ILA 13-FEB-75 09:53 31855

Meeting with DEC on 2-12-75

On 12 February, 1975 I met with representatives of the Digital Equipment Corporation, DEC, to discuss the PDP 11 system we envision using for NALCON. 1

Present at the meeting were Fritz Aumann and John Welsh of DECcomm in Maynard Mass. (617)897-5111, James Graves, Branch Manager Data Communications in the Washington area (301)459-7900 and Sam Lofthouse local DEC sales rep for NSRDC. 2

I presented our plans for NALCON and asked them for information on host interfaces for Univac, Burroughs, IBM and CDC equipment. I expressed interest in a cheaper CDC interface then we heard of before. They said there was a European made Univac interface, on which they will get me information. 3

I requested information on whether the DQ interface could be used for the VDH interface to the IMP rather then Bryans. They will check and get back to me. In the very near future DEC will be announcing the DV11 Synchronous Multiplexor. This will allow 8 or 16 lines with a total 16 line throughput of 38,400 characters per second. This has NPR input and output. There is a possibility that one of these could be configured for 50kb transmission (normally up to 9600 bps per line) by combining lines and still have sufficient capability for our synchronous requirements. 4

The highlights of the DV11 are 5

8- or 16-line synchronous multiplexor for use with PDP-11 family computers 5a

NPR Data Transfers on transmission and reception 5b

Total 16-line throughput of 38,400 characters per second 5c

Control table scheme provides considerable programming flexibility, particularly for special character and protocol handling 5d

Open ended, flexible design hardware not committed to any specific protocol 5e

128-character first-in, first-out receiver buffer 5f

Program selectable block checks (LRC-8, CRC-16, CRC/CCITT) 5g

Modem Control 5h

One or two synch characters for each line 5i

As a matter of passing DEC will be announcing the 11/70 today (2-13-75) which is a 32 bit member of the 11 family. This is intended as a high bandwidth system where fast disk access and throughput are required. 6

Meeting with DEC on 2-12-75

(J31855) 13-FEB-75 09:53;;;; Title: Author(s): I. Larry Avrunin/ILA; Distribution: /NETIMP([ACTION]) JPS([ACTION]) EHC([INFO-ONLY]) ; Sub-Collections: NIC NETIMP; Clerk: ILA;

JMB 13-FEB-75 19:58 31857

NSW promises to AFSDC (re--25298,)

I need some supporting info before I make the situation worse; thanks for warning me.

JMB 13-FEB-75 19:58 31857

NSW promises to AFSDC (re--25298,)

I have been giving Betty Finney's people training in NLS this week (another seession planned tomorrow-Friday), and they tried to corner me too for directions on how to structure their NLS files for AFmanual purposes; evidently the people being trained in NLS are going to be inputting documents too (? this is rather confusing as to who is going to do what)--in NLS. I felt just as flustered as EKM, especially when they told me about the army (as it seemed) of MTST typists blindly typing away. I talked with Finney and Bob Mortenson about the pros and cons of file structure methods relative to their manual's format ACCORDING TO THE FACILITIES OF NLS AT PRESENT. They expressed expectations that their NSW money would in the near future supply them custom designs to bypass the trade-offs in various file structure methods now existing. I guess I didn't quite realize the magnitude of potential danger in their following various directions either I or their expectations gave them, except to also gulp at the 25,000,000 characters. 1

NSW promises to AFSDC (re--25298,)

(J31857) 13-FEB-75 19:58;;;; Title: Author(s): Jeanne M. Beck/JMB; Distribution: /EKM([ACTION]) DVN([ACTION]) RWW([ACTION]) RLL([ACTION]) JCN([ACTION]) DCE([ACTION]) ; Sub-Collections: SRI-ARC; Clerk: JMB;

KM 14-FEB-75 05:56 31858

A visit in March

Jim, I will be visiting the West Coast in March and would like to take the opportunity to visit you and discuss details of the transfer of NLS/Tenex to the PDP/10 (which is being called our Network Management Facility) which we are setting up on our in-house network. Jess (Hill) and I tried to phone you yesterday but you were out. I am contacting you at this time because I have to include my itinerary in the trip justification which needs to be submitted now. Thus if you could tell me if it will be convenient to meet March 27th/28th, then we can give you further details of the subject matter later. We will try phoning you again later today. Thanks, Keith McCloghrie, Network Project Group, member of Network Management Facility Working Group, National Security Agency. 1

A visit in March

(J31858) 14-FEB-75 05:56;;;; Title: Author(s): Keith McCloghrie/KM; Distribution: /JCN([ACTION]) JHB([INFO-ONLY]) JNH([INFO-ONLY]) THP([INFO-ONLY]) KM([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: KM;

Section 1 INTRODUCTION

1.1 INVESTIGATION AND CONCLUSIONS

1.1.1 Purpose

The investigation and results presented in this report were accomplished to provide a system of subsets of the JOVIAL Standard Computer Programming Language (J73). This report presents the approach and rationale of subset selection, the application types and hardware each subset is intended for, and the syntax equations and semantics of each subset in order to enable Air Force installations to select the subset that best utilizes the resources of the procuring application and to establish the language specifications upon which acquisition of compilers for a subset can be based.

1.1.2 Approach

The investigation to develop a hierarchical system of subsets for JOVIAL (J73) was conducted as a benefit/cost trade-off analysis.

To establish the benefit value of JOVIAL (J73) features (i.e., language structures, computational structures, and the ways of realizing them), a relative utility value was identified for several hundred features within more than fifteen areas of application for JOVIAL. For example, the utility of having strings of logical values and of performing logical operations on those strings was rated (high, medium, low, very low) with respect to other features for use in realizing avionics, command and control, and data management systems. This utility of features was judged based upon (1) a survey of studies of the use of JOVIAL (J3) for programming in an application area, of comparative analysis among different programming languages for an application area or problem, and of the high-level language programming requirements for an application area; (2) the substantiation developed in considering JOVIAL (J3) for standardization by the American National Standards Institute; and (3) the deliberations held in defining and specifying JOVIAL (J73).

From this initial analysis certain features were judged to have high utility in all areas; others to have low or very low utility in all areas. For example, procedures and "item:declarations had overall high utility while "allocation:increment and tight tables had overall low utility. Features having overall high utility were relegated to basic (core) subset inclusion without further consideration of cost. Their constituent features were frequently not resolved at this stage, e.g., the types of variables declarable by an "item:declaration. Features having overall low and very low utility were relegated to exclusion from all subsets provided that an alternative means of realizing the computational structure a feature supported was available among features not excluded and/or the feature was primarily a convenience to the programmer rather than highly supportive in realizing a more efficient object program. For example, overlaying at an absolute address had no alternative for reaching machine registers, and "numeric:value:formula supports object program efficiency. There remained features of varied utility to be further analyzed.

The features not excluded or relegated to the base subset were then organized by high and medium utility for any application area into a composite that would suggest hierarchical subdivision along application boundaries, i.e., grouping hierarchical subsets by application utility. Many conflicts existed when considering only application utility.

Additional relative benefit and cost factors were judged for many features. They included compiler development complexity, computer storage requirement for compilation, processing time requirement for compilation, difficulty of learning and successful use by programmers as costs; and, object program time and memory space efficiency supported and programmer productivity improvement as benefits. Using these additional factors, the base subset was further expanded. For example, "environmental:specifier and "external:declaration were added to the base subset as their overall benefit was judged to be very high with respect to cost; multiple "variables as the left side of an "assignment:statement was added as object program efficiency is easily improved at practically no cost; and, "value:formula and "value:terminator were added as they improve object program efficiency in a unique way at relatively low cost.

The following major features remained in conflict to realize a hierarchy of subsets:

ordinary tables

specified tables

pointers

formatting

"concatenation

fixed values

floating values

attribute guidance

alternate entrance

"named:loop:control

"index:range

"status

recursion/reentrancy

direct code

parametrized define

round attribute for items

The analysis now considered direct trade-offs between features and cost/benefit influence of features included in the base subset on features not yet included. For example, pointers were included in the base subset as their incremental cost was low when implicit pointers for parameter handling were already included and their benefit in object program efficiency was high. Specified table was included as its cost was lower than an ordinary table, it covered all the capability and efficiency of a packed simple item (ordinary table does not), and it covered all the capability of a subordinate overlay (required to obtain similar capability for an ordinary table). On the other hand, direct code was excluded from all subsets as "external:declaration of a procedure (written for and compiled by an assembler) can provide the same capability, practically as efficiently, and at substantially less cost.

From this analysis, there evolved three subsets forming a hierarchical system in which each subset was judged to be suitable for certain application areas. At the same time, each subset took on a dominant characteristic. The base subset was most suitable for programming in a very explicit manner thereby providing greater control over the efficiency of the resultant object program. The middle level subset was most suitable for programming in a rather general fashion where the efficiency of the object program could yield in favor of increased productivity by possibly less skilled programmers. Of course, the middle level still permitted the more explicit programming since it contained the base level as a subset. The highest level subset was introduced to permit the programming of fixed type values.

Maintaining these characteristics as guidelines, the subsets were defined more completely. For example, concatenation of "character:formulas by the "concatenate operator was relegated to the middle level as a programmer could be expected to efficiently realize the same effect in the base subset using the "byte:string:function:call, "byte:functional:variable, and sometimes the "size:function:call which was introduced at this point. Following the same characteristics, features providing for programmer convenience were restricted in the base level and those with favorable cost/benefit factors were relegated to the middle level, e.g., factoring of "item:names.

In the final analysis, the three subsets were adjusted for consistency within each subset and to provide, still considering overall cost/benefit, that whatever computational capability could be realized in the full language could also be realized in every subset even though requiring a more arduous effort on the part of the programmer. For example, "index:range was introduced in the middle level for use in formatting and was extended for use as the left side of all "assignment:statements. The semantics of retrieving and storing in a specified table was more precisely defined to permit a system implementation to handle an unpacked item of any size as efficiently as would be possible if it were unpacked in an ordinary table. The semantics of explicitly pointed-to procedure data space was restricted to make recursion/reentrancy efficient at all subset levels.

1.1.3 Rationale of Subset Selection

Many computer systems and installations that perform computer program production cannot efficiently support the full Standard Computer Programming Language JOVIAL (J73). There has been a proliferation of subsets for previous versions of the JOVIAL programming Language, thereby hurting software transferability.

A system of subsets of JOVIAL (J73) has been developed to enable the use of a standard subset compiler by computer systems and programming installations that cannot efficiently support full JOVIAL (J73). The system of subsets is upward compatible within its own hierarchy and with full JOVIAL (J73) in order to retain a degree of inter-system compatibility (i.e., program transferability) and to reduce the cost of retraining programmers when transferred between facilities (i.e., programmer transferability).

The efficiency with which a computer system or programming installation can support JOVIAL (J73) or even one of its subsets involves various factors. Factors that can be influenced by the language itself are:

a. the size and complexity of the compiler required by the language.

b. the programmers' difficulty in learning and successfully using the language, and

c. the computer hardware features and capacity used in execution of programs declared in the language.

The size and complexity of the compiler impacts the utilization of the computer system resources on which compilation is performed. It also impacts the cost of obtaining a compiler. The compilers' size and complexity as well as the languages' size and complexity may impact the efficiency of programs compiled such that program production and operation subsequently impact the computer system resources they utilize. The programmers' difficulty in learning and successfully using the language impacts programmer productivity and computer system resources utilized for program production (recompilation, debugging, and program integration). The computer hardware features and capacity used in execution of programs impacts the effectiveness of utilizing computer system resources in program operation.

JOVIAL (J73) has been subsetted by eliminating features and syntactic forms that would be used infrequently in various application areas (e.g., avionics, logistics, command and control) or for which alternative language structures could realize the same effect. By reducing the size and complexity of the language to be compiled, compilation is more efficient. By reducing the size of the language to be learned and used successfully, programming, debugging and program integration are more efficient. On the other hand, features without an easy alternative needed by an application area, syntactic forms that permit the programmer to be concise, and features allowing more effective use of a computer systems capacity and power were retained. Finally, features and syntactic forms were combined in a subset for more than one application area in order to provide an upward compatible hierarchy of subsets while retaining a reasonable number of subsets.

An upward compatible hierarchy in a system of subsets and the full language must have for any two subsets any subset and/or the full language one as a proper subset of the other. A proper subset contains only syntactic structures and semantics that are present in the subset or full language of which it is a proper subset.

Three subsets were developed from standard JOVIAL (J73): JOVIAL, Level I(J73/I); JOVIAL, Level II (J73/II); and, JOVIAL, Level III (J73/III). Figure 1-1 depicts the hierarchical arrangement of the subsets and full JOVIAL. Level I (J73/I) is the base or core subset. Level II (J73/II) expands on Level I, and Level III (J73/III) expands on Level II. While each subset is intended for certain application areas, it does not follow that a higher level subset would be less efficient for certain computer systems and programming installations performing in a specific application area. Certain features not included in the intended subset may be appropriate more frequently than for the application area in general. Less precise utilization of the power and capacity of a specific ccmputer system may be efficient. The cost of more power and capacity may be more than offset by savings from increased programmer productivity and decreased program production activity that are realized by less precise programming of the computer system. This may hold for Level II as compared to Level I and full JOVIAL as compared to the subsets where certain of the added features are realized during computer execution of a program in a more general fashion than are specifically applied alternatives in lower Levels that have the same effect.

1.1.4 Features Excluded from All Subsets

The features of standard JOVIAL (J73) excluded from all subsets follow:

"chain:comparison

"concatenation in "bit:formulas

"bit:form

"character:form

"exrad:function:call

"significand:function:call

"fraction:part:function:call

"integer:part:function:call

"signed:function:call

"type:function:call

"number:of:words:per:entry:function:call

"alternate:entrance:function:call

"instruction:size:function:call

both sides of an "assignment:statement having multiple "variables and/or "formulas multiple ^loop:controls in a ^loop:statement

^exit:statement

^zap:statement

^remquo:procedure:call:statement

^direct:statement

list directed formatting

packed simple items

tight tables

^allocation:increment

^subordinate:overlays

dynamic allocation of procedure instructions

tables with variable bounds

procedure use before declaration

DEF and REF within a single program

^index:range in conjunction with ^number:of:entries:function:call

^actual:input:parameter forms STOP, RETURN, TEST, and EXIT

use of other type values in a character type context

^character:formula used as a ^format:list

rearrangement by ^index value of ^statements contained within a ^switch:statement

multiple ^names declared by a ^specified:table:item:declaration

^constant:formulas containing the operators &, @, and @@, ^floating: and ^fixed:constant operands and all ^intrinsic:function:calls except ^shift:function:call.

In addition, the ^copy:directive and all the ^directives for code optimization are system dependent in the subsets. The efficiency of having a library capability and/or code optimization capability varies among installations and computer systems. Therefore, the choice has been made system dependent. Most of these features were excluded because their usage in all application areas considered was judged to be infrequent, and, in general, alternative JOVIAL structures can provide the same effect or nearly the same effect with an acceptable loss in object program efficiency, increase in machine dependency, and/or decrease in data/procedure independence. Some of the features were excluded because they only provide programmer convenience which if absent would not impair unacceptably programmer productivity and computer system utilization for program production.

1.1.5 Application Areas Supported by Subsets

The system of subsets has been developed to support a number of application areas. Level I (J73/I) supports the production of avionics systems, executives and operating systems, communication systems, and other real-time control systems. Level II (J73/II) supports the production of logistics systems, data management systems, management information systems, simulation and planning systems, program production and utility systems, and off-line support systems for real-time on-line control systems. Level III (J73/III) supports the production of command and control systems and large scale tactical data systems.

The effect of certain features of standard JOVIAL (J73) that are introduced in Level II or Level III cannot be easily realized by alternative JOVIAL structures. They are "define;definition and "define:invocation with "parameters and the round attribute in an "item:declaration introduced in Level II; and, fixed type quantities ("variables, "constants, and intermediate results), "evaluation:control, and "attribute:association introduced in Level III. They have been judged to have an infrequent potential usage among many installations producing systems in an application area supported by a Level lower in the subset hierarchy. Some installations may have a more frequent potential usage for some or all of these features not included in the Level intended to support their application area. In such a case, a Level higher in the hierarchy will be more efficient for the installation.

Formatting, introduced in Level II, is similar to the situation just described although its effect can be realized by the structures of Level I and if more frequently used can be provided by library procedures. Otherwise, the features introduced in Level II (only the features mentioned above are introduced in Level III) are conveniences enhancing programmer productivity. Principally, these features are "alternate:entrance:declaration, "status:declaration, "ordinary:table:declaration, "exchange:statement, "indexed:variable:range, multiple names declared by a single "item:declaration and "concatenation in "character:formulas. Their effect can be realized by alternative JOVIAL structures included in

Level I. Their frequency of usage in producing systems in the application areas supported by Level I was judged generally to allow overall efficiency by programmers realizing their effect using alternative structures.

1.1.6 Computer Systems Effective for Subsets

The system of subsets has been developed to provide an effective JOVIAL language among computer systems (hardware) that differ in power and capacity.

From the standpoint of efficiency in compiling, systems having small storage capacity (32K bytes or less) slow execution speed and long accessing delays are best served by Level I. It contains the fewest JOVIAL structures and excludes a number of structures included in Level II and Level III which require considerable storage capacity, considerable execution time in compilation, and/or accessability to information that will be maintained in secondary storage by most compilers. Systems having less than 128K bytes but more than 32K bytes are better served by either Levels I, II or III than by the full language, standard JOVIAL (J73). Even with increased execution speeds and shorter accessability delays, the storage capacity of a computer system will dominate in determining the Level that best serves to perform compilation.

From the standpoint of system execution of JOVIAL programmed systems, Level I is most effective for those computer systems whose hardware resources are to be most efficiently utilized only for system execution while Level II is most effective where utilization of hardware resources is expendable in order to enable more efficient programmer prouctivity. Computer systems whose power and capacity are limited for the mission they are dedicated to are better served by Level I. Large scale computer systems used for batch processing and for many purposes are better served by Level II. Large scale computer systems used for a dedicated mission that requires a very complex and large software system is better served by Level II or Level III to obtain increased programmer productivity.

