82 informatica 3

YU ISSN 0350-5596

Iskra Delta

računalniški sistemi delta®

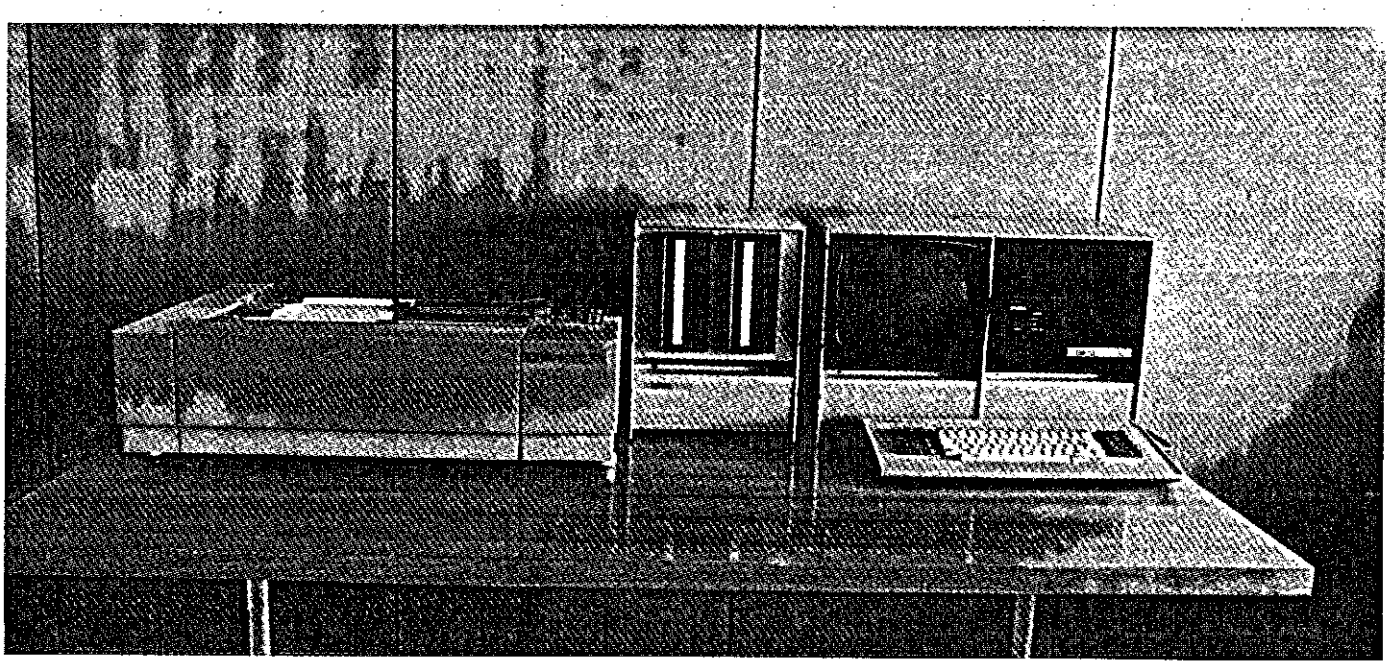
MIKRORAČUNALNIŠKI 16-BITNI, RAZVOJNI SISTEM ISKRADATA 100

HITER RAZVOJ IN VEDNO VEČJE TEHNIČNE AKTIVNOSTI ZAHTEVAJO SPOSOBNEJŠE MIKRORAČUNALNIŠKE SISTEME.
STROKOVNJAKI DELOVNE ORGANIACIJE ISKRA DELTA SO IZDELALI NAJNOVEJŠI 16-BITNI MIKROPROCESOR, KI GA ODLIKUJEJO SLEDEČE KARAKTERISTIKE:

- VEČJA OBDELOVALNA HITROST
- BOLJŠI UKAZI
- VEČJE NASLOVNO OBMOČJE
- MOŽNOST VEČJE OBDELOVALNE REZERVE
- NOVA ARHITEKTURA MIKROPROCESORJA

