriter. It is critically important that the current status of JOSS with respect to any task it may have been working on be perfectly clear each time control is returned to the user. To this end, JOSS transmits error messages, interrupt messages, and stop messages to distinguish these states from the state of having just successfully completed a task. The user obviously does not want a message for successful completion, because it would be so frequent and because it would intrude on his formal output. Error messages are of two types: those that report violations of language constraints (such as indices not within the permissible range of integers from 0 to 99); and malformations of expressions, steps, etc. The first type is infrequent and the message is long enough to be very explicit about the violation. The second type covers a multitude of situations which are easy for the human eye to detect, but for which a precise error message is

extremely difficult. All these errors are reported by the very brief message "Eh?". Thus, the user is forced to read his step to find the error, rather than possibly being misled by a message unrelated to the actual error. In every error situation, the user is able to proceed. Frequently he can repair an erroneous step or form and continue with a Go command. Some errors may require that the user ask JOSS to start over after the repair, which is accomplished by simply giving JOSS the same Do command used to initiate the task. Even when JOSS has run out of space in pursuing the task, it stands ready to help. The user may ask JOSS to delete portions of the program which are no longer essential to getting final answers (such as forms or steps no longer needed) and to continue with a Go command, this time with additional space for JOSS to work in.

The user need do no preplanning in composing his procedures before sitting down at the JOSS console, since he can depend on interacting with JOSS to perfect his program. This ideal situation holds for the two users in RAND who have personal consoles. The other stations are public and, because of heavy usage, some users prefer to plan their work before going to the console--but it isn't necessary.

All input to the system is free form. It is unreasonable to demand that certain items of input be typed in specified columns on the page. On output, however, it is important that the user be able to require that JOSS type answers in conveniently specified forms. It is also important that JOSS choose a reasonable output form when the user hasn't specified one. JOSS' choice here is one number per line. Each number

is identified by the very expression used in the step calling for the output. JOSS tries to line up equals signs and decimal points, and uses fixed-point notation except when the magnitude of the number makes this unreasonable.

For formal output, the user has the entire width of the line in which to specify the literal information and the blank fields to be filled in with numeric answers. Just two types of fields prove adequate. A string of underscores with an optional decimal point indication is used for fixed point. A string of periods specifies a tabular form of scientific notation in which only the digit part and the corresponding exponent part are typed with the base ten understood. The number of digits typed is determined by the length of the field, and JOSS rounds the answers to fit the fields. Page and Line steps may be used to direct JOSS in formatting the output. As mentioned above, JOSS relieves the user of having to count output lines, by maintaining the line number on the page as the value of the dollar sign. The user can, for example, call for a new page at line 50 (i.e., "Page if $=50."). JOSS provides margins automatically, and identifies each page with time, date, and user's initials typed at the very top, where it can be clipped off if the page is to be incorporated into a report. The saving of time and errors by eliminating necessity for transcription of results is no small part of the system's attractiveness.

It should now be clear that the user always interacts with JOSS at a reasonable language level, never in machine language, and that the suggestion to think of JOSS as a computing aide is entirely appropriate. In fact, except

for machine malfunctions, no lower-level model of JOSS' activities can be used to explain behavior of the system which is not adequately explained in terms of the simple higher-level model. Thus, there are no JOSS system experts to call in for consultations--the checked-out user can explain every result even though he has no knowledge whatever of JOHNNIAC, the system routines comprising JOSS, or the representations of the entities of his program.

IMPLEMENTATION

The administration of the time-sharing aspects, drum slots, line buffers, states of remote consoles, selection of tasks, etc., is accomplished by detailed but straightforward machine-language routines in JOHNNIAC. The priority scheme never shuts out any user indefinitely. It responds quickly to input signals, such as carriage return, and keeps output-limited stations typing at full speed. The routines for interpretation, execution, and sequence control of the user's program, however, represent solutions to many new problems. The user's block of information initially contains certain tables, storage for value assignments (to 52 letters), heads of empty pushdown lists, working storage, and a list of available space units.