1.2 LANGUAGE DEFINITION (SUBSETS)

1.2.1 The Descriptive Metalanguage

The descriptive metalanguage used to specify JOVIAL subsets is the same as for the full JOVIAL language (J73). The classification schema and names of classes of JOVIAL structures in the subsets is also identical to those of JOVIAL (J73). The explanation of the descriptive metalanguage and the means by which the subset languages are specified within this report can be found in STANDARD COMPUTER PROGRAMMING LANGUAGE JOVIAL (J73) dated 1973 January 1 in Section 1.4, "The Descriptive Metalanguage"; section 1.5, "JOVIAL CHARACTERS, "Examples"; Section 1.6, "Notational Symbols, System-Dependent Values"; Section 1.7, "One-Dimensional Nature of a Program"; and, Section 1.8, "Syntax and Semantics -- Illegal, Undefined, Ungrammatical".

1.2.2 Correlation with JOVIAL (J73)

The language definition of each JOVIAL subset J73/I, J73/II, and J73/III parallels the language definition of JOVIAL (J73). In fact, their definition frequently requires that paragraphs of STANDARD COMPUTER PROGRAMMING LANGUAGE JOVIAL (J73) dated 1973 Janury 1 be taken as the subset language specification authority for semantics that are unchanged between the full language J73 and a subset and for interpreting the effect of a specific semantic change to the full language.

The syntax equations appearing in boxes in the language specifications of the subsets are each explicitly a syntax equation of that subset. The text of the language specifications of the subsets are of two types distinguished by paragraphs each headed by a reference identifier enclosed in square brackets (e.g., [J73:3.3.1]) and paragraphs not headed by a reference identifier. Paragraphs headed by a reference identifier are changes in the semantics of J73 to apply them as the semantics of the subset. The part of the identifier following the colon (e.g., 3.3.1) indicates the paragraph of the J73 specification that is changed. There may be more than one paragraph identifier separated by commas. In a paragraph headed by a reference identifier, sentences having the forms

... does not apply in Level ...

... cannot be realized in Level ...

indicate JOVIAL or computing structures unavailable in a subset and the referenced paragraph(s) of J73 must be interpreted in light of this fact. The reference may be to a section or even a chapter of J73 in which case the interpretation must be made for the entire section or chapter. Otherwise, a paragraph headed by a reference identifier discusses JOVIAL and/or computing structures. In this case, the referenced paragraph of J73 is completely replaced by the paragraph in the subset specification.

Paragraphs not headed by a reference identifier are paraphrases of the corresponding section of J73. In general, they provide the complete semantics of a subset. However, J73 is the authority, and it must be referenced for a more expansive interpretation or possible specification of a detail not elaborated in the subset. Also, J73 provides any examples.

To support the reference between the specifications of JOVIAL (J73) and the specifications of the subsets, both the syntax equations and text for each subset language are especially organized.

The complete syntactic description of each subset is given in the appendix of this report. The metalinguistic equations are in alphabetical order and numbered in the same manner as for JOVIAL (J73). The numbering has been maintained such that the same metalinguistic term is identified by an identical number in all subsets as well as JOVIAL (J73). As a result, gaps in the sequence of numbers indicate metalinguistic terms, structures of the language, that do not exist for that subset. Metalinguistic terms that exist for both the full language (J73) and a subset but which differ in their definition can only be detected by side-to-side comparison of the syntax equations.

Section 2 of this report presents the language specification for JOVIAL, Level I (J73/I); Section 3 presents the language specification for JOVIAL, Level II (J73/II); and, section 4 presents the language specification for JOVIAL, Level III (J73/III). Within the section for a subset (e.g., Section 2 for J73/I), the highest level sub-sectioning corresponds to the overall chapter organization of the JOVIAL (J73) language specification. For example, Sections 2.5, 3.5, and 4.5 each correspond to Chapter 5, "Statements. At the next level, correlation is still maintained. For example, Sections 2.5.7, 3.5.7, and 4.5.7 each correspond to J73's Section 5.7, "Conditional:Statement. Thus, by dropping the leading number of the subset section identifier you have the section identifier of the corresponding semantics in J73, e.g., 2.5.7, "Conditional:Statement. To reference from J73 to a subset, prefix the J73 chapter or section identifier with the section number of the subset, e.g., section 4.18, "Function:Call is specified for J73/II in Section 3.4.18 of this report. Similarly, the "Index-Glossary" of STANDARD COMPUTER PROGRAMMING LANGUAGE JOVIAL (J73) may be used with respect to the subsets. For example, "compool:directive has an index 11.2 which becomes 2.11.2 for J73/I, 3.11.2 for J73/II, ad 4.11.2 for J73/III. The index for syntax equations can be used directly since the same metalinguistic term is identified in all subsets by the number used in the full language J73.

Within the lowest level sectioning of the subsets' specifications, the text flows in parallel with the J73 section but without a specific counterpart for the paragraph numbers of J73. This flow is sometimes interrupted by a paragraph headed with a reference identifier as previously explained. However, the reference is always to a J73 paragraph(s) in the section that is the counterpart of the invoking section of the subset specification.

In total, both the J73 specification and a particular subsets' specification given in this report must be taken inclusively for a rigorous definition of the subset language. For other purposes, the presentation of the subset using only this report should be sufficient.

(J31859) 14-FEB-75 07:19;;;; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RJC([INFO-ONLY]) DLS([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RJC; Origin: < CARRIER, J73/I/1.NLS;1, >, 29-JAN-75 13:46 RJC ;;;; ####;

Chapter 2

(XXX)

2.1 Introduction

Level I (J3/I) has been developed to support the production of avionics systems, executives and operating systems, communication systems and other real-time control systems. Because Level I is the most restrictive subset, some installations supporting these application areas may use more effectively a subset higher in the hierarchy.

Level I has been developed to support the realization of programs for computer systems whose power and capacity are limited for their dedicated mission. Level I is inappropriate for computer systems whose power and capacity are more effectively utilized by supporting less efficient programs to realize increased programmer productivity.

Considering computer systems for hosting the compiler, Level I compilation can be accomplished on systems with smaller main memory capacity, slower execution speed and longer accessing delays than can compilation of the other levels. Storage capacity is the most dominant aspect of a computer system in considering the subsets compilable on it. Only Level I has a potential to be compiled on a 32K byte or less main memory storage. Level I can be effectively compiled on computer systems with larger than 32K byte main memory. The upper limit of main memory size that can be effectively utilized is predominantly a function of the program production facilities and optimization to be included in or supported by the compiler rather than the nature of the language.

In general, Level I provides the same efficiency in executing a program as can be realized in any other level of the full JOVIAL (J3) with the exception of rounding on assignment fixed type quantities ("variables, "constants, and intermediate results), "evaluation:control, and "attribute:association. However, it yields less efficiency in programmer productivity for applications in which the features excluded from Level I would be used extensively.

The principal features of Level II (J73/II) that are excluded from Level I follow:

"define:definition and "define:invocation with "parameters

round attribute in an "item:declaration

format-directed formatting

"alternate:entrance:declaration

"status:declaration

"ordinary:table:declaration

"exchange:statement

"indexed:variable:range

multiple names declared by a single "item:declaration

"concatenation in "character:formulas

The principal features of Level III (J73/III) that are excluded from Level I follow:

fixed type quantities ("variables, "constants, and intermediate results)

"evaluation:control

"attribute:association

The remaining principal features of full JOVIAL (J73) that are excluded from Level I are listed in section 1.1.4, Features Excluded from all Subsets.

2.2 ELEMENTS

2.2.1 Introduction

A ^program:declaration written in JOVIAL consists, basically, of ^statements and ^declarations. The ^statements specify the computations to be performed with arbitrarily named data. ^Data:declarations name and describe the data on which the program is to operate, including inputs, intermediate results, and final results. ^Processing:declarations specify computations performed when the ^processing:declaration is specifically invoked by ^name. In addition to ^statements and ^declarations, there are ^directives which designate externally defined ^names, control selective compilation of ^statements and ^declarations, and provide information the compiler needs in order to optimize the object code. The ^statements, ^declarations, and ^directives are composed of ^symbols, which are the words of the JOVIAL language. These ^symbols are in turn composed of the ^signs that constitute the JOVIAL alphabet.

2.2.2 Spaces and SPACES

It is important to distinguish between a ^space, an element of JOVIAL, and a space, an element of our descriptive language. JOVIAL is written using ^symbols, composed of ^signs. ^Symbols do not contain ^spaces, except in ^comments and ^character:constants. In general, ^symbols are separated by ^spaces. Only in defining and explaining ^signs and ^symbols are any spaces included in the metalanguage formulas not meant to be included in the definition.

2.2.3 ^Signs, Elements of the JOVIAL Alphabet

(equ)

In the box above, the metalinguistic term associated with each ^mark defines the term as the ^mark to the left.

2.2.4 ^Symbols, The Words of JOVIAL

(equ)

The ^symbols or words of the JOVIAL language are composed of strings of ^signs, in some cases a single ^sign.

2.2.5 ^Primitive, ^Ideogram, ^Directive:Key, ^Comment

(equ)

[J73:2.5.1] "Primitives may be considered the key words of the JOVIAL 73, Level I language. They are generally used to give the primary meaning of a "statement or "declaration, although some are used for secondary purposes.

All "primitives of the full JOVIAL language are "primitives in Level I. This maintains upward compatibility of "program:declaration across the levels (including full) of the JOVIAL language. "Ideograms are generally used as "arithmetic:operators, as "relational:operators, and for purposes such as grouping, separating, and terminating. "Directive:keys are used to state the primary meanings of "directives. "Comments can be used to annotate a "program:declaration.

"Spaces are permitted within a "comment, but a "quotation:mark and a "semicolon are not permitted within a "comment.

The "system:dependent:characters that can be included in "comments (and other structures) are simply those "characters, other than JOVIAL "signs, that the particular system and compiler can read and write. "Primitives, "ideograms, and "directive:keys do not contain spaces.

2.2.6 "Abbreviation, "Name

(equ)

"Abbreviations are specific "letters having specific meanings in specific contexts, usually "data:declarations.

[J73:2.6.3] A "name must not be the same as any "primitive. Since all "primitives of the full JOVIAL language are "primitives in Level I, a "name in Level I cannot be the same as any "primitive in Level II, Level III or the full JOVIAL language. This is necessary for upward compatibility of "program:declarations. Notice that a "name must include at least two "signs. The use of the "dollar:sign is system-dependent. That is, it provides a means whereby a "name can be designated to have some special meaning in relation to the system in which the compiler is embedded. Such special meanings are outside the scope of this manual, however, and "names containing "dollar:signs are considered the same as other "names herein. "Names do not contain "spaces. An embedded "space would change a "name into two "names or other "symbols.

2.2.7 ¯Number, ¯Constant, ¯Status

(equ)

A ¯number is a string of ¯numerals, without ¯spaces.

[J73:2.7.3] A ¯character:constant is a ¯symbol. Between the ¯primes, the string of ¯characters may include ¯spaces, but these ¯spaces are significant. They represent part of the value represented by the ¯character:constant. In a ¯status:constant and a ¯qualified:status:constant, ¯space is not permitted between V and the ¯right:parenthesis.

2.2.8 ¯Constants and Values

(equ)

¯Character!constants are the direct means of representing character values to be manipulated by a program. (¯Character!variables and ¯character!formulas are indirect means.) The ¯characters acceptable as character values are whatever the system will accept from among those given in the body of Figure 2-1. At least the 59 JOVIAL ¯signs must be accepted.

All of the character values indicated in the body of Figure 2-1 can be represented in ¯character:constants (except for system-dependent limitations). Any ¯spaces within the delimiting ¯primes represent characters of value "space". In order to represent a single occurrence of ¯prime, two of them are used in succession. If a succession of these ¯primes are desired as part of the value represented by a ¯character!constant, the entire string is doubled. In summary:

_2n ¯primes are used to represent _n ¯primes.

The system may impose a limit on the number of characters in strings representable by ¯character!constants, ¯character!variables or ¯character:formulas. The size of a ¯character:constant is the number of characters represented in the value -- not necessarily the number of ¯characters between the ¯primes because of the duplication rule for contained ¯primes.

¯Pattern:constants directly represent values consisting of strings of bits. The ¯numeral to the left of the _B in the ¯pattern!constant is the "order" of the ¯constant and controls the possible ¯pattern:digits and affects their meanings. The right column contains the possible orders. The ˉpattern:digits are displayed in the center in braces. The permissible ˉpattern:digits are only those on the line with or above the selected order. The meaning of each ˉpattern:digit is given in the column on the left, but these are also affected by the order. If the order is _n, then the _n rightmost bits of each pattern represent the meanings of the corresponding ˉpattern:digits. No ˉspaces are permitted anywhere within this structure.

The meaning of a ˉpattern:constant is the string of bits resulting from the concatenation of the strings of bits (as modified by the order) represented by each ˉpattern:digit. The size of the ˉpattern:constant is the number of bits in the string.

ˉNumeric:constants represent numeric values. ˉNumeric:constants are described in terms of their modes of representation; as integer values, and floating values. The compiler may represent ˉconstants in modes other than those indicated by the ˉprogram;declaration; as long as the overall effect of the ˉprogram:declaration is not compromised.

An integer value is a numeric value represented as a whole number. A ˉnumber used as an ˉinteger:constant represents an unsigned integer value. The size of an ˉinteger:constant is the number of bits needed to represent the value; from the leading one bit to the units position, inclusive (value zero has size 1). No ˉspaces are permitted in an ˉinteger:constant. The system may impose a limit on sizes of integer values.

Floating values (_v) are represented by three parts, the significand (_s), the radix (_r), and the exrad (_e). The value of a ˉfloating:constant is given as:

v = s x 10(e)

[J73:2.8.14] Inquiry into the values of significands and exrads cannot be realized in Level I.

[J73:2.8.16] A ˉfloating:constant must not contain any ˉspaces. In the syntactic equation for a ˉfloating;constant, the ˉnumber (or ˉnumbers) and the ˉdecimal:point (if present) give the value of the external significand. The ˉscale (with or without its ˉplus:sign or ˉminus:sign) following E gives an exrad (exponent of the radix) to be used as a power of ten multiplier. If the exrad is zero, it and the E can be omitted. To be a "floating:constant, the "symbol must contain a "decimal:point, or a "scale as exrad, or both.

The "scale following M gives the minimum number of magnitude bits in the significand of the internal representation. If the "scale following M is greater than the maximum number of magnitude bits in any of the system-dependent modes of representing floating values, the "floating:constant is in error. Otherwise, the compiler chooses the mode with the smallest number of magnitude bits in the significand at least as large as the "scale following M. If there is a choice of exrad size also, the compiler chooses one that can encompass the value of the "floating:constant. If the M and its following "scale are omitted, the compiler chooses its normal mode of floating representation or one that can contain the value.

[J73:2.8.18, 2.8.19, 2.8.20, 2.8.21] A "fixed:constant does not apply in Level I. A fixed value cannot be realized in Level I.

"Status:constants and "qualified "status:constants represent constant integer values. How they become associated with these values and how they may be used are explained elsewhere.

2.2.9 Computer Representation of "CONSTANTS and "VARIABLES

The structure discussed in this section is the system structure; the structure presented to the programmer by the combination of a particular computer and a particular JOVIAL compiler that produces object code for that computer.

A "byte" is a group of bits often used to represent one character of data. The number of bits in a byte is system dependent. Although JOVIAL permits some leeway in positioning bytes, there are usually preferred positions, referred to as "byte boundary."

A "word" is a system-dependent grouping of bits convenient for describing data allocation. Entries and tables are allocated in terms of words. Data are overlaid in terms of words. The maximum sizes of numeric values may, but need not, be related to words. Word boundaries usually correspond to some of the byte boundaries.

[J73:2.9.5] The "basic addressable unit" is the group of bits corresponding to each machine location. In many machines, the basic addressable unit is the word. In others, it is the byte. If it is the word, each value of the location counter refers to a unique word. If the basic addressable unit is the byte, each location value refers to a unique byte. Addresses may be restricted to certain locations for each type of value. For instance, double-precision floating values may be restricted to starting only in bytes with locations divisible by 8, for bytes, or 2, for words.

[J73:2.9.6] Integer values are represented in binary as strings of bits. The number of bits used to represent the magnitude of a value is known as its size. For signed values, the sign bit is an additional bit. The maximum permissible size of an integer value is system dependent. The maximum size of a signed integer is one less than this system-dependent size and the places where unsigned values of maximum size may be used are restricted; i.e., they must not be used in conjunction with any "arithmetic:operators, nor with the four nonsymmetric "relational:operators (_<, _>, _<=, _>=), and when used with the symmetric "relational:operators (_= and _<>) the other operand must not be signed.

[J73:2.9.7] The compiler determines the sizes of "constants. The programmer usually supplies the sizes of "variables. The size does not include the sign bit for signed data. For unpacked data not positioned by the programmer, there may be more bits in the space allocated for an item than are specified by the programmer. Whether or how these extra bits are used is system dependent, but in any case they are known as "filler bits". The sign bit, if there is one, and any filler bits are to the left of the magnitude bits. It depends on the system whether the sign bit is to the left or right of the filler bits. For unpacked or medium packed data positioned by the programmer (declared by a "specified:table:declaration) there may be more bits in the space accessable for the item than are specified by the programmer. Whether or how these extra bits are used in handling the item is system-dependent. If there is a sign bit, the programmmer considers it to be in the bit position declared by "bit:number followed contiguously by the magnitude bits. Whether the sign bit occupies this position or the position of the leftmost extra bit on its left, if there is one, is system-dependent.

The meanings of bit values _0 and _1 is not stipulated, but in most implementations _0 stands for _0 and _1 for _1 in positive values. For negative values, there is considerable variation.

Floating values are represented by two numbers, both signed. The significand contains the significant digits of the value and the exrad is the exponent of the understood radix. A system has one or more modes in which additional modes have more bits in the significand, the exrad, or both. The programmer can usually choose among the modes. In the absence of an indication of such choice, the compiler will use a standard mode, normally single precision. The radix is an implicit constant having a system-dependent value.

Character values are represented by strings of bytes, each byte consisting of a string of bits. The number of bits in a byte is system dependent. The number of bytes used to represent a character value is under control of the programmer, but there is a system-dependent maximum.