SISTEM ISKRADATA 100 JE POPOLNI RAZVOJNI SISTEM, KI OMOGOČA PISANJE, POPRAVLJANJE, PREVAJANJE IN PREIZKUŠANJE LASTNE PRORAMSKE OPREME, PREIZKUŠANJE IN EMULIRANJE ŽE RAZVITE APARATURNE OPREME IN PROGRAMIRANJE V VIŠJIH PROGRAMSKIH JEZIKIH. APARATURNA IN PROGRAMSKA OPREMA OMOGOČATA PROGRAMIRANJE 16-BITNIH IN 8-BITNIH MIKROPROCESORJEV.

VSE OSTALE INFORMACIJE VAM NUDI: ISKRA DELTA, SLUŽBA TRŽNEGA KOMUNICIRANJA, 61000 LJUBLJANA, PARMOVA 43.



delta računalniški sistemi®

informatica

Časopis izdaja Slovensko društvo INFORMATIKA, 61000 Ljubljana, Parmova 41, Jugoslavija

Uredništvo in uprava: Informatica, Parmova 41, 61000 Ljubljana, telefon (061) 312-988, teleks: 31366 YU DELTA

Letna naročnina za delovne organizacije je 500,00 din, za redne člane 200,00 din, za študente 100,00/50,00 din, posamezne številke 100,00 din
Žiro račun št.: 50101-678-51841
Stališče uredništva se lahko razlikuje od mnenja avtorjev.

Pri financiranju revije sodeluje tudi Raziskovalna skupnost Slovenije.

Na podlagi mnenja Republiškega sekretariata za prosveto in kulturo št. 4210-44/79 z dne 1.2.1979, je časopis oproščen temeljnega davka od prometa proizvodov.

Tisk: Tiskarna KRESIJA, Ljubljana

Grafična oprema: Rasto Kirn

ČASOPIS ZA TEHNOLOGIJO RAČUNALNIŠTVA IN PROBLEME INFORMATIKE
ČASOPIS ZA RAČUNARSKU TEHNOLOGIJU I PROBLEME INFORMATIKE
SPISANIE ZA TEHNOLOGIJA NA SMETANJETO I PROBLEMI OD OBLASTA NA INFORMATIKATA

YU ISSN 0350–5596

LETNIK 6, 1982 – Št. 3

V S E B I N A

informatica

Published by INFORMATIKA, Slovene Society for Informatics, 61000 Ljubljana, Parmova 41, Yugoslavia

Headquarters: Informatica, Parmova 41, 61000 Ljubljana, Phone: (061) 312-988, Telex: 31366 Delta

Annual subscription rate for abroad is US $ 22 for companies, and US $ 7,5 for individuals.

Printed by: Tiskarna KRESIJA, Ljubljana

DESIGN: Rasto Kirn

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 6, 1982 – No. 3

CONTENTS

AN INTRODUCTORY TUTORIAL ON DATA DICTIONARY SYSTEMS

A. MILTON JENKINS

INDIANA UNIVERSITY

UDK: 681.3.01

This tutorial on data dictionaries assumes that the reader has a basic knowledge of modern data processing. Because of ambiguities in the literature key database systems terms are defined. The characteristics of data dictionaries are described both in the context of database management systems and as stand alone systems. Many specific features of data dictionaries are examined, including: data definition generation, generalized I/O procedures, validation, privacy and security, test data generation, browse data facilities, maintenance support and data definition languages. The various uses of data dictionaries are then discussed. The tutorial concludes with an examinnation of the costs associated with the acquisition, development, and use of data dictionaries.