List processing routines are used to store away the literal strings of characters for steps and forms as list structures, and to perform inserts, replacements, and deletes on these structures. Similar routines are used to build list structures holding numerical representations of the elements of arrays, and to perform inserts, replacements, and deletes on them. A vector, then, is

represented by a list of elements, each labeled explicitly with its index. It need not be dense, since values are looked up by scanning the list for a match on the explicit index. Similarly, a matrix is represented by a list of lists of elements where the lists for the rows also carry explicit indices. Pushdown lists for operators and operands, as well as an auxiliary pushdown list, are used in the process of evaluating a numerical expression. The evaluation is programmed conveniently as a recursive routine. The most elaborate list structure arises in the bookkeeping JOSS must do to record the current step number at each level in a hierarchical task. Each Do causes JOSS to descend a level to perform the required part as a subroutine, then return and advance from the Do. If the Do step carries a for clause, then a list structure is built to record information essential to control of the iterations. This discussion is to point out that list processing is a cornerstone of the JOSS implementation.[7-9]

But, the list processing too is strictly an internal matter to JOSS and is completely sealed off. The user reaps the benefits in flexibility and interaction at a high language level. The challenge was not in the list processing as such, but rather in the clean-up and backing off to the beginning of a step to report an error to the user. It was essential that no irreversible change be made in the user's block until the interpretation of a step could guarantee that the execution would proceed to completion without an error. This consideration had to be modified slightly for the Do step with a for clause, but satisfactory stopping positions were found, even for the

case where the user deletes a part being iterated. This attention to detail has paid off. It is most satisfying to watch open-shop users with no previous computer experience, and little JOSS training, extract themselves from errors with JOSS' help, then continue with their problems.

CONCLUSION

JOSS now has more than 100 qualified users among staff members at The RAND Corporation. Their most frequent requests have been for more scheduled JOSS time, for more storage space, and for long-term storage of programs.* No one complains of the speed, although JOSS is slow. Everyone is enthusiastic about the simple language and ease of interaction. The distinguishing features of JOSS are: typewriters with an excellent touch and carefully selected keyboard; quick response to trivial requests; report-quality output; highly readable language; English capitalization and punctuation rules; exact input; familiar decimal arithmetic; exact output; no declarations; easy editing; powerful language for small numerical problems; and high-level language interaction at all times.

In this designer's view, the acceptance of an open-shop computing system depends on the little things--hundreds of them!

---

*In July of this year, JOSS service was extended into the evening hours. At this writing, a version of JOSS is being prepared to double the present user's block of 512 words and to record programs in punched cards.

ACKNOWLEDGMENTS

Many individuals have contributed to the building and installation of JOSS. The critical feedback and enthusiastic response of N. Z. Shapiro, A. C. Smith, and the other initial users of JOSS was invaluable.

REFERENCES

1. Baker, C. L., JOSS: Scenario of a Filmed Report, The RAND Corporation, RM-4162-PR, June 1964.

2. Boilen, S., E. Fredkin, J.C.R. Licklider, and J. McCarthy, "A Time-Sharing Debugging System for a Small Computer," AFIPS Conference Proceedings (1963 SJCC), v. 23, Spartan Books, Baltimore, Maryland, 1963, pp. 51-57.

3. Coffman, E. G., Jr., J. I. Schwartz, and C. Weissman, "A General-Purpose Time-Sharing System," AFIPS Conference Proceedings (1964 SJCC), v. 25, Spartan Books, Baltimore, Maryland, 1964, pp. 397-411.

4. Corbató, F. J., M. Merwin-Daggett, and R. C. Daley, "An Experimental Time-Sharing System," Proceedings 1962 Spring Joint Computer Conference, v. 21, The National Press, Palo Alto, California, 1962, pp. 335-344.

5. Dunn, T. M., and J. H. Morrissey, "Remote Computing--An Experimental System. Part I: External Specifications," AFIPS Conference Proceedings (1964 SJCC), v. 25, Spartan Books, Baltimore, Maryland, 1964, pp. 413-423.

6. McCarthy, J., "Time-Sharing Computer Systems," Management and the Computer of the Future, M.I.T. Press, Cambridge, and John Wiley & Sons, Inc., New York, 1962, pp. 221-236.

7. Bobrow, D. G., and B. Raphael, "A Comparison of List-Processing Computer Languages," Comm. ACM, 7 (1964) 231-240.

8. Newell, A., and J. C. Shaw, "Programming the Logic Theory Machine," Proceedings of the Western Joint Computer Conference (1957), Institute of Radio Engineers, New York, 1957, pp. 230-240.

9. Shaw, J. C., A. Newell, H. A. Simon, and T. O. Ellis, "A Command Structure for Complex Information Processing," Proceedings of the Western Joint Computer Conference (1958), American Institute of Electrical Engineers, New York, 1959, pp. 119-128.