A character item that fits in one word is always stored in one word, by the compiler. By use of a "specified;table;declaration, the programmer may override this rule. A character item not densely packed always starts at a byte boundary. If it crosses a word boundary, a character item always starts at a byte boundary. The programmer must not attempt to override this rule.

An entry variable whose relevent "table:declaration does not describe it as being of some other type is a bit variable. It is merely the string of bits, of a size corresponding to the number of words in an entry, representing the entry.

2.2.10 "Spaces, "Comments

It is always permitted to place one or more "spaces between "symbols. At least one "space is required between "symbols when a single unintended "symbol would result where two "symbols were intended. "Comments can often replace required "spaces.

A "comment must not occur within a "definition nor within any "constant.

A "comment must not be used where the next structure required or permitted by the syntax is a "definition. That is, a "comment must not follow the "define:name or a "right:parenthesis in a "define:declaration and a "left:parenthesis or a "comma in a "definition:invocation.

(J31860) 14-FEB-75 07:25;;;; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RJC([INFO-ONLY]) DLS([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RJC; Origin: < CARRIER, J73/I/2.NLS;1, >, 30-JAN-75 07:41 RJC ;;;; ####;

Chapter 3

2.3 ~VARIABLES

2.3.1 Concept of ~Variables

A JOVIAL ~program:declaration consists of a string of ~statements and ~declarations that specify rules for performing computations with sets of data. The basic elements of data are items. The value of items and other data can be changed in various ways. A data element whose value can be changed by an ~assignment:statement is known as a variable.

(equ)

A ~variable is the designation, within a ~program:declaration, of a variable to be manipulated within the computer. The two syntax equations above indicate, first, the type of data involved, and second, the grammatical form of the ~variable.

2.3.2 ~Named:Variable

(equ)

A ~named:variable is a reference to a variable by means of a ~name associated with the variable through a ~data:declaration. A ~simple:variable is a reference to a variable not declared as a constituent of a table. A ~table:variable is a reference to a variable declared to be part of a table. A table consists of a collection of entries and there is an occurrence of each table item in each entry. An ~entry:variable is a reference to the entire entry as a single variable. An ~indexed:variable (a ~table:variable or ~entry:variable) includes an ~index to select the particular occurrence of the variable being referenced.

An ~index is correlated with the ~dimension:list in the ~table:declaration bearing the ~table:name or containing the ~item:declaration bearing the ~item:name. The ~dimension:list prescribes the number of dimensions and the extent in each of these dimensions by its ~lower:bound and ~upper:bound. Each ~index:component evaluated to an integer value selects within the extent in the corresponding position of the ~dimension:list.

2.3.3 ~Functional:Variable

(equ)

[J73:3.3.1] ^Format:variable does not apply in Level I.

The ^functional:variable beginning with BYTE is a ^character:variable whose size is determined by evaluation of the second ^numeric:formula. The ^character:variable is determined by selecting the number of bytes represented by the second ^numeric:formula from the ^named:character:variable beginning with the byte specified by evaluation of the first ^numeric:formula. Bytes are numbered from the left starting with zero. If the second ^numeric:formula is omitted, only the leftmost byte is selected.

The ^functional:variable beginning with BIT is a ^bit:variable whose size is determined by evaluation of the second ^numeric:formula. The ^bit:variable is determined by selecting the number of bits represented by the second ^numeric:formula from the ^named:variable beginning with the bit specified by evaluation of the first ^numeric:formula. Bits are numbered from the left starting with zero (beginning with the sign bit in signed variables). If the second ^numeric:formula is omitted, only the leftmost bit is selected.

2.3.4 ^Bit:Variable, ^Character:Variable

[J73:3.4] ^Format:variable does not apply in Level I.

A ^bit:variable denotes a string of bits without consideration of any numeric or other meaning associated with those bits. Almost all ^named:variables carry an implication of some data type other than "bit". However, an ^entry:variable, if the ^table:name is not declared so as to imply some specific data type, denotes only the string of bits constituting the entry.

The ^named:character:variable is a ^named:variable using a ^name declared to denote a variable (an item or an entry) of character type.

2.3.5 Numeric:Variable

[J73;3.5] Any ^numeric;variable can be used as a ^pointer:variable. The details of the use of ^pointer:variables are given in J73, Chapter 7 in conjunction with the discussion of controlled allocation. All ^names that can be used as ^named:variables are declared as explained in J73, Chapter 7. Some "entry:variables may use "names not associated with any data type. All other "named:variables use "names that are associated with "item:descriptions. These "item:descriptions give the data type among other things (see J73, Section 7.16 for details). One data type is "character" as mentioned above in J73, Section 3.4.2. Another data type is "floating". "Floating:variables use "names declared to be of floating type. The other descriptive terms in "item:descriptions denote "signed" and "unsigned". Signed and unsigned data are of integer value and the "named:variable denoting an item so described is an "integer:variable.

(J31861) 14-FEB-75 07:26;;;; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RJC([INFO-ONLY]) DLS([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RJC; Origin: < CARRIER, J73/I/3.NLS;1, >, 30-JAN-75 08:07 RJC ;;;; ####;

Chapter 4

2.4 "FORMULAS

2.4.1 Concept of "Formulas

"Formulas are the means for specifying the new values for "variables. "Formulas also generally supply values for any purpose--such as comparisons and other selections of courses of action. Since "constants and "variables denote values they are also "formulas.

(equ)

Any "numeric:formula can be used as a "pointer:formula.

2.4.2 "Constant:Formula

(equ)

A "constant:formula is a "formula whose value can be determined at compile time, once and for all. The values of all the elements must be known at compile time. A "constant:formula may be used anywhere that the syntax calls for a constant or number except as part of another "symbol.

[J73:4.2] "Constant:formulas are restricted to having only "integer:constant operands and only "shift:function:call among the "intrinsic:function:calls.

2.4.3 "Conditional:Formula

(equ)

A "conditional:formula is any "formula following the three "primitives _IF, _WHILE, or the "directive:key _!TRACE. After the "conditional:formula has been evaluated, the rightmost bit of the result is examined without further conversion. If that rightmost bit is _0 the "conditional:formula represents the logical predicate "false". If the rightmost bit is _1 the "conditional:formula represents the logical predicate "true".

2.4.4 "Character:Formula

"Character:constant and "character:variable are explained elsewhere. A "character:function:call is the invocation of a certain kind of "procedure:declaration having a character type implicit output parameter.

[J73:4.4] "Character:form does not apply in Level I.

(equ)

One of the "intrinsic:function:calls, the "byte:string:function:call is a "character:function:call. Any "character:formula represents a value having a size measured in bytes. The "byte:string:function:call derives from the "character:formula another "character:formula, whose size is the value of the second "numeric:formula. The derived "character:formula is extracted from the "character:formula given as the first "actual:input:parameter beginning with the byte specified by the first "numeric:formula. Bytes are numbered starting from the left with zero. If the second "numeric:formula is omitted, one byte is derived.

[J73:4.4.1, 4.4.3, 4.4.4, 4.4.5, 4.4.6] Concatenation (the "ampersand) does not apply in Level I.

[J73:4.4.6] "Bit:formula may not be used in the context of a "character:formula in Level I.

2.4.5 "Numeric:Formula

(equ)

"Numeric:constant and "numeric:variable are explained elsewhere. A "numeric:function:call is the invocation of a certain kind of "procedure:declaration having a numeric type implicit output parameter. Several of the "intrinsic:function:calls are "numeric:formulas (see Section 2.4.19).

A "bit:formula in a context requiring a "numeric:formula is treated as an unsigned integer value. The string of bits comprising the value of the "bit:formula is considered as the magnitude of a non-negative integer value. If its size is too great for the use to which it is being put, leading bits are truncated. If its size is unknown at compile time it is given a system-dependent default size unless its maximum possible size is known to be less than the default size; then, the maximum possible size is taken. If default size, the bits are right justified and any extra leading bits are zeros.

The only contexts requiring any ¯formula to be treated as a ¯numeric:formula are:

a. As an operand to participate in arithmetic.

b. As an ¯index:component.

c. As a ¯pointer:formula.

[J73:4.5.1, 4.5.2] Formatting and ¯numeric:format do not apply in Level I.

[J73:4.5.2] Attribute guidance does not apply in Level I.

2.4.6 Arithmetic

¯Arithmetic:operators specify arithmetic calculation in determining numeric values. The meanings of the ¯arithmetic:operators are:

_+	Add.
_-	Subtract. (or negate)
_*	Multiply.
_/	Divide.
_\	Determine the residue (modulo).
_**	Raise to the power of (exponentiation).

The ¯minus:sign as a unary operator means to negate the following ¯numeric:formula. The ¯plus:sign can be used as a unary operator, but it has no effect. The result of division by a zero value is undefined. The result of exponentiation of a negative base by a non-integer exponent is undefined.

Determination of a residue (modulo) is defined as follows:

x\y = x - y * trunc (x/y)

where trunc (v) is an integer whose sign is the same as v and whose value is the largest integer less than or equal to the absolute value of v.

For y = 0, x\y is undefined.

2.4.7 Default Scaling

The type (integer or floating) of a value denoted by a "numeric:formula depend on the attributes of its constituent "numeric:formulas and the arithmetic involved. The left-to-right rule and the precedence rules determine the order in which the values of two operands are combined--to form a single value to be an operand in another combination--or for assignment or other uses. If one of the two operands is floating, the operation is carried out in floating form and the result is floating type. Exponentiation is carried out in floating form and the result is floating type unless the base is an integer and the exponent is a positive "integer:constant.

[J73:4.7, 4.7.3, 4.7.4, 4.7.5, 4.7.7, 4.7.8, 4.7.9, 4.7.10, 4.7.11, 4.7.12, 4.7.13, 4.7.15, 4.7.16, 4.7.17] Fixed type operands and "numeric:formulas do not apply in Level I.

2.4.8 Uniform Rules of Calculation

"Formulas used in indexing and pointing are the same as the rules for all "formulas. When the value is finally used as if it were being assigned to an "integer:variable of the system-dependent size used for addresses. Certain arithmetic operations are carried out by explicit direction from the programmer--operations involved with such activities as calculation of addresses and the incrementing and testing of "control:variables. All intrinsic numeric quantities have system-dependent sizes. All calculations carried out by implicit directions comply with the default scaling rules for explicit calculations unless system-dependent documentation may make specific exceptions.

2.4.9 Attribute Guidance

[J73:4.9] Attribute guidance does not apply in Level I.

2.4.10 Scaling under "Evaluation:Control

[J73:4.10] "Evaluation:control does not apply in Level I.

2.4.11 Calculating, Rounding, Packing, Storing, Retrieving

When storing a value, items adjacent to the stored item, in adjacent words or in adjacent bits in the same word are protected (assuming the "packing:specification does not deny such care). When retrieving a value, bits in adjacent items, in adjacent words or in adjacent bits in the same word are not retrieved. All the bits stored for a value and only those bits are retrieved for the value.

[J73:4.11, 4.11.1, 4.11.2, 4.11.3] Rounding, scaling of fixed type operands and ¯description:attributes do not apply in Level I.

2.4.12 ¯Bit:Formula

(equ)

A ¯bit:formula is the representation of a string of bits, without regard to any meaning it might have as a numeric value or as a string of bytes. A ¯numeric:formula or a ¯character:formula in a context requiring a ¯bit:formula is treated as a bit string without regard to their numeric or character meaning. On computers having the attribute that the number of bytes per word does not fill the word, the unused bits are dropped when using a ¯character:formula as a ¯bit:formula.

¯Pattern:constant and ¯entry:variable are explained elsewhere. Two of the ¯intrinsic:function:calls are ¯bit:formulas. They are the ¯shift:function:call and the ¯bit:string:function:call.

[J73:4.12.1] ¯Bit:form does not apply in Level I.

(equ)

Any ¯formula represents a value consisting of a string of bits. The ¯bit:string:function:call derives from the value of the ¯formula a ¯bit:formula with the number of bits given by the second ¯numeric:formula. The bit string is extracted from the value of the ¯formula beginning with the bit specified by the first ¯numeric:formula. Bits are numbered starting from the left with zero. Bit zero of a character formula is the leftmost bit of the leftmost byte. Bit zero of signed values is the sign bit and the leftmost magnitude bit is bit one. Bit zero of an unsigned value is the leftmost magnitude bit. Floating values are system-dependent for the part of the floating form that occupies the leftmost bit. If the second ¯numeric:formula is omitted, a ¯bit:formula of 1 bit is derived.

[J73:4.12.2] Concatenation does not apply in Level I.

The ¯shift:function:call shifts the ¯bit:formula left if the ¯numeric:formula's value is positive and right if it is negative. Vacated bit positions are filled with zeros. The resultant ¯bit:formula has the same number of bits as the original ¯bit:formula.

[J73:4.12.5] The ¯signed:function:call does not apply in Level I.

2.4.13 ¯Comparisons and ¯Chain:Comparison

(equ)

A ¯comparison yields a ¯bit:formula one bit in size. A ¯comparison consists of a left operand, a ¯relational:operator, and a right operand. It has the value _1 if the left operand stands in the relationship stated by the ¯relational:operator with respect to the right operand. Otherwise, the ¯comparison has the value zero.

If both operands are ¯numeric:formulas, the truth or falsity of the ¯comparison is obtained by subtraction according to the rules of arithmetic between ¯numeric:formulas.

If one operand is a ¯bit:formula, the other operand becomes a ¯bit:formula. The truth or falsity is obtained by subtraction considering each to be an unsigned integer. If one ¯bit:formula is shorter than the other, the shorter is padded on the left with zero bits. If one operand is a ¯numeric:formula and the other is a ¯character:formula, they both become ¯bit:formulas. If both operands are ¯character:formulas, the truth or falsity is determined by considering each operand to be an unsigned integer. If one ¯character:formula is shorter, it is padded on the .B=1;right.B=0; with space characters to equalize the sizes.

[J73:4.13.4, 4.13.5, 4.13.6, 4.13.7, 4.13.8] ¯Chain:comparison does not apply in Level I.

2.4.14 Operations on ¯Bit:Formulas

(equ)

¯Bit:formulas represent strings of bits, each of value zero or _1. _NOT applied to a ¯bit:formula produces a derived ¯bit:formula in which each _1 in the value of the stated ¯bit:formula is replaced with zero and each zero is replaced with _1. The derived ¯bit:formula is the same size as the stated ¯bit:formula.

[J73:4.14.2] Concatenation does not apply in Level I.

The "logical:operators have their usual meaning, EQV meaning "equivalence" and XOR meaning "exclusive or". The "formula value with shorter bit string size is padded with zeros on the left to match the longer before performing the operation. The size of the resultant bit string is the same as the longer. Any "formula as an operand of a "logical:operator is treated as a bit string without conversion.

[J73:4.14.3] The maximum size bit string to which "logical:operators can be applied is system-dependent.

2.4.15 Precedence of Operations

The order of "formula evaluation is determined by operator precedence and grouping "parenthesis. In evaluating any "formula, operations are usually performed from left to right with the above in mind except where it is necessary to determine a value before a value can be set or obtained. These exceptions are:

a. The "formula on the right side of an "assignment:statement must be evaluated before evaluating any "index:components of the "variable being assigned and then any "pointer:formula needed to locate the "variable.

b. "Index:components must be evaluated before the "indexed:variable is evaluated.

c. The "pointer:formula must be evaluated before a pointed to "variable is located. Note that a "pointer:formula may be a pointed to "simple:integer:variable.

If a binary operator immediately precedes a unary operator, the unary operation takes effect first.

[J73:4.15.1] An "assignment:statement having a list of "formulas to the right of the "assignment:operator does not apply in Level I. "Exchange:statement does not apply in Level I.

[J73:4.15.2] The basic precedence of each operator is given in the list below.

0 = (assignment)

1 EQV XOR

2 OR

3 AND

4 NOT

5 = < > <= >= <>

7 + -

8 * / \

9 **

10 indexing @ (pointing)

[J73:4.15.3, 4.15.4] -Chain:comparison does not apply in Level I.

[J73:4.15.6] Figure 2-1 summarizes all conversions of data from one type to another possible in JOVIAL 73, Level I. Formulas or variables represented by XYZ, and of the five possible types as indicated at the top of the figure, are converted as indicated in the body of the figure under the influence of the operations and the types of the other operand (ABC) as shown at the left. To determine the conversion applying to both operands of a given operation, first consider one and then the other as XYZ. Whenever an operand of bit type is converted to integer ("Int") it is to unsigned integer. In some cases, a series of conversions (at least conceptually) is required. These are indicated by references to the following notes:

Note 1. In arithmetic operations with floating and character operands, the character string becomes a bit string, then an unsigned integer, then the integer is floated.

Note 2. In arithmetic operations with floating and bit operands, the bit string becomes an unsigned integer, which is then floated.

Note 3. In arithmetic operations a character string becomes a bit string, then an unsigned integer.

Note 4. In arithmetic operations a bit string becomes an unsigned integer.

Note 5. In comparing two character strings, the shorter is padded on the right with blanks. Then both are converted to bit strings and then to unsigned integers for the comparison.

Note 6. In comparing numeric with bit, character with bit, or numeric with character, the character is converted to bit type. Then both are converted to unsigned integer for comparison.

Note 7. A character string used for pointing or indexing is converted first to a bit string and then to an unsigned integer.

(equ)

2.4.16 Short-Circuit Evaluation

The order of evaluating "statements and "formulas in a "program:declaration is for effect only. As long as the computational results are the same, the compiled program may execute computations in a different order or computations may be omitted. The omission and rearrangement of computations are aspects of optimization. Formulas containing only one bit operands will be computed only until the result is known.

2.4.17 "Form

[J73:4.17] "Form does not apply in Level I.

2.4.18 "Function:Call

(equ)

"Intrinsic:function:calls are discussed in the next Section. Other "function:calls are very similar to "procedure:call:statements. The "procedure:name must be one whose "declaration associates an "item:description with the "name. This association of an "item:description makes the procedure a function, describes the implicit output parameter for the function, and establishes the "formula type and size for the "function:call. The matching and assignment of "actual:input:parameters with "formula:input:parameters is the same as with the "procedure:call:statement. The use of "actual:input:parameters in a "function:call is the same as their use in a "procedure:call:statement; except, if exit from a procedure is effected by a "go:to:statement referencing a "formal:input:parameter or an outer scope "statement:name, the function value is not returned.