UVODNI PREGLED SISTEMOV S PODATKOVNIMI SLOVARJI. Pričujoči pregled podatkovnih slovarjev predpostavlja, da ima bralec znanje s področja sodobne obdelave podatkov. Zaradi literarnih dvoumnosti se najprej opredeljujejo pojmi s področja sistemov podatkovnih baz s ključi. Značilnosti podatkovnih slovarjev so opisane v povezavi s sistemi upravljanja podatkovnih baz in kot samostojni sistemi. Obravnavanih je več specifičnih lastnosti podatkovnih slovarjev, kot so: generiranje podatkovnih definicij, posplošene V/I pprocedure, veljavnost, zasebnost in varnost, generiranje preizkusnih podatkov, pripomočki za podatkovno vejitev, vzdrževalna podpora in jeziki za definiranje podatkov. Preučujejo se različne uporabe podatkovnih slovarjev. Pregledni članek prinaša tudi analizo stroškov, povezanih z nastajanjem, razvojem in uporabo podatkovnih slovarjev.

INTRODUCTION

A data dictionary may be viewed initially as simply a list of data item descriptions. As data becomes more voluminous and multi-functional, it becomes increasingly important to an organization, and some sort of data management becomes necessary [5]. Just as managers use other management tools to manage the organization's capital and human resources, the data dictionary is a management tool useful in managing the organization's information/data resources [9]. As the use of data has become more important and complex in supporting decision support systems (DSS), data processing has also become more complex as applications have grown in size and numbers. Traditionally each application owned its own data and data management was a relatively simple task. With the proliferation of application systems throughout the organization many application systems access the same or similar data. A data dictionary is the tool that may be used as documentation for "inventory control" of data to prevent or reduce unwanted redundancy and inconsistency in both new and existing applications [6]. It should be noted that a data dictionary can function in either a data base management system environment or in a traditional file environment.

This paper explains what data dictionaries are, their functions, how they can be used in commercial environments and what costs are involved in developing and operating a data dictionary.

GENERAL DEFINITIONS

The brief definition of a data dictionary used to open the paper suggests that data dictionaries may take on a wide variety of forms. The physical list of data item descriptions may be on paper or on a computer file. The data items described may be restricted to program variables or they may include nearly every data item in

the organization, including reports, forms, files, and descriptive data used in or by various organizational components including users, departments, programs, procedures, projects, personnel, etc. The descriptions also may vary from a few characters in length to a narrative description containing validity checks, privacy information, and "where used," cross-reference reports. Finally, the list may be oriented for human use, machine use, or both.

With such a wide variety of data dictionaries, many different definitions and many similar terms have appeared in the literature. Some authors draw the distinction between human usable data dictionaries and machine readable data directories, occasionally ignoring the possibility of a data dictionary being readable by both groups. Others claim that data dictionaries are programs that perform validity checks and supply COBOL programs with data divisions. To minimize the confusion this paper will use the following definitions for the key terms.

Data Dictionary (DD) - an automated list of data item descriptions that is readable by both humans and machines. For development tasks the list could be read to avoid creating files and procedures that are similar to those already in use. Maintenance programmers could use the list to learn the full impact of a change, and users may read the list to better learn what data and routines are currently available to support their needs. The computer may read the list to generate COBOL data divisions or to retrieve necessary password information and control schemas about a data item.

Data Dictionary System (DDS) - the data dictionary together with a variety of computer programs and manual procedures to help manage data. Some programs will generate reports or respond to queries from systems analysts or other users concerning the current or proposed inventory of data items. Other programs may generate validation routines or other generalized procedures.

International Data Base Management System (DBMS) Directory (IDBMSD) - a list of the physical locations of data items and the relationships among data items, in machine readable form only. (Some IDBMSDs also contain definitions of logical data models, e.g. Prime DBMS.) In a data base environment, programs would report any inconsistencies between the dictionary and the directory. Many of the functions of DDS and IDBMSD overlap or complement each other. For example, the DDS could be chosen to supply passwords if the IDBMSD has inadequate or nonexistent security. The two systems together, however, may provide an unnecessary repetition of functions.

Data Administrator (DA) - a person or department that controls all or some of the changes and additions to the data dictionary.

Database Administrator (DBA) - in a database environment, a person or department that controls all or some of the changes and additions to the internal DBMS directory.