[J73:4.18, 4.18.3] "Alternate:entrance:name and alternate entrance do not apply in Level I.

If the procedure corresponding to the "procedure:name is declared to be pointed to, the "function:call must include the "pointer provide a location for the data space of the procedure during this invocation.

2.4.19 "Intrinsic:Function:Call

(equ)

"Byte:string:function:call is a "character:formula. "Bit:string:function:call is a "bit:formula. "Shift:function:call is a "bit:formula. Their details were given previously.

[J73:4.19] "Format:function:call and "signed:function:call do not apply in Level I.

[J73:4.19.1, 4.19.2] "Alternate:entrance:function:call does not apply in Level I.

[J73:4.19.4] "Allocation:increment does not apply in Level I.

[J73:4.19.6, 4.19.7, 4.19.8] "Index:range does not apply in Level I.

(equ)

The "location:function:call is an unsigned "integer:formula of default size. Its value is possibly the sum of three elements:

a. The value of the "pointer:formula or "pointer:variable pointing to the structure (procedure instruction space, table, data block) containing the named entity or the compiler-assigned location of the structure.

b. The relative position of the named entity in its structure--item in entry, table in data block, "statement in procedure, etc.

c. Relative positioning due to the "index if present.

[J73;4,19,9] A table cannot be allocated space by submanifolds in Level I.

(equ)

[J73:4,19,11] The "absolute:function:call is a "numeric:formula of the same size and type as its "parameter, except that if the "parameter is not floating the function is unsigned. The value of the function is the absolute value of its "parameter.

[J73:4,19,12] "Words:per:entry:function:call does not apply in Level I.

[J73:4.19.13] "Exrad:function:call does not apply in Level I.

[J73:4,19,14] "Significand:function:call does not apply in Level I.

(equ)

The "sign:function:call is a signed "integer:formula one bit (besides the sign bit) in size. The value of the "sign:function call is zero if its "parameter is zero, +1 if its "parameter is greater than zero, and -1 if its "parameter is less than zero.

(equ)

The "size:function:call is an unsigned "integer:formula of default size. The value of the function is the number of units (bits, bytes, or words) in the "formula, "data:block, or "table.

[J73:4.19.16] "Fixed:formulas do not apply in Level I.

[J73:4.19.17, 4.19.18] "Type:function:call does not apply in Level I.

[J73:4,19,19] "Fraction:part:function:call does not apply in Level I.

[J73:4.19,20] "Integer:part:function:call does not apply in Level I.

[J73:4,19,21] "Instruction:size:function:call does not apply in Level I.

(equ)

The "data:size:function:call is an unsigned "integer:formula of default size. Its value is the number of words in the pointed-to data space of the cited procedure, if the "procedure:heading contains a "data:allocation:specifier.

2.4.20 Use and Qualification of "Status:Constants

Each "status:constant is given a constant integer value by means of its position in a "status:list (see section 2.7.17). Wherever the "status:constant is subsequently used (except in another "status:list) it represents that constant integer value. If a "status:constant appears in more than one "status:list, its meaning is resolved by context.

A "status:constant may be used to represent its value as the presetting "constant of, or in the "constant:list of, an "item:declaration (or "ordinary:table:heading or "specified:table:heading) containing an "item:description that contains or cites the "status:list in which the "status:constant is given its value.

A "status:constant may be used as the entire "numeric:formula providing the value to be assigned to an "integer:variable by means of a "simple:assignment:statement if the "item:description for that "integer:variable contains or cites the "status:list in which the "status:constant is given its value. A "status:constant may be used as the entire "actual:input:parameter corresponding to a "formal:input:parameter whose "item:description contains or cites the "status:list in which the "status:constant is given its value. A "status:constant may be used in the following context:

(equ)

In the above context, the "item:description associated with the "variable or the implied output parameter of the "function:call must contain or cite the "status:list in which the "status:constant is given its value.

In other contexts a "qualified:status:constant must be used. It may be used in the contexts described above. A "qualified:status:constant may be considered to consist of two parts -- the "name preceding the "status, and the "status:constant that remains when that "name and its following "colon are deleted. The meaning of the "qualified:status:constant is the same as the meaning of its corresponding "status;constant derived from the "status;list associated with its corresponding "name.

(J31862) 14-FEB-75 07:28;;;; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RJC([INFO-ONLY]) DLS([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RJC; Origin: < CARRIER, J73/I/4,NLS;2, >, 3-FEB-75 11:27 RJC ;;;; ####;

Chapter 5

2.5 STATEMENTS

2.5.1 Concept of "Statements

"Statements are the operational units of JOVIAL. They specify self-contained rules of computation, specifying manipulations of data, and/or conditional or unconditional sequencing of execution.

(equ)

A "controlled:statement is a required part of a "conditional:statement or "loop:statement.

(equ)

Any "statement may be used where a "controlled:statement is specified--except for the particular forms prohibited in the description of the "conditional:statement.

The kinds of "simple:statements are listed below.

(equ)

2.5.2 "Null:Statement

A "null:statement is used where no significant "statement is desired.

(equ)

2.5.3 "Compound:Statement

The "compound:statement provides for treating a group of "statements and/or "declarations as a single "statement.

(equ)

2.5.4 "Named:Statement

Any "statement can be named. A "statement:name is defined by attaching it to any "statement. Since a "named:statement is also a "statement, another (as many as desired) "statement:name can be attached. A "statement:name also may be attached to the first BEGIN in a "switch:statement.

[J73:5.4.4] "Exit:statement does not apply in Level I.

(equ)

2.5.5 ¯Assignment:Statements, ¯Exchange:Statement

(equ)

A ¯simple:assignment:statement specifies that the ¯formula to the right of the ¯equals:sign be evaluated and become the new value of the ¯variable to the left of the ¯equals:sign. The ¯formula is evaluated, then any ¯index or ¯pointer:formula associated with the ¯variable on the left is evaluated, and the value of the ¯formula is assigned to the ¯variable.

In the forms:

```
   _   BIT      _(¯formula_, ¯numeric:formula    _,
   ¯numeric:formula   _)

   _   BYTE
```

the leftmost ¯formula is evaluated first, then the second and then the third.

If the form on the left is:

```
      _BIT      _(¯named:variable_, ¯numeric:formula
   _,¯numeric:formula  _)

      _BYTE
```

any ¯index and ¯pointer:formula the ¯named:variable bears are evaluated first, then the second and third (if any) formulas.

Assignment of any type ¯formula to any type ¯variable is permitted. Conversions are performed as needed when the operands are of different numeric types. If the types seem incompatible, the ¯formula on the right becomes a ¯bit:formula and replaces the bits of the ¯variable on the left. If there are too many bits, leading bits are truncated; if there are too few bits, leading zeros are supplied before assignment. Note that this applies when assigning a ¯character:formula to any ¯variable not a ¯character:variable or a non-character to a ¯character:variable. When on a computer where filler bits are required to fill out a character word, these bits are dropped or inserted, respectively, in the conversion.

In assigning a "character:formula to a "character:variable, if the "formula is too long, excess bytes on the right are truncated; if too short, blank characters are added at the right to match the size of the "variable.

[J73:5.5.4, 5.5.5] "Indexed:variable:range does not apply in Level I. "Format:variable and "format:function:call do not apply in Level I.

[J73:5.5.5] Only a single formula (not a list of formulas) can be the value for assignment in Level I. The "assignment:statement differs from the "simple:assignment:statement in that more than one "variable receives the value of the "formula on the right. The formula on the right is evaluated first, then any "index or "pointer:formula for the leftmost "variable is evaluated and the leftmost "variable is assigned. Next, any "index or "pointer:formula for the second "variable on the left is evaluated and the formula is assigned to this "variable. This process continues until all "variables have been assigned.

The handling of BIT or BYTE, conversions, and type considerations are the same as for a "simple:assignment:statement. These considerations apply independently to each assignment.

[J73:5.5.6, 5.5.7, 5.5.8, 5.5.9] "Indexed:variable:range does not apply in Level I. Assignment from multiple "formulas does not apply in Level I.

[J73:5.5.10] The "exchange:statement does not apply in Level I.

2.5.6 "zap:Statement

[J73:5.6] The "zap;statement does not apply in Level I.

2.5.7 "Conditional:Statement

The "conditional:statement provides for the conditional operation of a "statement or "statements based on the value of a "conditional:formula.

(equ)

The "controlled:statement is any one "statement.

The value of the "conditional;formula is the rightmost bit of the evaluated "formula. If the value is 1, the first "controlled:statement is executed and program flow is continued with the "statement immediately following the "conditional:statement. If the value is 0, the first "controlled:statement is skipped and program flow is transferred to either the second "controlled:statement, if present, or the "statement immediately following the "conditional:statement. Exceptions occur within nested "conditional:statements. If a "controlled:statement preceded by ELSE immediately follows a "conditional:statement, then it is the "controlled:statement of an outer "conditional:statement and program flow is transferred to the "statement immediately following the later (outermost) "controlled:statement. This applies for any number of nested "conditional:statements. some nestings of "controlled:statements are not permitted. An embedded "conditional:statement omitting the ELSE "controlled:statement is not permitted as the first "controlled:statement of a "conditional:statement ending with ELSE "controlled:statement; except within a "compound:statement.

2.5.8 "Loop:Statement

The "loop:statement provides for the iteration of a "controlled:statement.

(equ)

The "controlled:statement is any "statement.

The "control:variable is assigned values for successive executions of the "loop:statement. The "loop:statement consists, then, of a means of specifying and controlling "control:variables and a "controlled:statement that is to be iteratively operated.

(equ)

The "while:clause form of the "loop:statement has no "control:variable; the "controlled:statement is executed repetitively until the "conditional:formula is false (the rightmost bit is zero).

(equ)

[J73:5.8.4] A "for:clause has only a single "loop:control in Level I, i.e., parallel control is not provided. The "control:clause associated with the "control:variable provides the successive values to be assigned to the ¯control:variable for successive executions of the ¯controlled:statement. The ¯control:variable is given a successor value for each execution of the ¯controlled:statement. Execution of the ¯loop:statement is terminated when the ¯controlled:statement causes a non-return jump out of the ¯loop:statement or when there is no successor value available.

The ¯initial:phrase, if present, must come first in the ¯control:clause and serves to provide an initial value for the ¯control:variable. There may be either a ¯replacement:phrase to specify the next value for the ¯control:variable or an ¯increment:phrase to specify the amount by which the ¯control:variable is to be modified on each iteration. A ¯terminator:phrase may contain the test by which the end of the iteration process is determined.

(equ)

¯Formulas in a ¯control:clause are normally reevaluated during each cycle. The effects of omitting various parts of the ¯control:clause are detailed in the following table.

(table)

The presence of a ¯terminator:phrase causes testing after the ¯control:variable gets its new value (if it does get a new one) and before the ¯controlled:statement is executed. The termination mentioned in the table applies to utilization of the ¯control:clauses. If the ¯primitive is WHILE and the ¯conditional:formula is 0 (false), the ¯loop:statement is terminated.

[J73:5.8.8, 5.8.9] A ¯for:clause has only a single ¯loop:control in Level I.

[J73:5.9.3, 5.9.4, 5.9.5, 5.9.6] Multiple ¯loop:controls in a ¯loop:statement, a parallel ¯loop:statement, do not apply in Level I.

2.5.11 ¯procedure:Call:Statement

(equ)

A ¯procedure:call:statement is used to invoke a procedure that is not a function.

[J73:5.11.1] ¯Remquo:procedure:call:statement does not apply in Level I.

[J73:5.11.2] ¯Actual:input:parameters must match the ¯formal:input:parameters associated with the named procedure in number, kind, and position in the list, and ¯actual:output:parameters must match the ¯formal:output:parameters in number and position in the list. The matching as to kind is that if the ¯formal:input:parameter is a ¯statement:name, the corresponding ¯actual:input:parameter must be a ¯statement:name. If the ¯formal:input:parameter is an ¯item:name, the corresponding ¯actual:input:parameter must be a ¯formula. If the ¯formal:input:parameter is a ¯table:name or a ¯data:block:name, the corresponding ¯actual:input:parameter must be a ¯data:block:name, a ¯table:name (with or without an ¯index), a ¯named:variable, a ¯constant or -@ followed by a ¯pointer:formula. If the ¯formal:input:parameter is a ¯procedure:name, the ¯actual:input:parameter must be the ¯name of a procedure with the same number, kind, and position of ¯formal:input:parameters and ¯formal:output:parameters as the ¯formal:input:parameter procedures.

In a procedure making a call on a procedure which is one of its ¯formal:input:parameters, the conversions between ¯actual:input:parameters and ¯formal:input:parameters are made in accordance with the descriptions of the ¯formal:input:parameters of the procedure used as a ¯formal:input:parameter. No cognizance is taken of the descriptions of the ¯formal:input:parameters of the procedure which is given as an ¯actual:input:parameter.

The order of evaluation of parameter data is left to right. The values of ¯actual:input:parameters are assigned to ¯formal:input:parameter:variables from left to right as if by an ¯assignment:statement. Upon exit, ¯formal:output:parameters are assigned to ¯actual:output:parameters from left to right as if by an ¯assignment:statement.

If the ¯formal:input:parameter is a ¯table:name or ¯data:block:name, a ¯table:name or ¯variable as an ¯actual:input:parameter means the location (of the variable or table). If the ¯formal:input:parameter is a ¯variable, a ¯table:name as an ¯actual:input:parameter means the value of the first ¯entry:variable, and any other ¯variable as an ¯actual:input:parameter just means its value.

If the procedure exits by means of a "go:to:statement referencing a "formal:input:parameter that is a "statement:name, the "actual:output:parameters are not set. The "go:to:statement is treated as if it referenced the corresponding "actual:input:parameter "statement:name; all intervening scopes will be deactivated.

A "go:to:statement referencing an outer "statement:name deactivates all procedures called from the scope of that outer "statement:name, and procedures called by those procedures, etc. It bypasses the setting of "actual:output:parameters of the procedure in which the "go:to:statement is executed and all other procedures deactivated.

[J73:5.11.11, 5.11.14, 5.11.15, 5.11.16] The "procedure:call:statement form including @ "pointer:formula is used when the procedure being called has pointed-to data space (see Section 3.8).

[J73:5.11.17, 5.11.18, 5.11.19] "Remquo:procedure:call:statement does not apply in Level I.

2.5.12 "Go:To:Statement, "Stop:Statement, "Return:Statement, "Exit:Statement

[J73:5.12] "Exit:statement does not apply in Level I.

(equ)

[J73:5.12.1] The "go:to:statement effects a transfer of control to the "statement bearing the referenced "statement:name.

The "stop:statement is the logical termination of execution of a program. Depending on the system, _STOP may cause a machine halt or a normal return to the executive.

The "return:statement is permitted only within a "procedure:body. Its effect is to terminate execution of the procedure, set the "actual:output:parameters from the "formal:output:parameters, and return control to the "statement following the call in whatever program invoked the procedure. The call might have been in any scope such as another procedure, the main program, or even the system executive.

[J73:5.12.4] If a "procedure:name is not referenced, the "statement means to return from the most local procedure.

[J73:5.12.5] Within nested procedures, the referenced ˉname in a ˉreturn:statement means to return from the procedure having the referenced ˉname. (If nested procedures use the same ˉname, it means return from the most local procedure, within which the ˉstatement appears, having the referenced name). If return is made to an outer procedure from within an inner procedure, the ˉactual:output:parameters are not set for the inner procedure.

Return to a procedure that is not active is undefined. "Active" means the procedure has been called but an explicit or implicit return from the procedure has not yet been executed. Such a return could only be attempted from a procedure declared with an external definition within another procedure.

[J73:5.12.7, 5.12.8, 5.12.9, 5.12.10] ˉExit:statement does not apply in Level I.

2.5.13 ˉSwitch:Statement

A ˉswitch:statement provides a multipath branch to other ˉstatements contained within it.

(equ)

[J73:5.13.1] Each ˉstatement between _BEGIN and _END in the above form is associated with an integer. In the absence of explicit bracketed ˉnumbers ahead of or between ˉstatements, the first ˉstatement is associated with zero, and successive ˉstatements (including ˉnull:statements) are associated with successive integers. (A ˉcompound:statement, ˉswitch:statement, ˉloop:statement, or ˉconditional:statement counts as a single ˉstatement.) Each bracketed ˉnumber, where present, interrupts the succession of associated integers and states a positive or negative integer value to be associated with the next ˉstatement. The succession then resumes following the stated value. There must be no repetition in values--each ˉstatement must be associated with a unique integer value.

In executing the ˉswitch:statement, the ˉnumeric:formula following _SWITCH is evaluated as an integer (truncated if necessary). Then the ˉstatement enclosed in the BEGIN END brackets and corresponding (as described above) with the values of the ˉformula is executed. If the ˉnumeric:formula does not yield a value corresponding to a ˉstatement in the list (including ˉnull:statements), the result is undefined. values skipped due to explicit ˉnumbers in the list do not correspond to ~statements. A ~statement in the list can be executed, if it bears a ~statement:name, by execution of a ~go:to:statement somewhere that references that ~name.

After execution of any ~statement in the list, if it does not permanently transfer execution elsewhere, the next ~statement in the list is executed, unless they are separated by a ~comma or a gap in the sequence of integer values, in which case the execution sequence is transferred to the ~statement following the END.

[J73:5.13.9] The ~exit:statement does not apply in Level I.

2.5.14 ~Direct:Statement

[J73:5.14] ~Direct:statement does not apply in Level I.

(J31863) 14-FEB-75 07:45;;;; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RJC([INFO-ONLY]) DLS([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RJC; Origin: < CARRIER, J73/I/5.NLS;1, >, 4-FEB-75 06:23 RJC ;;;; ####;

Chapter 6

2.6 FORMATTING

[J73:6] Formatting does not apply in Level I.

Chapter 7

2.7 "DECLARATIONS

2.7.1 Introduction

In this section, it will be seen how "names are associated with structures in JOVIAL and how definitions are provided for those structures via the various "declarations.

2.7.2 Undefined and Predefined "Names

Not all "names depend upon "declarations for their definition. "Names can be defined by their appearance in a "program:declaration and "names can be predefined.

A "statement:name is defined by its appearance at the beginning of a "statement. In a "procedure:declaration, a "formal:input:parameter can be a "statement:name.

"Names may be predefined for a "program:declaration. A reference to such a "name causes the compiler to seek its definition from a source external to the "program:declaration. "Item:names, "table:names, other "program:names, "procedure:names, "define:names, in fact, "names of any kind of JOVIAL entity can be predefined by means of a compool, a library, or both. The distinction between a compool and a library is arbitrary and beyond the scope of this language manual.

If a "program:declaration written in JOVIAL makes reference to a "name defined in the compool or library and if this reference is compatible with the compool or library definition, then the reference is taken to be a reference to the compool or library-defined "name. Any such referenced "name must be listed (either explicitly or by construction) in a "compool:directive at the beginning of the "program:declaration. If, however, the "program:declaration properly defines such a "name explicitly, then, if there is a conflict, this definition takes precedence and the compool or library definition is disregarded. "Proper" definition has reference to the necessity of placing "data:declarations ahead of any references to them. When a local "procedure:declaration is intended to override a compool or library "procedure:declaration, proper definition may require the local "declaration to precede any reference to the procedure or function.

2.7.3 Scope of Definition of "Names

The concept of scope determines the portion of the "program:declarations in which the declared "name is active. The scope of a "name is defined as that segment of code over which a "name has meaning. In JOVIAL there exists a hierarchy of scopes. Starting with the broadest, the scopes are named compool, external, main and procedure.

[J73:7.3.2] "Names declared in compools and libraries are of compool scope. (Local "names in library procedures remain local, of course.) They are available to all "program:declarations compiled under the influence of the compool or library. References in these "program:declarations to such "names are taken to be references to the associated structures provided the "names have been properly identified in the "compool:directive and are not masked by "declarations of identically spelled "names within the "program:declaration. All "compool:directives begin the "program:declaration and serve to establish a compool scope outside the "program:declaration in which the "names indicated in the "directive are assumed declared. This provides for overriding any "name declared in a compool at any level of source program scope; a "declaration is in effect for the local scope at which the "declaration occurs as well as any inner scopes not containing a declaration for the same "name. "Table:names, "item:names, "data:block:names, "statement:names, "procedure:names, "status:list:names and "define:names may all be declared in a compool.

External scope covers those entities (items, tables, data blocks, procedures, "statement:names) which while declared in a program are flagged in the "declaration as being common to more than one program by the presence of one of the "primitives _DEF or _REF.

Within a "program:declaration, scopes are defined by the "program:declaration itself and also by all "procedure:declarations within the "program:declaration. Data declared in the "program:declarations but not within any "procedure:declaration is of main scope. Data declared within a "procedure:declaration is of procedure scope.

Notice that the definition of procedure scope allows unlimited nested levels.

The above scope nomenclature is absolute. There is often a need for relative scope terminology. The relative terms local, outer, and inner allow scope to be discussed in relation to any particular point in a "program:declaration or system. Local scope refers to "names declared in the same scope as the reference point. Outer scope refers to "names declared in a more extensive scope than the scope of the reference point. Inner scope refers to "names declared within a more restricted scope within the scope of the reference point.

"Names may not be multiply declared in the same scope.

"Names can be repeated in different scopes. A "name comprises both a spelling and a scope; "names with the same spelling but different scope are distinguishable and not the same "name at all.

The scope of a "name local to a "procedure:declaration is the "procedure:declaration in which it is defined and all contained "procedure:declarations that do not have their own definitions of the "name. A main scope "name is defined wherever in the "program:declaration inner "procedure:declarations do not have local definitions of the same "name.

[J73:7.3.10] "Names explicitly defined within a "procedure:declaration are local to that particular "procedure:declaration. This includes all "formal:input:parameters and "formal:output:parameters. Conflicting local definitions within a particular "procedure:declaration are not allowed. A "procedure:name is of outer scope to the "procedure:declaration that it names.

"Names of local scope are not available in outer scopes. "Names of outer scope, however, are available in local or inner scopes provided the "names have not been redefined locally.

Resolution of a "name of outer scope used in a nested body of code begins from the current or local scope and works outward accepting the first "declaration encountered.

[J73:7.3.13] "Program:names, "procedure:names, "define:names, "status:list:names, "data:block:names, "item:names, and "table:names, but not necessarily "statement:names that are to be defined by "declaration must be declared before they are used in their respective scopes.

2.7.4 "Declarations

"Declarations are the principal means of naming and defining the various parts of a program.

(equ)

"Processing:declarations declare programs and procedures and are the subject of Section 2.8. "External;declarations cut across "data;declarations, the "name;declaration, and "processing;declarations and are discussed separately in Section 2.9.

2.7.5 "Null;Declaration

The "null;declaration is a means for satisfying a language requirement for the appearance of a "declaration even when no significant "declaration is desired.

(equ)

2.7.6 "Data;Declarations

"Data;declarations serve to declare and describe the data on which a program is to operate. The "names given to the data follow the "primitives that begin the "declarations.

(equ)

2.7.7 Fixed and Controlled Allocation

The data structures about to be described in the various "data;declarations are materialized in data space. Space must be provided to the program and the data structure must be associated with that space.

There are two basic ways of associating a data structure to space. The association can be made by the system, known as fixed allocation; or, the association can be made dynamically by the program during the operation of the object program, known as controlled allocation.

Fixed allocation is achieved by declaring the structure (either in the compool or the program) without indicating that controlled allocation is to be applied to the structure.

Data structures are identified as controlled allocation structures by the inclusion of an "allocation;specifier within the "data;declaration. Pointers are thereby established which locate the data structure.

Controlling the location of data structures within dynamic space is accomplished by assigning values to pointers. Each reference to a variable declared to have controlled allocation must employ, either explicitly or implicitly, an associated pointer. The value of the pointer will be used in the calculation of the effective address of the structure.

Dynamic association is independent of when space is obtained. The association of structure to space actually occurs when the value of the pointer to the space is appropriately established. Space can be received from the system or from some large block in the program's own environment, but attaching a structural definition (table, item, data block) to the space is performed during the program's execution by setting the value of the pointer to the structure to be equal to some address in the space.

2.7.8 Pointers and Their Association with Structures

Controlled allocation structures are also called pointed-to structures. This name expresses the requirement that there be a pointer that locates these structures as space is dynamically allocated them. The pointer is expressed as a "pointer:formula.

(equ)

"Pointer:formulas describe the address of pointed-to structures. The "pointer:formula may contain "pointer:variables as well as other "numeric:variables and "constants. The result of the evaluation of the "formula (truncated to an integer) is the pointer value used in referencing the structure.

If the "pointer:formula is not a "constant, it is evaluated at each reference to the pointed-to structure. If the "pointer:formula is anything more than a "constant or a "name, it must be enclosed in "parentheses.

A "pointer:variable is a special case of the more general "pointer:formula.

A "pointer:variable is a storage element which contains an address of some program element. Its "declaration and usage is as an unsigned integer item.

(equ)

The associations of pointer and controlled allocation data structures can be formed either in the "declaration of the structure or by explicit scripting at point of reference.

An association established in a "data:declaration is in effect unless an override (explicit association at reference) is encountered. At every reference, the established ¯pointer:formula is supplied automatically. The programmer need only script the ¯variable.

If no pointer is established in the ¯declaration of a pointed-to structure, then one must be supplied with each reference to the structure. Such explicit scripting of a pointer also serves to override a declared pointer for the current reference.

2.7.9 ¯Allocation:Specifier

The ¯allocation:specifier is an optional part of ¯simple:item:, ¯table:, and ¯data:block:declarations. Its appearance marks the data structure as a controlled allocation entity.

(equ)

In Level I, the ¯pointer:formula must be a ¯simple:item:name in an ¯allocation:specifier.

The -@ ¯ideogram is required to mark the data structure as pointed to and signify that it will receive dynamic allocation. The ¯pointer:formula provides the location of the structure and associates the data structure and the pointer such that the pointer is employed automatically with each reference to the data structure unless an explicit override occurs at some subsequent point. Absence of the ¯pointer:formula indicates that a pointer must be associated explicitly at subsequent points of reference. All ¯variables in a ¯pointer:formula must be declared prior to being referenced within a ¯data:declaration.

Wherever an implicitly pointed-to structure is referenced without an explicit ¯pointer:formula, the meanings of all the ¯names in its associated ¯pointer:formula are those in effect at the ¯declaration of the structure. If there is an explicit ¯pointer:formula associated with the structure ¯name at the point of reference, the current scopes of all the ¯names explicitly stated are in effect.

2.7.10 Data Permanence

Data that is allocated in a fixed manner (as opposed to controlled allocation data) may be considered to be either private or environmental to a given scope. Data that is private to a scope is available only while that scope is active. Data that is environmental to a scope is protected even while that scope is not active.

Data environmental to a scope is of greater permanence than data private to the scope. It is private to some outer scope and is protected while that outer scope is active.

Data that is environmental to the main scope is said to be "reserved" or "in reserve." All data declared in a "program:declaration automatically becomes reserved unless its permanence is restricted by being made private to some inner scope by the occurrence of an "environmental:specifier. Data that is to be initialized or preset must be in reserve. Reserved data are allocated when a program is loaded and remain until the program is unloaded even though the program is not active (executing). Data of compool scope that is not pointed to is environmental to all programs and in reserve. External data is environmental to all referencing programs as well as the program in which it is declared and is in reserve.

Private data exists and must be protected only when a procedure or program is active. Private data comes into existence when a scope is activated and disappears when the scope is left. Reentrance to the scope again activates the private data space; however, values left in the private data at the previous exit cannot be assumed valid.

Data generated by the compiler incidental to the processsing of forms other than "data:declarations become part of the unnamed data space of the procedure and is private to the procedure scope. This may include things such as temporary procedure space, register save areas, return address, perhaps some parameter space, and other linkage convention space.

[J73:7.10.4] Alternate entrance does not apply in Level I.

2.7.11 "Environmental:Specifier

The "environmental:specifier is an optional part of "data:declarations. It normally makes the data private to some scope. Without restrictions, the data being declared would become environmental to the program and in reserve.

(equ)

If IN is given, the data is made private to the local scope.

The entire environment of a procedure can be made private by so stipulating in the "procedure:declaration. An "environmental:specifier in a "data:declaration affects only the associated data structure. If a procedure's data space is made private, particular data can nevertheless be placed in reserve by an "environmental:specifier that incorporates _RESERVE.

A branch to an external "statement:name does not activate a scope. The scope must already be active through other action. A branch to an external "statement:name is considered an exit from the program and all private data is deactivated. Similarly a branch to an outer scope "statement:name deactivates all data private to scopes inner with respect to the target scope.

The "environmental:specifier and the "allocation:specifier must not appear in the same "declaraton. Controlled data are neither private nor reserved. The permanence of controlled data depends on both the permanence of their pointers and the execution of relevant "statements.

2.7.12 "Packing:Specifier

[J73:7.12] The "packing:specifier provides information used in the generation of code to access items or entries in the most efficient manner.

(equ)

[J73:7.12.1] _N indicates no packing, i.e., items do not share words.

[J73:7.12.2] _D means dense packing such that items are immediately adjacent to each other but not overlapping. Only character items or items whose size is longer than a word may cross word boundaries. A byte of a character item must not cross a word boundary.

[J73:7.12.3] _M specifies medium packing, i.e., items occupy without sharing machine dependent fields of a word.

[J73:7.12.4] "Packing:specifier in a "simple:item:declaration does not apply in Level I. In an "ordinary:table:item:declaration, the position of the item in an entry is affected. In the heading of an "ordinary:table:declaration, the "packing:specifier serves as default for "item:declarations that do not contain their own packing specification.

[J73:7.12.5] For specified tables, the "packing:specifier informs the compiler of any accessing convenience provided in the specified packing. Normally, the "packing:specifier should be at least as dense as the actual packing of the item. If it is not, code may result that can erroneously disturb surrounding items. The specified packing must agree with the meaning of the system dependent "packing:specifier.

2.7.13 "Constant:List

A "constant:list provides initial values for the items and entries of tables. The table must be declared in reserve.

[J73:.7.13] The extents of a table can only be given in terms of "constants in Level I.

(equ)

"Constant:lists consist of signed or unsigned "constants separated by "commas or indices. Successive "commas indicate null or undefined values. There may be indices (enclosed in brackets) among the elements of the list -- but not within any of the "parentheses in the list. Parts of the list may be enclosed in "parentheses preceded by a "count indicating that number of repetitions of the contents of the "parentheses, separated by "commas. A "constant:list:element following an "index must not be vacuous.

A fully expanded "constant:list may be thought of as consisting of "constants and nulls separated by "commas and "indices. If the "constant:list does not begin with an "index, initialization will proceed from the first element of the first row of the first plane, etc. "Indices are used to change the location from which initialization proceeds. All components of the "index must be "constants and must be compatible with the dimensionality of the table.

The order of values in a "constant:list is elements of a row, rows of a plane, planes of a volume, etc. An "index within the "constant:list resets the location at which values of the "constant:list are to be applied to the table. The reset location can be an increase or decrease from the current location. Whenever multiple initialization of the same bits is attempted the result is undefined.

If the bits of some items preset in a specific entry overlap, their initial values are undefined. If two "constant:lists each specify values for the same bits in a word, the initial values of those bits are undefined. The initial values of any bits not specified within a word and words or entries completely skipped over in presetting are undefined. Words partially preset by two or more overlaid items not in the same table, even though not interfering in terms of bits within the word, are undefined.

2.7.14 Data Structures

The simplest data "structure" is the bit string; all other structures are based on certain combinations of and interpretations of bit strings. The basic JOVIAL data structure is the item. An item is a bit string of specified length, with a specified interpretation.

[J73:7.14.1] The "intrinsic:functions INT and FRAC do not apply in Level I.

The larger JOVIAL data structures are created from items. They are the entry, table, and data block. An entry consists of one or more (possibly overlaid) items. Each item (known as a table item) is separately named and separately accessed; the entry may also be accessed as a unit. Entries are not declared as separate entities, but are associated with tables. A table is a (multiply-) indexed list or array of entries, all having the same structure. Each table is separately declared and named; the "declaration also names the items in the entry and may specify their location, possibly overlaid, within the entry. A table is referenced by its "name; normally a table entry is referenced by the "table:name and the "index of the entry; an item within an entry is referenced by the "item:name and the "index of the entry.

A simple item is simply an item that is not part of an entry.

A data block consists of one or more (possibly overlaid) data structures, simple items, tables, or other data blocks. The data block cannot be indexed and have its own "name. Data blocks may be accessed as a whole. They are permitted in only a very few operations; their primary use is for controlling data allocation.

2.7.15 "Item:Declarations

"Item:declarations name and describe both simple items and table items.

(equ)

2.7.16 "Item:Description

[J73:7.16] The "item:description is used in an "item:declaration to give the type and size of declared items. It may also be used in the heading of "table:declarations for similar purposes.

(equ)

[J73:7.16.1] "Abbreviations in the "item:description give the basic type of the item (or items) as follows:

_C	character
_F	floating
_S	signed
_U	unsigned

For character items the "size:specifier tells how many bytes in the item. If the "number is omitted, the default size is one byte.

[J73:7.16.3] For floating items the "significand:specifier, if present, gives a minimum size of the significand in bits--excluding the sign. The "exrad:specifier, if present, gives a minimum size based on a radix of 2, of exrad in bits--excluding the sign. If both "numbers are omitted, the default is the system-dependent single precision. If the system can provide alternative forms for floating values, it chooses one to accommodate the stated sizes of the significand and exrad. If it cannot do this, the "declaration is in error.

[J73:7.16.4] Signed and unsigned items are integer. The "size:specifier, if present, gives the size of the item in bits--excluding the sign for signed items. If this "number is omitted, the size is system-dependent--the size normally used by the compiler for addresses (at least for unsigned items).

Integer items (either signed or unsigned) may have "status:constants associated with some of their possible values by including a "status:list:name in the "item:description. The values given to each "status:constant must be compatible with the other specifications in the "item:description.

2.7.17 "Status:List and "Status:List:Declaration

A "status:list lists "status:constants each having a unique value. The "status:list can be declared in a "status:list:declaration for subsequent reference by "name in "item:descriptions occurring within the scope of the "status:list:declaration.

(equ)

Within a "status:list, each "number (and the sign if present) provides a value to be the meaning of the first "status:constant following it. Sequential "status:constants then take on sequential values, unless a new "number (and optional sign) sets a new value for the next "status:constants. There may be gaps in the sequence of values, but the sequence must be absolutely increasing. The "number immediately following the "name may be omitted -- giving a starting value of zero.

The "status:list:declaration associates a "status:list:name with the "status:list. The "status:list:name can be referenced within the "item:declaration thereby associating the "status:constants with the item. This allows several items to be defined in terms of the same "status:list.

A particular "status:constant may be in more than one list with more than one value, but it must not reoccur in a given "status:list.

2.7.18 "Simple:Item:Declarations

"Simple:item:declarations name and describe those items not associated with tables.

(equ)

[J73:7.18.1] Each "simple:item:declaration names one item. Space is normally allocated to each item independently as reserved data unless modified by the use of "independent:overlay:declarations, "environmental:specifier, or "allocation:specifier.

Use of an "environmental:specifier provides that storage for the item shall be environmental or private. Use of an "allocation:specifier causes controlled allocation for the "named:variables. If a "simple:item:name follows the _@, it is taken as the implicit pointer to the item declared. If there is no implicit pointer, every reference to each item requires an explicit "pointer:formula.

[J73:7.18.3] Only one item is declared by a "simple:item:declaration in Level I.

[J73:7.18.4] The "item:description gives the type and size of the item declared in this "declaration.

[J73:7.18.5, 7.18.6, 7.18.7] "Simple:item:declarations cannot contain a "packing:specifier or positioning information in Level I. No packing is employed for simple items.

[J73:7.18.8] A single "constant may be included in the "declaration of simple items that are in reserve. If the size of the "constant is larger than the stated size of the item, the "constant is truncated as for assignment to the item.

2.7.19 "Ordinary: and "Specified:Table:Item:Declarations

"Ordinary:table:item:declarations and "specified:table:item:declarations name and describe an item (or items) of ordinary and specified tables. The "declarations of the items occur within the bodies of the corresponding "table:declarations.