CHARACTERISTICS OF DATA DICTIONARIES

The automated DD must be stored on a computer file of some type. Arthur Andersen's Lexicon System actually defines a data dictionary as a direct access storage device [8]. Although it would be possible to have a DD of limited utility stored as a sequential tape file or a sequential disk file, direct access is necessary to efficiently perform tasks such as retrieving an individual data item's password or valid range in order to automatically generate an input procedure. Because the DD is, simply stated, a file, it has many of the same requirements as other files. For example, the DDS must provide the DD with validation and security. Consequently, the DD will contain information about itself.

General Attributes of DD

A DD contains data about data. It is possibly the only computer file in many organizations that is maintained primarily for data processing's use. Just as a payroll file contains information about the operations of the payroll department, in order to help it run more efficiently, the DD contains information about the operations of the data processing department for the same reason [4].

Investigating the DD's configuration further, the DD could be stored as an ISAM file. In a database environment, the DD could be stored as part of a database it defines or even as an entirely separate database. For example, a total database could actually include the records for the DD that describe the remainder of the database.

Relationship to DBMS

Although it is becoming common to think of the DD as part of a DBMS, the two should be viewed as distinct. A DD may be implemented with or without a DBMS. An organization planning to eventually have both could install either one first and later install the other as needs dictated. Although the DD is separate from the internal DBMS directory, many of their functions are

similar, and the DD and the DBMS may interact with each other [1]. Further, the DDS may read the internal DBMS directory to ensure that the two are consistent.

Specific Features of DDS

It should be noted that a single purchased DDS package rarely has all the features that the data processing department needs. This has caused many organizations to build their own DDS. Unfortunately the results are usually unsuccessful. A major cause of failure is the high degree of technical expertise needed to maintain the DDS [7]. The remainder of this section will discuss specific features of DDS that are currently implemented on either purchased or "home-grown" packages.

Data definition generation

Among the most common functions associated with DDS is that of supplying data definitions, such as a COBOL Data Division, to a compiler preprocessor [8]. In addition to greatly reducing the coding effort, this enforces some amount of discipline on the programmers. It also enables a programmer to code "brief" names, and let the DDS expand them into longer, more meaningful names thereby overcoming a frequent criticism of COBOL. If all data divisions must be built by the DDS, the data administrator is assured of control over the storage for every computerized data item in the organization. (Note that realistically "all" data items will not come from the DDS. For example, local loop counters or switches are of interest only in the context of the logic of a particular program.) This makes the control of unwanted redundancy much easier. The practice also helps establish common names so that the same data item is not known by different names in different departments. However, when there are valid reasons for having different names, the DDS enables the use of synonyms without storage redundancy. Thus the DD functions as a communication aid throughout an organization.

The DDS may also maintain several different versions of the data or multiple logical descriptions of the data. It is not uncommon for a separate production and several test versions of the same data to coexist.

Generalized I/O procedures

In addition to supplying data divisions, the DDS may supply generalized input and output procedures using information in the data item description. This further reduces the programming load and may lead to increased standardization among reports.

Validation

In the absence of a DBMS, a DDS may provide validation rules for data entering or exiting the file or database. These rules may be sent to either the generalized I/O procedure mentioned above, or to conventionally written input code. The validation rules, such as valid ranges, a list of valid values, or datatype tests, may be retrieved from the DD either as the input procedure is being compiled or during input processing or during the output procedure.

For compile-time use, a compiler preprocessor reads the records of the DD to obtain the valid ranges that are part of each data item's description. These values are then supplied to the compiler and hardcoded into the input procedure. Consequently, a change in the valid range would require recompilation of the affected modules, but perhaps not the whole program. (COBOL with its global data environment would require a complete recompilation, but other languages, e.g., PASCAL, ASSEMBLY, etc., may not.)

For execution-time use, the input procedure reads a data item and then searches the DD for its description to determine the validity of the input. A change in the valid range with this method would require only an update to the DD.

Since the DD itself is a file, control over its own validity is also required. Thus as the DD is updated it should supply validity rules for itself.