(equ)

These declarations name an item of the table corresponding to the containing "table:declaration.

If controlled allocation or unusual permanence is desired it must be accomplished within the "table:declaration or, if the table is contained in a data block within the "data:block:declaration.

The "item:description serves the same purpose as in the "simple:item:declaration. It gives the type, size and certain other information about the declared items.

In the "ordinary:table:item:declaration, the "packing:specifier directs the compiler as to how it should pack the item in an entry; either no packing, some degree of medium packing, or dense packing. If the "packing:specifier is not included, the "ordinary:table:heading provides the "packing:specifier to be used. In a "specified:table:item:declaration, the "packing:specifier tells the compiler how it must access this item. The default value is _D, dense packing.

The "bit:number gives the position of the first bit of the item in a word of an entry. Numbering of bits starts with zero on the left and counts to the right. The "word:number tells the word of the entry (starting with zero) in which the item resides or begins.

[J73:7.19.5] Tight structured tables do not apply in Level I.

[J73:7.19.6] Initial values may be provided for items by means of the "constant:list. There must be no attempt to preset (i.e., no "constant:list) items that are not in reserve. If the size of any "constant exceeds the size stated in the "item:description, the "constant is truncated as for assignment to the item. "Formal:parameter:tables must not be preset.

2.7.20 "Table:Declarations

[J73:7.20] Tables are collections of entries and the entries are themselves collections (possibly empty) of items. Tables have size and dimensionality; they are structured. They may be statically allocated at compile time or they may be dynamically allocated at run time. The "table:declaration provides the means to describe these various traits of a table.

(equ)

2.7.21 "Allocation:Increment

[J73:7.21] "Allocation:increment does not apply in Level I.

2.7.22 "Dimension:List

The "dimension:list of a "table:declaration provides the dimensionality of the table and the extent (size or number of entries) of the table in each dimension.

(equ)

[J73:7.22.1] The bounds are enclosed in "brackets. Absence of a "lower:bound (and a "colon) before an "upper:bound means the "lower:bound has the implied value zero. Each "lower:bound pesent and each "upper:bound must be a "number.

The "lower:bound (or the implied zero) and the "upper:bound in each position gives the range of values for an "index:component in the corresponding position. In subsequent references to "variables of this table, the corresponding component of the "index must be within this "range; an out-of-range "index:component is undefined.

The extent of the table in a particular dimension is "upper:bound + 1 - "lower:bound. The extent of the entire table is the product of the extent of each dimension.

[J73:7.22.5] "Allocation:increment does not apply in Level I.

2.7.23 Harmony of "Allocation:Increment, "Dimension:List, and "Allocation:Specifier

[J73:7.23] This section does not apply because "allocation:increment does not apply in Level I.

2.7.24 "Structure:Specifier

[J73:7.24] "Table:declarations employ the "structure:specifier to determine the basic structure of the table. The basic structure of a table refers to the arrangement of table entries in the words of a computer. Tables may have a serial or parallel structure.

(equ)

[J73:7.24.1] A _P indicates parallel structure. In its absence, the table structure is serial.

For serial and parallel tables there are one or more computer words for each entry. In a serial table, all the words for an entry are allocated contiguously in storage or memory. In a parallel table, the first words of all the entries are stored contiguously, followed by the second words of all the entries, etc.

[J73:24.2] "Allocation:increment does not apply in Level I.

[J73:7.24.3, 7.24.4] A tight structure table does not apply in Level I.

[J73:7.24.6] For any table, a parallel structure applies to all entries of the table.

[J73:7.24.7] A tight structure table does not apply in Level I.

2.7.25 "Ordinary:Table:Declaration

(equ)

An ordinary table is one for which the compiler determines the size and composition of an entry based upon the information in its "declaration.

2.7.26 "Ordinary:Table:Heading

The "ordinary:table:heading contains information necessary to describe and name an ordinary table.

(equ)

If the table is to be independently, dynamically allocated, the "allocation:specifier must be present. If the permanence of the table is to be restricted to private, the "environmental:specifier must be present. If neither an "allocation:specifier nor an "environmental:specifier is present, the table has fixed allocation and is in reserve. If the table is to be collected within a data block, there must be no "allocation:specifier or "environmental:specifier in the "ordinary:table:declaration. The desired effect must be achieved at the data block level via the "data:block:declaration.

[J73:26.3] "Allocation:increment does not apply in Level I.

The "dimension:list gives the number of dimensions of the table as well as the size of the table in each dimension.

The "structure:specifier marks the table as being of serial or parallel structure. Serial is the default structure; parallel is obtained by scripting P.

[J73:7.26.6] The "packing:specifier allows the table to be designated for dense, medium, or no packing. This occurrence of the "packing:specifier serves as a default value for "item:declarations of this table that do not contain their own. The overall default packing specification is "no packing".

An "item:description can be given to apply to the "table:name. Its effect is the same as if it were the description of a single item contained in the table. If the "item:description occurs the "ordinary:table:declaration omits the "ordinary:table:body, and the "table:name may be used as a "table:variable. If this "item:description is missing, such a reference means a reference to an "entry:variable of type "bit".

A "constant:list may be used to give initial values to some or all of the entries of a table in reserve. This "constant:list is independent of the occurrence of an "item:description in the "ordinary:table:heading. The programmer must avoid conflicts with "constant:lists in the "ordinary:table:body.

2.7.27 "Ordinary:Table:Body

The "ordinary:table:body follows the "ordinary:table:heading if that heading did not include an "item:description. The "ordinary:table:body lists those "item:declarations which make up an entry of the table.

(equ)

The "ordinary:table:body may consist of only a "null:declaration. The "null:declaration indicates that no named items exist for the table. An entry size of one word is assigned the table. The "ordinary:table:item:declaration was discussed in Section 2.7.19.

[J73:7.27.4] "Subordinate:overlay:declaration does not apply in Level I.

2.7.28 "Subordinate:Overlay:Declaration

[J73:7.28] "Subordinate:overlay:declaration does not apply in Level I.

2.7.29 "Specified:Table:Declaration

A "specified:table:declaration consists of a "specified:table:heading usually followed by a "specified:table:body.

(equ)

[J73:7.29.1] A specified table is one for which the entry is completely described by the user. Entry specification is performed by the contents of both the "specified:table:heading and the "specified:table:body.

2.7.30 "Specified:Table:Heading

The "specified:table:heading contains information necessary to name a specified table and describe its entry makeup.

(equ)

Every table must be named. The "name becomes a "table:name.

The "environmental:specifier, "allocation:specifier, "dimension:list, "structure:specifier, and "constant:list all function in the same way as in the "ordinary:table:heading.

[J73:30.2] "Allocation:increment does not apply in Level I.

[J73:7.30.3] The size of an entry of a specified table is described by a "number. This "number, the "words:per:entry, tells how many computer words are occupied by each entry.

[J73:7.30.4] Tight structured tables do not apply in Level I.

An item of the same "name as the table can be declared within the "specified:table:heading by "item:description, optional "packing:specifier, and location information. The various components declaring the item function in the same way as in the "specified:table:item:declaration. If an item is declared by the "specified:table:heading then the "table:declaration is judged complete and there can be no accompanying "specified:table:body. If the item is declared, then the "table:name can be used as a "table:variable; otherwise, such a reference means a reference to an "entry:variable.

2.7.31 "Specified:Table:Body

The "specified:table:body follows "specified:table:headings that do not declare items. It declares those items which make up an entry of the named specified table.

(equ)

The "specified:table:body can consist of a single "null:declaration, a single "specified:table:item:declaration, or several such collected as a compound "declaration. Regardless of positioning information in the "specified:table:body, the entry size is taken from the declared size in the "specified:table:heading.

[J73:7.31.1] "Subordinate:overlay:declaration does not apply in Level I.

(equ)

[J73:7.31.3] The "item:description gives the type, size and precision of the item. The "packing:specifier tells how the item is to be accessed and defaults to dense. The "bit:number and "word:number locate the item within an entry of the table. The "constant:list can provide initial values for the item if the table is in reserve.

The positioning information for an item can be incompatible with the entry size as specified in the "specified:table:heading. To what extent this can have any meaning is system dependent.

2.7.32 "Data:Block:Declaration

A data block is a convenient structure for grouping and allocating simple items, tables and other data blocks.

(equ)

The "environmental:specifier and "allocation:specifier function the same as in other "data:declarations. If neither is present, the data block is environmental to the program and in reserve. The "environmental:specifier can make the data block less permanent by declaring it to be private to the procedure. The "allocation:specifier can provide controlled allocation for the data block.

[J73:7.32.2] Controlled allocation and restricted permanence must be made collectively for the data block as a whole.

The items, tables and data blocks declared within this "data:block:declaration make up the associated data block. Unless otherwise directed by an "independent:overlay:declaration, the compiler allocates space in a manner it considers convenient and efficient. In all cases, allocation of all elements is fixed and contiguous. Reference to an element within a controlled allocation data block causes its relative position within the data block to be added to the value of the data block pointer effective at the point of reference.

"Independent:overlay:declarations within the "data:block:declaration arrange the named data structures of the data block. An "overlay:declaration within a "data:block:declaration is restricted to arranging elements declared within the data block.

2.7.33 "Independent:Overlay:Declaration

The "independent:overlay:declaration is used to specify the relative allocation of data within a data block or in fixed allocation storage or to specify the allocation of data to "absolute locations." The meaning of an "absolute location" is system dependent.

(equ)

The materialization in storage of data structures referenced by the "independent:overlay:declaration give rise to overlay structures known by similar words in English. Thus, we have overlay element for "independent:overlay:element, overlay string for "independent:overlay:string, and overlay expression for "independent:overlay:expression.

An overlay element has a length measured in words. An overlay element that is referenced as a "data:block:name, "table:name, or "simple:item:name has the length of the associated data structure. A "spacer defines an overlay structure that is pure length measured as the number of words equal to the value of the "spacer. There is no data structure associated with a "spacer. An overlay element designated as an "independent:overlay:expression in "parentheses has the same length as the associated overlay expression.

The overlay element specified following a "comma is allocated storage space immediately following that allocated the overlay element specified preceding the same "comma. For overlay elements specified by "spacers, space is left in the allocation process equal to the length expressed by the "spacer. An overlay string has length equal to the sum of the lengths of the overlay elements.

The overlay structures specified by two or more "independent:overlay:strings connected with "colons are allocated storage space so that they begin with the same word. Thus, the length of an overlay expression is the length of the longest overlay string it contains.

A compool datum mentioned in an "independent:overlay:declaration fixes the locations of all data mentioned in the same "independent:overlay:declaration. The optional "number or "pattern:constant in "brackets fixes (by whatever meaning the system ascribes to such a "number) the location of the first overlay element, and therefore of all overlay elements, of the "independent:overlay:expression. If any datum mentioned in an "independent:overlay:declaration is also mentioned in an earlier "independent:overlay:declaration, the previously allocated location fixes the location of all overlay elements of the current "independent:overlay:expression. It is the responsibility of the programmer to avoid contradictions due to the presence of the same datum in different "overlay:declarations.

The entire set of structures related by an "independent:overlay:statement, in effect, form a data block. None of the related data structures can be declared to be pointed-to.

All the structures of an effective data block must be at the same level of permanence; i.e., they must all be in reserve or all at the privacy level of a single procedure or program. If the effective data block is to be private to some scope, at least one of the overlay elements must be declared to be private to the selected scope. None may be declared to be pointed to. None may have a contradictory ¯environmental:specifier.

Each ¯independent:overlay:declaration must occur within a ¯program:declaration or ¯procedure:declaration such that all the data elements it mentions are local or outer, but never of an inner or disjoint scope. An ¯independent:overlay:declaration occurring in a ¯data:block:declaration is restricted to arranging data elements declared therein.

2.7.34 ¯Define:Declaration

The ¯define:declaration provides the means to manipulate the source program at compile time.

(equ)

Any reference to the ¯define:name beyond the terminating semicolon, within the scope of the ¯define:declaration, is known as a ¯definition:invocation and the ¯definition is substituted for the ¯define:name.

A ¯comment is not permitted between the ¯define:name and the ¯definition.

A single ¯quotation:mark is not permitted within a ¯definition; each ¯quotation:mark desired in the ¯definition should be scripted as two adjacent ¯quotation:marks, which upon ¯definition:invocation, will be substituted as one ¯quotation:mark.

The use of a single ¯exclamation:point within a ¯definition is restricted to the identification of a ¯formal:define:parameter. However, in a way similar to that for ¯quotation:marks, each desired ¯exclamation:point that does not identify a ¯formal:define:parameter should be scripted as two. At invocation, strings of ¯exclamation:points are halved as they are copied.

¯Define:names have scope. The rules are, as much as possible, the same as for all other declared ¯names. ¯Define:names are not permitted in a ¯procedure:declaration between the ¯primitive _PROC and the first terminating ¯semicolon. The ¯name:declaration is provided to allow an inner scope ¯statement:name to duplicate the spelling of an outer scope ¯define:name and still be treated as a ¯declaration of a new local ¯statement:name.

A "define:invocation will not be recognized where the "names being declared are given in a "data:declaration.

A "definition:invocation causes the "definition to be substituted for the "definition:invocation in the source code of the "program:declaration.

(equ)

[J73:7.34.11] "Definition means the same here as it does in a "define:declaration. The bracketing "quotation:marks are optional unless the "definition contains one or more "commas or "right:parentheses or the representation of a "quotation:mark; then, they are mandatory. If the bracketing "quotation:marks are omitted, the "definition consists of those characters beginning with the first non-blank character and ending at, but not including, the first "comma or "right:parenthesis. Exclamation points are treated as any other character in "actual:define:parameters.

A "definition:invocation may refer to "definitions existing external to the current program. All such external "define:names must be introduced to the program by the "compool:directive. A "define:name is not considered to be a "definition:invocation where it occurs as part of another "symbol such as a "comment, "character:constant or "name. A "definition:invocation may occur in another "definition even before its own "define:declaration; however, it is not recognized as a "definition:invocation except when the encompassing "definition is invoked and the substituted string of "symbols is examined as source code. Then the embedded "definition:invocation invokes its own "definition. Circular or recursive "definitions are not permitted.

When a "definition:invocation occurs, the associated "definition is substituted. If the "definition contains sequences of two or more "quotation:marks or two or more "exclamation:points in juxtaposition, one "mark is deleted from each pair of such "marks during substitution. A "definition:invocation must not be used to create "symbols by juxtaposing the "definition with the "symbols preceding or following the invoking "define:name.

Within a list of "formal:define:parameters, a "letter may occur only once. A correspondence is established between the "actual:define:parameters of a "definition:invocation and these "formal:define:parameters. Occurrence of the "leters as "parameters in the "definition must be preceded by an "exclamation:point to indicate where the corresponding "actual:define:parameters are to be inserted in the "definition.

Substitution of the "actual:define:parameter for the "formal:define:parameter neither removes nor adds significant "spaces that may lead or trail the "formal:define:parameter so that juxtaposition of other "signs in the "definition with the substituted "actual:define:parameter can create "symbols from the combination.

When a "define:declaration contains a list of "formal:define:parameters, it is invoked by using a "definition:invocation which includes "actual:define:parameters. The "definitions in the list of "actual:define:parameters are matched with the "letters in the "list of "formal:define:parameters. There must not be more "actual:define:parameters than there are "formal:define:parameters. "Commas must be used as needed to indicate missing "actual:define:parameters where the temporary "definition of the corresponding "formal:define:parameter is to be a null string. If the "define:declaration is parameterized, any corresponding "definition:invocation must include the "parentheses.

2.7.35 "Name:Declaration

[J73:7.35] The "name:declaration is a scoping mechanism. It provides the means of clarifying the intended scope for "procedure:names and "statement:names.

(equ)

The "name:declaration can occur in any scope. In the main scope, it occurs as a part of an "external:declaration to designate that "statement:names either referenced or declared in the current program are of external scope.

[J73:7.35.2] In an inner scope, the "name:declaration resolves any possible conflicts between local "statement:names and outer scope "define:names. Since the "declaration of local "statement:names are not introduced by defining "primitives and "statement:names can be referenced before they are declared, it is necessary to be able to make a distinction between this use of a "name and the invocation of an outer-scope "definition with a similarly spelled "define:name. To avoid such conflicts, "statement:names can be listed in a "name:declaration.

(J31864) 14-FEB-75 07:50;;;; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RJC([INFO-ONLY]) DLS([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RJC; Origin: < CARRIER, J73/I/7,NLS;2, >, 11-FEB-75 06:45 RJC ;;;;< CARRIER, J73/I/7,NLS;1 ####;

2.8 PROGRAMS, PROCEDURES, AND FUNCTIONS

2.8.1 Introduction

"Processing:declarations, declare programs, procedures, and functions containing both "declarations and "statements.

[J73:8.1] "Form:declaration does not apply in Level I.

2.8.2 "Processing:Declarations

Programs, procedures, and functions are established by a class of "declarations called "processing:declarations.

(equ)

"Processing:declarations are a major means of defining scope. A "program:declaration defines main scope and "procedure:declarations within "program:declarations define procedure scope.

[J73:8.2.2] "Alternate:entrance:declaration does not apply in Level I.

2.8.3 "Program:Declaration

"Program:declarations declare independent and dependent programs. A dependent program is generally compiled independently but intended to be utilized as a procedure to be executed when called by another program.

(equ)

If any names from a compool are referenced in the program, one or more "compool:directives must precede either of the introductory "primitives PROGRAM or PROC. Other "directives may also appear at this point.

The arbitrary string of "characters following the "program:name allows for the implementation-specific expression of any required system parameter type information.

2.8.4 "Procedure:Declaration

A "procedure:declaration is a closed body of code. A procedure is invoked by "name in explicit "procedure:call:statements and never operates as a result of normal sequential operation of "statements. Inherent in the nature of procedures is the transmittal of "parameters and the automatic return of control to the point following the invocation.

(equ)

[J73:8,4,1] The "procedure:heading names a procedure, determines the form for proper invocations of the procedure, and controls the allocation of data. The "procedure:body is made up of the "statements and "declarations which give rise to the instruction set of the procedure and its local environment.

[J73:8,4,2] The "procedure:declaration is used to declare both procedures and functions. The declaration of a procedure may specify "formal:output:parameters while that of a function must not. The "procedure:heading for a function, however, must contain an "item:description (with certain other specifications optional) which acts as the description of the implicit output parameter of the function. This implicit output parameter--the only output of a function--is referenced within the "procedure:declaration as a simple "variable of the same "name as the function.

2.8.5 "Procedure:Heading

[J73:8,5] The "procedure:heading names the procedure, optionally provides direction for controlling the allocation of data of the procedure, lists any "formal:input:parameters and "formal:output:parameters, and, in the case of the "declaration of a function, describes the implicit output parameter.

(equ)

[J73:8,5,1] "Instruction:allocation:specifier does not apply in Level I.

The occurrence of "parameters in a "procedure:heading is optional. "Formal:output:parameters are not allowed in the "declaration of a function. A particular "name can appear no more than once as a "formal:input:parameter and no more than once as a "formal:output:parameter. Only a given "simple:item:name can appear as both in the same "procedure:heading.

[J73:8,5,3] The "item:description following the parenthesized list of "formal:input:parameters, if any; acts as if it declared a simple item of the same "name as the function. The "item:description functions as it does within any "item:declaration to give the type and size of the implicit output parameter. Any preset value for the implicit output parameter follows the ^item;description. If preset, the implicit output parameter must be in reserve and the presetting is done just once for each loading of the entire load module.

[J73:8.5.4] An ^allocation:specifier just for the implicit output parameter does not apply in Level I.

If the ^procedure:heading lists any ^parameters, all must be declared within the ^procedure:body whether or not they are referenced (except for ^statement:names).

For ^formal:input:parameters that are ^simple:item:names there occurs, at invocation of the procedure or function, a transfer of the values of the corresponding ^actual:input:parameters to the ^formal:input:parameters as if by an ^assignment:statement. For ^table:names and ^data:block:names as ^formal:input:parameters the location of the data structure associated with the corresponding ^actual:input:parameter is transferred and not the value or contents of the structure; these formal parameters may not be overlayed, preset, pointed to, or declared with an ^environmental:specifier.

Reference to a ^formal:input;parameter that is a ^statement:name within a ^go:to:statement causes exit from the procedure or function. ^Actual:output:parameters are set. Control is transferred in accordance with the ^actual:input:parameter corresponding to the referenced ^formal:input:parameter.

A ^procedure:name which is a ^formal:input:parameter must be declared as a ^procedure:name within the ^procedure:body by a ^declaration consisting of a ^procedure:heading followed by a ^procedure:body containing only the ^declarations for its formal ^parameters.

When a ^formal:input:parameter that is a ^procedure:name is referenced in a ^procedure:call:statement or ^function:call (within the ^procedure:body) it is as if a ^procedure:call:statement or ^function:call were executed referencing the ^procedure:name presented as the corresponding ^actual:input:parameter.

^Formal:output:parameters are limited to ^simple:item:names. Upon exit from the procedure (functions do not have ^formal:output:parameters) the values of the ^formal:output:parameters are transferred to the corresponding ^actual:output:parameters as if by an ^assignment:statement.

2.8.6 Location of Data and Instructions

[J73:8.6] The "data:allocation:specifier and "environmental:specifier provide the means for controlling the allocation of data space.

The data declared within a "procedure:declaration are of procedure scope and the default permanence is environmental to the program and the space is in reserve. This space is efficient to access and provides for the saving of values between procedure invocations. The remaining space of a procedure, that is the data that is not in reserve or is not declared individually as pointed to, can be considered to be in an unnamed data block associated with the procedure. Also contained in this data block is the data generated by the compiler to support the procedure, such as dynamic parameter lists, save areas for register contents and return addresses, temporary cells, and unnamed pointers to name parameters.

Data may be declared individually in the block by placing an "environmental:specifier in the declaration. In addition, all local scope data, not otherwise declared in reserve or pointed to, may be placed collectively in this unnamed data block by the presence of the IN "environmental:specifier or "data:allocation:specifier in the containing "procedure:heading. The existence of the "data:allocation:specifier indicates that this data block is pointed to; its location is the value of the formula passed at the procedure call. In absence of an outer containing procedure with a "data:allocation:specifier, the presence of the IN "environmental:specifier in a "procedure:heading indicates that the procedures unnamed data block is to be allocated much as reserve data except that the space may be overlaid with other procedures data space. This overlaying is permissible only in so far as none of the procedures are invoked by any other directly or indirectly.

The unnamed data block of a procedure contained within a procedure declared with a "data:allocation:specifier is considered part of the outer procedures unnamed data block.

DSIZE of a pointed to procedure is the total length of the following data space:

* The named data of the pointed to procedure which is not declared in reserve or pointed to.

* Any inner scope named data declared as IN data.

* All unnamed data generated by the compiler for this procedure and any contained procedure.

A procedure may not be declared with a ^data:allocation:specifier if an outer containing procedure has been declared with a ^data:allocation:specifier. The example on the following page should clarify remaining questions regarding data permanence.

[J73:8.6.5] An ^environmental:specifier affecting only the implied output parameter of a function does not apply in Level I.

A ^procedure:declaration is made recursive or reentrant by declaring its data space to be pointed to and properly managing the pointers used at the ^procedure:declaration.

```
PROGRAM AA;  BEGIN

  ITEM A1 U;

  ITEM A2 @A1 U;

  ITEM A3 IN U;

  PROC BB;  BEGIN

    ITEM B1 U;

    ITEM B2 @B1 U;

    ITEM B3 IN U;

  END

  PROC CC IN;  BEGIN

    ITEM C1 RESERVE U;

    ITEM C2 @C1 U;

    ITEM C3 U;

  END

  PROC DD @;  BEGIN

    ITEM D1 RESERVE U;

    ITEM D2 @D1 U;

    ITEM D3 U;
    ITEM D4 IN U;
    PROC EE;  BEGIN
      ITEM E1 U;
      ITEM E2 @E1 U;
      ITEM E3 IN U;
    END
  END
END
```

In the above program, A1, B1, C1, D1, E1, and A3 are in reserve, A2, B2, C2, D2, and E2 are pointed to and have no allocated space. (B3 and the compiler generated space for BB) may overlay (C3 and CC's generated space) in reserve if BB and CC do not call each other, D3, D4, DD's generated space, E3, and EE's generated space are in an unnamed data block whose location is passed at the call to DD. DSIZE of DD is the total length of the space.

[J73:8.6.11] "Instruction:allocation:specifier does not apply in Level I.

2.8.7 "Procedure:Body

The "procedure:body contains the "statements and "declarations which determine the procedure and its environment.

(equ)

If the "procedure:body is to contain more than just a single "statement or "declaration, then the collected "statements and "declarations are delimited by the "primitives _BEGIN and _END. All "formal:input:parameters (except "statement:names) and "formal:output:parameters must be fully declared in the "procedure:body before they are first referenced; as must all local data.

The "procedure:name is local to the scope in which the "procedure:declaration occurs. It is outer to the "procedure:declaration.

Whereas the "name of a function is outer to its "declaration, the "name of the implicit output parameter of the function is local to the "procedure:declaration bearing the same "name, Inasmuch as functions can be called recursively, there is need to distinguish between a "function:call and a reference to the implicit output parameter. For a "function:call, the "parentheses that enclose the "actual:input:parameters--even empty "parentheses in case there are no "parameters--are required. The "name of the function (omitting "parentheses and "parameters) indicates a reference to the implicit output parameter in "statements and in "overlay:declarations.

2.8.8 Alternate Entrances

[J73:8.8] "Alternate:entrances do not apply in Level I.

2.8.9 "Form:Declaration

[J73:8.9] "Form:declaration does not apply in Level I.

(J31865) 14-FEB-75 07:58;;;; Title: Author(s): Roberta J. Carrier/RJC; Distribution: /RJC([INFO-ONLY]) DLS([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RJC; Origin: < CARRIER, J73/I/8,NLS;2, >, 13-FEB-75 07:12 RJC ;;;; ####;

Happiness is.....

The quick brown fox jumped over the lazy dog. 1

The quick brown fox jumped over the lazy dog. 2

Now is the time for all good men to come to the aid of their
party,one beef sirloin, two Idaho baked potatoe with sour cream,
salad with roquerfort dressing,, very dry Beefeaters martini. 3

Good morning America! It is very cold in Washington DC and the
surrounding suburbs (i.e. the metropolitan area) 4

one turtledoveand a partridge in a pear tree with yellow and pink
branchesoverlaid with purple and green colored spiderwebs that
havetightly clustered dew-drops hanging languidly from the pendulous
strands,the black and gold "princess" telephone jangled rauchously a
s Silvia shifted her solid-ivory cigarette holderto her left hand and
exhal ed tentatively as the blue-grey ash drifted toward the salmon
tinted plush carpet.Her post-maturely grey hair appeared to be
frosted in the manner which was popular with teeoage girls in the
summer of '62. Silvia's lythe body swayed as she rasped into the
hard black plastic mouthpiece with a tremoulous "Hello." 5

this has been a terribly frustrating day Valentine's day is no
picnic. How's by you? 6

Happiness is.....

(J31871) 14-FEB-75 12:25;;;; Title: Author(s): Robert E. Mortenson/RBTM; Distribution: /CAM2([ACTION]) JMB([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: RBTM; Origin: < MORTENSON, YESTERDAY.NLS;3, >, 13-FEB-75 12:51 RBTM ;;;;Silvia's post-maturely grey hair appeared to be frosted in themanner popular with teenage girls in the summer of 62.Silvia's lithe bod y swayed as she rasped into the hard black plastic mouthpiece with a tremoulous , "Hello".####;

Series 66

The evolution of Honeywell's large computer systems which begins with the H635 and the H645 processors is an important development which highligts the role and continued utilization of Honeywell hardware and software by current users such as WWMCCS. Currently all WWMCCS sites are utilizing H6000 systems with a WWMCCS Operating System that is basically a GCOS Operating System. The inability of the GCOS operating system to be responsive to the following user problems: 1

(a) Security 1a

(b) On-line interactiveness 1b

(c) Restart and Recovery 1c

(d) Communications and Networking 1d

has been a continuing problem of WWMCCS users in meeting their specified requirements. As a result users have resorted to building their own subexecutives. This has resulted in costly duplicative efforts each trying to achieve the same basic capabilities. If however these same users decide that the current GCOS operating system cannot ever meet their requirements a change in the operating system environment will be warrented. 2

Based on this choice WWMCCS users will face the following alternatives regarding a hardware and software system upgrade if they choose to remain with Honeywell software: 3

(1) Series 66 (GCOS IV) 3a

(b) Series 68 (MULTICS) 3b

(c) Virtual Mhine Monitor 3c

(d) Other 3d

The decision on which path to choose is most critical and requires an in-depth study. 4

The following chart traces the evolution of Honeywell's GCOS and MULTICS Operating System on HIS hardware. 5

6

7

8

9

The top line on this chart repsents the basic GCOS evolution from the H600 to the H6000 to the currently comtemplated Series 66. The basic difference between the 600 and 6000 are integrated circuits, speed and the extended instruction set. The GCOS operating system on the H6000 is basically the same as that on the H600. There are diffenences in machine macros and EIS instructions, but the basic philosophy of GCOS remains the same. WWMCCS GCOS which also operates on the H6000 retains the same GCOS philosophy as H6000 GCOS, but adds a security package, WWMCCS unique user provisions and the WWMCCS data management system WWDMS. 10

If we evaluate each of the above options the following assessments can be made. With regard to option (a), Honeywell's current philosophy regarding its commercial users is to insure that in any upgrade from the H6000 to the Series 66, complete upward software compatibility will be maintained. This insures that those programs which operate under GCOS III will also execute under GCOS IV.

Therefore if WWMCCS chooses to upgrade to a Series 66 all of its current software developed will not become obsolete. 11

TThe architecture of the Series 66 hardware and software is being designed to address currrent user problems with the GCOS III Operating System on the H6000. Most notably, the current design for GCOS IV on the Series 66 offers the following new GCOS features: 12

(a) Shared procedures 12a

(b) Distributed memory management 12b

(c) Hardware controlled access 12c

The following schedule indicates the current schedule for the evolution of GCOS IV: 13

14

15

Date	Series 66	H6000 Commercial	H6000 WWMCCS	16
	SR1	G	WW 6.0	17
	SR2	H	WW 7.0	18
	SR3(level 66 GCOS)	I(6000 GCOS)	WW 8.0(WWMCCS GCOS)	19
	4(internal release)			20
Jan 77	SR5			21

22

23

First Quater 1977 System Release (SR/5) on Series 66 will be the first GCOS release to begin to take advantage of the series 66 architecture. At this point in time the GCOS III on the H6000 and the GCOS IV on the Series 66 will not be compatible. 24

If WWMCCS chooses to upgrade to a level 66 processor at any time during this transition IS will no longer be in a position to operate WWMCCS software on-site. In addition IS will not be able to evaluate the level 66 architecture on-site prior to a WWMCCS upgrade for evaluation purposes. 25

Basically GCOS IV will operate in a static paging environment. As a result, a program must be completely loaded into core (although not necessarily contiguous) prior to execution. At this point in time there are no plans for a demand paging environment. 26

A segment description register will designate access rights for programs to individual pages. If a program tries to write into a read only area a hardware fault will occur. Program's will be loaded into hardware protected areas known as work spaces. These work spaces will serve a function smiliar to that of the MULTICS ring concept for achieving security. That is a program executing in work space 1 will not have access to procedures or data in work space 0. 27

Each of these segments loaded can be shared by more than one program if so desired. This will allow two programs to share a pure procedure or two programs access to common data. 28

As a result of the transition from GCOS III to GCOS IV, Honeywill is planning changes to both the FORTRAN and COBOL compilers to generate code which can execute on the series 66. There are no current plans to perform similiar changes to either the JOVIAL or ALGOL compilers. 29

A study of this architecture has been completed by a team of government-and contractor personnel under a study contract by JTSA and System Development Corporation (SDC). This study basically examined the Series 66 architecture and its functionality regarding security. The study basically recommended that the Series 66 architecture can provide a suitable base to achieve an effective level of protection. However the study also observed that the GCOS IV software design,, as currently proposed, does not demonstrate that it will provide an effective level of protection. 30

Our involvement with WWMCCS users and other H6000 GCOS users has given IS a good understanding of the GCOS operating system. ISIM has currently a number of efforts related to improvements of GCOS because of known problems in areas of: 31

(a) On-line interactiveness 31a

b) Data Management 31b

(c) Communications 31c

(d) Security 31d

Basically because the understanding of problems in these and other areas, in any redesign of the GCOS Operating System it would be fortunate if we could be in a position to influence any new software. Once the level 66 Software becomes available we will not be able to evaluate its effectiveness with regard to hardware utilization. 32

Therefore based on the results of the JTSA/SDC study and the results of a recent RADC trip to Honeywell in Phoenix, it is imperative that an in depth evalution of the Series 66 architecture is made. This evaluation must relate the capabilites required of this new hardware and software in meeting the user requirements of on-line interactiveness, restart and recovery,, distributed communication,, and security. This evaluation can be accomplished as a result of direct experimentation with programs that can operate in both architecture modes. An effort must be started which can integrate the current design goals of the vendor Honeywell with the complex command and control requirements. Such an effort will enable RADC to closely monitor evolutionary development of the Series 66 hardware and software. 33

In regard to option (b) which involves an upgrade to the MULTICS Operating System the analysis can be made. 34

If we go back to our original chart and look at the evolution of the H645 we find the basic difference between the H6000 and the H6180 is the virtual memory and ring hardware. The basic difference between the H6180 and the H6880 will be the addition of a cache memory and the utilization of Metallic Oxide Semiconductor (MOS) memory. Thus the upgrade from a H6180 system which will reside at RADC to a level 68 will not be difficult. 35

However the basic difference between a H6880 and a H6680 is that the H6880 will run H6000 GCOS but will not be able to run level 66 GCOS. Thus inder option (b), IS has and is continuing to evaluate the MULTICS Operating System software. 36

The next option (c) refers to a new technological concept known as a Virtual Machine Monitor. The basic idea of a VMM is to construct a software package which can execute on a given hardware processor

(Series 66 or Series 68) and basically extends an unlimited number of interfaces to that same processor. The concept allows for more than one copy of an operating system to execute simultaneously or two different operating systems executing concurrently. A number of unique advantages are immediately available when utilizing such a concept: 37

(1) The use of VMM notions as organizing principles for very reliable systems. 37a

(2) The use of VMM's as software tools to influence program development productctivity. 37b

(3) The use of VMM's in meeting Air Force real-time operating system requirements. 37c

(4) The utilization of VMM's as a technique for the development of modular operating systems. 37d

Series 66

(J31872) 14-FEB-75 13:05;;;; Title: Author(s): Ray A, Liuzzi/RAL; Distribution: /WFS([ACTION]) ; Sub-Collections: RADC; Clerk: RAL; Origin: < LIUZZI, GE/66.NLS;1, >, 14-FEB-75 11:09 RAL ;;;;####;

Tentative Agenda for KWAC Meeting

Well, here it is- one tentative agenda, slightly delayed, Maybe the first thing Tuesday morning, we can discuss it and make any desired modifications, Have a good trip all-see you Tuesday, Frank 1

AGENDA Tentative Agenda for KWAC Meeting 2

Mon: holiday 2a

Tue: 2b

A.M. Introductory remarks 2b1

Doug Engelbart, Jim Norton, etc. & KWAC'ers. 2b1a

P.M. SRI News 2b2

ARC Development 2b2a

The NSW and NLS: Implications for the NLS User Community 2b2a1

ARC Applications 2b2b

Discussion of support that Applications can and is planning to provide, 2b2b1

Plans for further expansion of the user community 2b2b2

ARC User Services and Applications Development 2b2c

Discussion of User Services and Applications Development 2b2c1

Training: review courses(?), explain the design and plan 2b2c2

User Programs operation and training 2b2c3

The amount of service to be provided 2b2c4

the difference between training and user assistance 2b2c5

The Feedback service operation and how much should architects intercept, and how direct inquiries to Feedback can be coordinated with the architects, 2b2c6