Privacy and security

The DD may provide additional security and privacy to the files or the database [5]. As with the validation rules, the privacy information may be supplied to the input/output procedures either at a compile-time or execution-time. The DD is read by the compiler preprocessor of the input/output procedure to obtain the

valid password or perhaps a list of program names, users, or terminals authorized access to the item. Providing privacy is one of the functions that may overlap with the internal DBMS directory. For example, IMS provides privacy down to the data item level.

Once again, since the DD is itself a file, it also needs controls to ensure privacy and security. Since a major objective of the DDS is often to allow only one individual to monitor all changes to uses of data, uncontrolled updates to the dictionary are unacceptable.

Test data generation

Another function that a DD could perform is to automatically provide test data as changes are made to programs or to relationships among data [2]. The generation of this test data would be based on the validity rules in the DD and possibly with some information about the nature of the program change. This testing data could go beyond "black-box testing" [13] and facilitate the examination of the program structure.

Statistics generation

The generation of certain statistics could provide valuable assistance to the database administrator in database performance tuning. The DDS can store the frequency of use for each data item and compare this frequency against other data items over time. While many of the previously mentioned features could be provided by a relatively unsophisticated COPY Library, this feature requires an executing program to feed information back to the DD.

This tuning is especially important when the details of actual use cannot be predicted before the system is implemented, or when usage may change drastically as the user's understanding evolves and becomes more sophisticated. If a data item, or an entire file, is being used much more frequently than the system designer initially anticipated and response time is critical, the data item should be changed to a more accessible storage structure. Further, if a data item is used infrequently but within a short or highly clustered time pattern, the data item could be staged to a faster storage structure for a specified period of time.

Browse data facility

Some DDS provide users with the ability to browse through the actual data on an interactive basis [2]. This facility may increase the understanding of users who are unfamiliar with data processing and be useful in the intelligence phase of decision making. For systems analysts and programming users, this facility provides assistance during the design and maintenance of application systems.

Maintenance aid

The DD can identify which programs require modification or recompilation for any given change by generating a "where used" report. This can greatly improve program maintenance. Some DDS go so far as to automatically generate and submit the job control language (JCL) necessary to carry out the recompile for a modification.

Data definition language

As with most other files, the DD must be updated. The element within the DDS that does this is known as the Data Definition Language (DDL). (This DDL should not be confused with a DBMS DDL, which in most cases will not be the same language.) The DDL may be run in a batch or interactive mode, or both. The optimal mode would be installation dependent. If the installation has many new systems in the development phase with updates occurring frequently, an easy to use interactive DDL may be appropriate. If, however, updates to the DD are rare and seldom of an emergency nature, they may be handled entirely through the data administrator. In this situation, a batch system may be quite adequate.

The DDL may be set up such that there is relatively little control over modifying the test version and strict control over the production version. In addition to updating the DD, the DDL also provides the capability to browse through the DD and examine descriptions of existing data. If the DD is implemented as a database via a DBMS, then a common DDL can be used for both the BMS and DDS. This constrains the proliferation of languages and, especially for in-house development, may be a powerful advantage over using an additional

language.

Thus, the features of many DDS reduce the amount of coding required for the system while at the same time adding many important benefits such as sophisticated validity checks and security. The DDS also aids developers, maintenance programmers, users and managers by giving them a clear overall picture of the organization's data. Finally, the DD allows MIS managers to gain some measure of control over the organization's data resources and imposes some discipline in the MIS department.

USES OF DATA DICTIONARIES

Besides generating elaborate input-output procedures, a DD is also useful as a documentation or communication tool. However, unlike much documentation, the DD can be easily updated and is useful throughout the development, operations, and maintenance phases of the systems life cycle [12]. Also unlike much documentation, its update is an essential part of ongoing operations. For the development of new systems, the analyst can begin with knowledge of what data already exist. This is a substantial step toward preventing additional redundancy and speeding the development process for a new system. Although programmers may be required to learn an additional language, a DDS DDL, the overall programming load is reduced by the automatic generation of various procedures. For the maintenance programmer, the impact is greater on the effort required for change. The DDS can disseminate information to many groups and highlight areas of conflict that need to be addressed. The DD may also be used to clean up the mess created by years of unmanaged data growth and to reduce unwanted redundancy.