ARC Marketing 2b2d

Marketing definition, plans and strategy (RLL) 2b2d1

Analysis, needs and possibilities 2b2d2

Special ARC involvements, such as NSF (DCE), the DPCS (DVN) 2b2e

Wed: 2c

A.M. show-tell-share session 2c1

P.M. Continue the session; Possibly begin Workshops on Special Problems 2c2

Thu: 2d

A.M. Dialogue between DCE and Architects. This should be, in part, directed by the events of Wednesday. 2d1

P.M. Meeting other communities including Bill English (Xerox parc) and potential/actual users of NLS 2d2

Fri: 2e

A.M. Visit to Xerox PARC (if feasible) 2e1

P.M. Workshops on Special Problems, Hardware demo's (if feasible), Continuing dialogue with ARC personnel, training in DNLS for those who want it 2e2

TOPICS To be addressed during the week 3

Getting Acquainted 3a

New Sites: Who/Where are they? What are they doing? When/Why are they coming on? 3a1

Old Sites: Something like the new sites plus last 6 months in review 3a2

Newsletter Revisited: Put one out as "meeting notes" so we will publish at least one issue this time around 3a3

The Applications of NLS or Beyond Text Editing 3b

Talks of specialized use of NLS by some architects 3b1

Integrating NLS into local site operations. 3c

For those sites willing to share experiences, what does effective NLS use require in terms of people support, documentation support, hardware support (e.g., hi-quality printers, cassettes,..), etc. 3c1

Improving the Effectiveness of KWAC 3d

Given: The effect of KWAC on development of NLS is nearly ZERO. Pro's and Con's anyone? 3d1

Future NLS expansion & to what degree can KWAC affect this? 3d2

Potential system additions desired by KWAC collectively 3d3

Each of us are probably doing things of interest to others (e.g., writing user programs, coordinating the production of documents, etc.) Can we take another crack at establishing a better mechanism for information exchange? 3d4

There is also the old problem of establishing user idents and possibly getting them validated for one directory or another. And then, I imagine one must remove idents from time to time. Why can't the local Architect be responsible for this? 3d5

Improving the Effectiveness of Office-1 3e

How SRI-ARC might better improve communications with the community. 3e1

At any given time, one may encounter difficulty in determining what is at Office-1, what will be available , and what is going away. Suggestions ? 3e2

Documentation: status, needs, and plans 3e3

POLICY/MANAGEMENT/OPERATION of OFFICE-1 3f

Offquota login policy 3f1

usage statistics 3f2

Improved access to NLS (e.g., via a commercial net) 3f3

Is "terminal like" access enough? 3f3a

How do we get existing information into NLS? Out of NLS into existing systems? 3f3b

DEX over the Net (& how it compares to having very large buffers available in your TIP,ANTS,ELF or whatever) 3f3c

DEX through TIPS- Problems and Remedies 3f3d

Alternative pricing schemes; For example, plans for payment by usage (instead of slots) 3f4

Expansion of Office-1 facilities 3f5

For example, can a teleconferencing package be made available. 3f5a

Resolution of severe problems 3f6

ARPANET Problems and potential solutions 3f7

Office-2 3f8

Miscellaneous 3g

Equipment demo - local vendors and their wares. Have vendors bring in equipment for demo, particularly printers. 3g1

Specific Problems 3h

My own involve integrating NLS into our local site operations and using NLS to cooperatively produce documents (where the cooperators are geographically dispersed and have varying degrees of expertise in NLS), If you all will send me additional topics, I'll try and work up some sessions to discuss them in, 3h1

Tentative Agenda for KWAC Meeting

(J31873) 14-FEB-75 13:24;;;; Title: Author(s): Frank G, Brignoli/FGB; Distribution: /KWAC([ACTION]) FGB([INFO-ONLY]) ; Sub-Collections: NIC KWAC; Clerk: FGB;

DLS 14-FEB-75 18:27 31874

Procurement Snags

Ed and Mac...if anything comes up while I'm gone, this is the story as best I see it. Mac try to get a resolution of the ARPA 50% thing, since it affects other efforts from MAW & TFL.

Procurement Snags

I know you are overwhelmed with requests for help, attention, handholding, etc; but knowing your ability to handle a couple of hundered problems simultaneously, here are a few more for the stack. 1

I had a long talk with Col Kelly about the new Workshop Utility Contract. He seems hung up on a number of points. 1a

The ARPA order reads ...A contractual work statement consistent with... SRI proposal...with the following changes: "Provision of 10 ARPA slots for a period ending 30 June 1975." This apparently means to him that the government can't sign a contract for the full compliment of slots for a year. He feels that if they did, and then ARPA didn't come through in FY-76, then the gov't would be liable for the $200+K...ie SRI could sue the gov't or claim partial closing costs. (His hangup on ARPA is related to the necessity to avoid a D&F procedure...for items over 100K...by attaching himself to the ARPA blanket D&F) There is also some guidance from P. De Lorenso that ARPA must foot 50% of the bill, before their D&F applies. 1a1

I don't know what happened to the original idea of the "on-call" contract. This was essentially one where services to be delivered were spelled out in advance and prices for each were negotiated, but where the exact number of units of service and their delivery points could not be precisely predetermined, indeed where they would expect to fluctuate. 1a1a

His reluctance to write a year contract may stem from his feeling that the TYMSHARE effort has to be a subcontract. I tried to convince him that we didn't care (although we do) where you got the TENEX service, as long as the statement of work was met. I tried to argue that if SRI was "foolish" enough to go out on a limb and sign a year contract with TYMSHARE, then that was their business. We also would not want to restrict SRI to TYMSHARE as the only suppliers of TENEX CPU cycles...you guys might find a better deal somewhere else. I also wondered out loud how the government could "officially" bind TYMSHARE as a subcontractor to the main contract, since only a portion of the TYMSHARE services would be delivered under the government contract. 1a2

There was a brief mention by Kelly of the possiblility of contracting on a monthly rate, rather than by the year. 1a2a

This is just to let you know (as best I understand it) what is transpiring here. Kelly had information that Floyd would be visiting ARPA next week, and felt that Spencer some resolution of the problem would come out of that meeting. Meanwhile, I will continue on my end to retell the story to procurement etc. 1a3

Procurement Snags

Col Kelly mentioned that he still needs a brief statement for NSA addition to the initial Office-1 contract, AF30602-74-C-0076. 1b

I think you should also ask for a no cost extention to that contract (if you haven't already), say for 5 months (as long as it doesn't go over the FY boundry) to cover your tail and mine. We will not be able to start the JOVIAL mannual publication up again for 3-4 weeks. After all the crap I went through here to get the money, I'd hate to see it not happen....Think about this anyway. 1b1

Maybe you should also ask for a no cost extention to contract AF30602-72-C-0313, to cover you through Mar-Apr time when you machine goes away. I think it now runs out/ran out in Jan. This also gives you guys (Watson?) another grace period for the delivery of the final report. 1c

Procurement snags

(J31874) 14-FEB-75 18:27;;;; Title: Author(s): Duane L. Stone/DLS; Distribution: /JCN([ACTION]) ELF([INFO-ONLY]) JLM([INFO-ONLY]) ; Sub-Collections: RADC; Clerk: DLS;

unsuccessful attemt to use tab in NLS

Ron, 1

2

I am trying to use NLS to enter some tabular data on the contract status. Wish me luck! 3

Well, now that I have read what <help> has to say about TABS I am not very interested anymore. So I will go back to XED at ISIA. Besides, OFFICE-1 is going down in a little while and I don't want to get caught short. 4

Ron, I will send this to you via NLS SENDMAIL, but will CC myself vie "copy" etc, since I still don't know how to mail things to myself at isi from NLS. 5

See you, stef 6

unsuccessful attemt to use tab in NLS

(J31875) 16-FEB-75 22:45;;;; Title: Author(s): Stan M. Taylor/SMT; Distribution: /RPU([ACTION]) BRL([INFO-ONLY]) ; Sub-Collections: NIC BRL; Clerk: SMT;

'move message' tutorial ---smt

Jim, here is that text I promised your.

SMT 17-FEB-75 13:12 31876

'move message' tutorial ---smt

Dave 1

...I decided to give you 'the whole thing', so what follows if a sort of blow-by-blow account of what should be done with comments where I would think you need them... 1a

BASE C: Goto (subsystem) C: Programs OK: (cr) 2

PROG C: Load C: program T/[A]: message 2a

here you must type out the entire word - message then [cr] 2a1

Comment: 2b

...the system will then give you the following: 2b1

Loading User Program (NOTE: I have deliberately provided indentation to help... I hope???) 2b1a

Don't Execute via RUN PROGRAM Command 2b1b

Use GOTO SUBSYSTEM Command 2b1c

Subsystem MESSAGE Now Available (Attached) 2b1d

PROG C: Quit OK/C: [cr] 2c

BASE C: Goto (subsystem) C: message OK: [cr] 3

again you must type out the entire word - message 3a

...the system then will respond... 3b

MESS C: (NOW YOU ARE READY OT ENTER COMMAND TO THIS SUBSYSTEM) 4

MESS C: Move C: message (file) OK/T/T/[A]: FILENAME = t.t<esc>ext;1 [cr] (to follow) A: STATEMENTNUMBER, OR, STATEMENTNAME [cr] L:(It is usually a good idea to use a letter d here) [cr] 4a

...here you have entered the 'move' with keystroke m, the system moves to the next C: and you type letter m and the system moves to the [A]: .... where you enter just the name of the file, i.e., t.txt;1, or something like that -- then hit the [cr key] and the system moves to the next [A]: ......, where you enter the statement number below which you want the stuff ---if a new file, use 0 . 4a1

...I'll leave You with the HELP system to work out what a STATEMENTNAME is, you really do not need it now, 4a1a

...the system now 'mutters' to itself, does a few line-feeds, and is done...., there is no overt indication that anything has happened,,,,,,, if you are unsure, you could quit the 'mess c:' subsystem, then do a print statement 1, 4a2

MESS C: <>Sort C: Message (Plex at) A: (here you enter the same address you used above, and the messages will be sorted according to date,,,,at least I think they will,,,NOTE: the address must include the specification of the 'level' ---for instance if you find them at statement number 1, the address must be --- 1 ,d [cr] 4b

MESS C: Quit OK/C: [cr] 4c

BASE C: Update C: File OK/C: [cr] 5

...this last command, is just insurance to make sure the file you have been filling if 'for real' and wont get lost in a system crash or some such,,, 5a

BASE C: Print C: Branch (at) A: (same address used above, including the <sp>,d [cr] V: (Use viewspec xmb, and you will get a truncated listing of the sorted file) 6

and ....finally, if you have several such files to move, I would not do the sort command untill al had been moved, then follow the sequence through from there on 7

I hope all this 'garbage' is of some help,,, Stan 8 'move message' tutorial ---smt

(J31876) 17-FEB-75 13:12;;;; Title: Author(s): Stan M. Taylor/SMT; Distribution: /JHB([ACTION]) SMT([ACTION]) ; Sub-Collections: NIC; Clerk: SMT; Origin: < TAYLOR, MSG-MVR-DLG.NLS;3, >, 11-FEB-75 16:54 SMT ;;;;####;

To Anyone in PSO who works with Form 2's: This is an alternative to the present way of putting form 2 information into NLS. There are, naturally, many ways of doing this. I believe that this new ay may be more effective. If you find it easier or faster, you may wish to use it (or invent your own way).

The current procedure for inputting the Form 2 data is to set up a half-blank file that contains the reporting period for that month. The screen is then split and the job-order-number & SSN are copied from a separate file. The numers of hours charged against this job by this individual is then inserted. All of this involves 3 commands per line entry, not counting setting up the original file in the first place. 1

I would like to suggest an alternative way by having the previous month be copied into a new file (or simply make a new version of it), and then substitute the new reporting period for the old one over the entire file. It is my contention that most people use basically the same job-order-numbers that they needed the month before. In that case you only need one command, replace number, to update the hours that were charged from those of the previous month to the current values for the month you are inputting. If a particular job-order-number was not used, simply delete that statement. If a new one is added, copy any other entry from that person and then replace the job number and the hours. This is the worst case and it needs three commands though might even be done with fewer. 2

To simplify this process, I would advise making a hard-copy of the prior month's file so that the changes from the form 2 can be made onto that. Then, when you get on-line, the order that appears in your file will match the one in your hand and help avoid confusion. Of course, this is not necessary but the time spent doing this off-line may be worth it by increasing reliability of the data. 3

It is my guess that this procedure would involve more than 60% less commands than the current way. And when the system is slow (as it has been), the time savings for the inputter could be considerable. Furthermore, connect time would be drastically reduced and the NLS system would be available for other RADC users. 4

(J31878) 18-FEB-75 13:09;;;; Title: Author(s): Joe P. Cavano/JPC; Distribution: /RJC([ACTION]) EJK([INFO-ONLY]) JLM([INFO-ONLY]) ELF([INFO-ONLY]) RBP([INFO-ONLY]) ; Sub-Collections: RADC; Clerk: JPC; Origin: < CAVANO, PSO/2.NLS;1, >, 18-FEB-75 12:58 JPC ;;;;####;

Lexicography

If you wish you can look at my file 'LEXICOGRAPHY', but don't try to write in it.

Lexicography

I am putting together a listing of common terms bandied about in our division. I will be coming to some or all of you to get some help in defining these catchy little devils. Ultimately I will put together a glossary or dictionary of these terms so that we can have agreed-upon definitions. Meanwhile, I have been looking at publications like the American National Standard Vocablulary for Information Processing. Can anyone define a "Hartley" for me? Please don't look it up - I know the definition. 1

Lexicography

(J31879) 18-FEB-75 14:05;;;; Title: Author(s): Edmund J. Kennedy/EJK; Distribution: /RADC([INFO-ONLY]) ; Sub-Collections: RADC; Clerk: EJK;

Reply to WWP2 on Output Processing Directives

19-FEB-75 0612-PST PATTERSON:
Distribution: KENNEDY
Received at: 19-FEB-75 06:12:49 1

Ed, I'm putting a paper together and need some help with the directives to the output systems. What directives do I use and where do I put them to do the following: put a title on the first page centered, where it may take more than one line (single spaced) and double space the text with line numbers. Also I have to do a staff summary sheet. Is the forms system ready to do that? In any case that needs to be single spaced with spaces between paragraphs, except that I need a two line signature block, single spaced. Thanx---bill 1a

REPLY 2

Bill, 2a

There is a little known but very useful program that should take care of most of your problems with respect to the report you want to prepare. 2b

With the file of interest already loaded, the procedure is as follows: 2c

Goto Programs<CR> 2c1

Load Program Format<CR> 2c2

Goto Format<CR> 2c3

Insert Format<CR><CR> 2c4

Answer the questions that you are asked. Ident is your NLS ident which I think is WWP. Title is the title of the report just as you want it, don't do anything about spacing, the program should take care of it. For Journal number type a space or you can make up your own number ie. 75-1 or WWP-1 or something (you may not be able to use more than five characters) When you are asked for format number the answer is 1<CR>. 2d

This procedure shuld format the report for you and make you a title page. 2e

NB. The title page will indicate that you are William W. Patterson, Rome Air Development Center, Griffiss Air Force Base, N. Y. 13440. You will have to edit it to get your own information in there. 2f

After you have the file in pretty good shape you must output the file to the terminal. The commands are simply output to terminal which I think you get by typing OT<CR>. When you do this you will be prompted by the system asking you questions. The answers are Y N Y in order. This will print thefile with the processing directive functional. 2g

2h

If you want a real rundown on Output directives you will have to go to tenex and type dir<userguides>. See if the listing includes one on Output Processing Directives. If so you can load it in NLS and print it out. 2i

With respect to Your needs at this time I think that most of them can be solved with only a couple of types of directive. Note that all directives are of the form ".Directive;" 2j

- the directive makes the statement the first one on a new page. I think it works anywhere in the statement. I usually put it at the beginning of the statement. 2j1

- the directive makes the statement the last one on the page. I think it works anywhere in the statement. I usually put it at the end of the statement. 2j2

- puts a single line between statements regardless of level. Can be used for more than one line ie =2 or =3. 2j3

- puts a single line between statements at different levels. Can be used for more than one line ie =2 or =3. (Bill, I do not absolutely guarantee these because two I personally do not use them. YBS may only work within the plex, and YBS and YBL may be additive but I do not think so. I want this to go out to you quickly and I'm not in my office where I can look them up.) 2j4

- Makes sure that one line in a statement does not appear on the top of a ppage. This insures that at least five lines will appear. 2j5

The forms system per se is not ready to do much of anything but these will solve most of your problems. 2j6

Let me know how you make out. 2j7

Reply to WWP2 on Output Processing Directives

(J31880) 19-FEB-75 10:17;;;; Title: Author(s): Edmund J. Kennedy/EJK; Distribution: /WWP2([ACTION]) JLM([INFO-ONLY]) RDK([INFO-ONLY]) ; Sub-Collections: RADC; Clerk: EJK;

EJK 19-FEB-75 14:26 31881

Most Useless Command Contest Entry

I think DVN's entry, Forcecase Invisible, is a very useful command either with or without a filter and should therefore be disqualified. After all, some of us shy, unassuming types are secretly very proud of the fact that all of our invisibles are, aggressively, UPPERCASE.

Most Useless Command Contest Entry

My candidates for the least useful commands are Merge Plex and Merge Branch. For some mysterious reason all they do for me is tell me that my first entry is non-existent. Of course, some purists would ask why anyone in his right mind would want to merge anyway, but here on the other coast some of us are old fashioned. 1

Most Useless Command Contest Entry

(J31881) 19-FEB-75 14:26;;;; Title: Author(s): Edmund J. Kennedy/EJK; Distribution: /FEED([ACTION]) DVN([ACTION]) DLS([INFO-ONLY]) ; Sub-Collections: RADC; Clerk: EJK;

ILA 20-FEB-75 10:48 31882

test of journal mail system

Hi John this is a test of the JOurnal system--let me know if you get this message--Larry 1

ILA 20-FEB-75 10:48 31882

test of journal mail system

(J31882) 20-FEB-75 10:48;;;; Title: Author(s): I. Larry Avrunin/ILA; Distribution: /JJZ([ACTION]) ILA([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: ILA;

first attempts

1

first attempts

(J31883) 20-FEB-75 14:38;;;; Title: Author(s): John J. Zenor/JJZ; Distribution: /ILA([ACTION]) JJZ([INFO-ONLY]) ; Sub-Collections: NIC; Clerk: JJZ;