The DDS provides information about data from one centralized and coordinated source, rather than dispersed among all applications. As managers begin to gain some control over data, the data processing department comes closer to being auditable. A global view of data can be obtained, showing the interdependencies among applications. At a time when many people within the data processing department may be concentrating on their own narrow, specialized areas, a formalized view of data as a whole is necessary for management of the organization's data resources.

In addition to being centralized, the automated documentation provides much faster retrieval times and provides ad hoc services that were impossible with manual documentation. The notion of centralization is strong in most DD applications. However, there are no hard constraints that would prohibit the development of multiple or segmented DDs to support distributed environments. Distributed DDs could parallel the distributed database concepts that are now beginning to appear.

Program development of new systems in an integrated environment has many of the same characteristics as program maintenance in a traditional environment. In both cases the programmer/analyst is constrained by what already exists. Before beginning new development, the programmer/analyst must determine the existing structure of the data in order to choose a file or data base design for the new system that will best fit in with the overall data processing of the organization. The DDS could also be used to simulate various file design alternatives.

A DDS aids development by allowing more flexibility in dividing tasks among the MIS staff. Using a DDS as a communication tool, a programmer/analyst can let others know the current state of the project by updating the DD after completing a task. Without this method, the amount of communication required among staff greatly inhibits progress on the development task.

The generation of data divisions and input/output procedures with validation and privacy logic are additional features of DDS that aid system development. This reduces the development time required while generally enhancing the quality of the system.

For a proposed maintenance change, the DDS can generate "where used" reports to show inconsistencies or gaps in the proposed change. The change could be simulated by the DDS to provide an estimate of the time and cost necessary to implement it. This prediction can be used for planning purposes or even to decide whether the change should be made at all. To some extent the DDS could be used to evaluate the feasibility of a proposed

new system [12].

A DDS can also point out areas in need of tuning based on the performance statistics it generates. Finally, DDS can aid maintenance by automatically generating test data based on the program or logical data changes.

Many DDS packages are oriented toward either maintenance or development. MSP's Data Manager is primarily for maintenance, while Cincom's PRIDE is intended primarily as a design tool [7]. The PRIDE development method breaks the development cycle into nine phases, with signoffs required after each step. Throughout these steps PRIDE maintains the repository for information about data definitions and other related information about the current state of the project. Users and designers make queries into the DD throughout the nine steps. Interestingly, several companies have found that PRIDE's powerful analytical capabilities are also extremely useful during maintenance.

As systems become more interdependent by sharing data, standards become crucial. One area may forget to inform others of a change, or the affected department may forget or misunderstand the message. By having a data administrator responsible for all changes and by requiring all changes to be made through the DD, the organization has a mechanism to help ensure that all the necessary steps in the change process have been taken. The standardized, centralized authority benefits new development as well, ensuring that overall system performance is considered.

The very presence of completely enforceable standards creates an atmosphere of stability -- perhaps too much stability. On the one hand, the process of requiring a request to go through the data administrator prevents haphazard, careless changes. On the other hand, the data administrator may at times become a bottleneck and delay simple emergency changes.

THE COSTS OF DATA DICTIONARIES

Although this paper has thus far focused on how and why DD$ are useful, they are not appropriate for every organization. Whether the DD is developed in-house or purchased, a relatively high installation cost must be incurred [3]. To operate the system efficiently, a data administrator must develop plans for introducing changes to existing data covered by the DD. In return for this effort the DD generally does not produce much operational data, such as payroll checks or statements. The DDS helps the data processing department in maintaining a smoothly run operation by providing information to control unwanted redundancy and aid in maintenance.

An installation that shares little data across departments or otherwise manages its data well with traditional paper documentation probably would not gain adequate benefits from a DDS to offset its development and operating costs.

The adjustment to increased standardization also carries with it both installation and operating costs. For example, the changeover to commonly named program variables (even utilizing aliases) may represent a tremendous program maintenance effort. Once the application system is running, a policy of having all changes go through the data administrator could cause more problems than it solves, if there is relatively little danger of most changes affecting other parts of the system. For example, if the data administrator is gone or flooded with other requests, an emergency fix with no danger of affecting other systems may be needlessly delayed. There are, however, managerial solutions to this problem.

The chance of failure is another factor that must be considered in deciding whether to install a DDS. The DDS will require a great deal of commitment from all involved and many tasks will be transferred from programmer/analysts to the data administrator. How users and programmer/analysts feel about the DDS will greatly influence its success or failure. The transfer of duties may be welcomed by the programming staff or may be perceived as a loss of power or status. If the latter is true, programmer/analysts may not provide the cooperation necessary for the DDS to be successful.

CONCLUSION

A DDS contains many tools that may help the data processing department gain efficiencies. One way this is done is by increasing the quantity and quality of commu-

nication among all parties during system development and maintenance phases.

Communication aids include performance statistics to be used by analysts, "where used" reports for maintenance programmers, and query languages for the user community to search the DD and become aware of what is already available.

A DDS also aids the data processing department by generating data divisions and input/output routines containing validation and security logic. The DDS can generate test data and JCL to recompile programs affected by a change. By formalizing the maintenance and by requiring all changes to go through the data administrator, a DD can impose standards that decrease the risks inherent in changes.

Many claims have been made by both hardware and software vendors touting the advantages of DDS. And while these claims may be viewed with some skepticism, empirical evidence is now beginning to appear in the literature confirming many of the advantages discussed in this paper [10, 14]. In the final analysis, however, it remains the task of MIS managers to determine whether the opportunity to benefit from these tools in their own environment is adequate to justify the development and operating costs of these systems.

REFERENCES

1. Astrahan, M.M., et. al. "System R.: A Relational Data Base Management System," Computer (12:5) May 1979, pp. 42-48.

2. "The British Computer Society Data Dictionary Systems Working Party Report," Database (9:2) December 1977, pp. 2-24.

3. Bruun, Roy, "Ingredients of a Data Base," Infosystems (20:12) December 1973, pp. 32-36

4. Cahill, John J., "A Dictionary/Dictionary Method for Building a Common MIS Data Base," Journal of Systems Management (21:11) November 1970, pp. 23-29.

5. Collard, Albert F., "A Data Dictionary Directory," Journal of Systems Management (25:6) June 1974, pp. 22-25.

6. "The Data Dictionary/Directory Function," EDP Analyzer November 1974, pp. 1-13.

7. "Installing a Data Dictionary," EDP Analyzer January 1978, pp. 1-12.

8. Lexicon-General Description Manual, Arthur Andersen and Company, 1977.

9. Martin, George N., "Data Dictionary/Directory System," Journal of Systems Management (24:12) December 1973, pp. 12-19.

10. Martin, James, Computer Data-Base Organization, Second Edition, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

11. Myers, Glenford, J., The Art of Software Testing, John Wiley and Sons, New York, 1979.

12. Sakamoto, J.C. and Ball, F.W., "Supporting Business Systems Planning Studies with the DB/DC Data Dictionary," IBM Systems Journal (21:1) 1982, pp. 54-80.

13. Schussel, George, "The Role of the Data Dictionary," Datamation (23:6), June 1977, pp. 129-142.

14. Semprevivo, Phil, "Incorporating Data Dictionary/Directory and Team Approaches Into the Systems Development Process," MIS Quarterly (4:3) September 1980, pp. 1-15.

PROGRAMIRANJE V ADI I

ANTON P. ŽELEZNIKAR

ISKRA DELTA, LJUBLJANA

UDK: 681.3.06 ADA:519.682

Članek opisuje programiranje v jeziku ADA, natančneje v jeziku JANUS ADA, ki je v bistvu podjezik ADE. JANUS prevajalnik se izvaja na operacijskem sistemu CP/M, ki ga uporabljajo tudi mikroračunalniki Iskra Delta. Prikazani primeri kažejo, da je moč v ADI programirati vrsto "vsakdanjih" nalog. Članek (prvi del) prinaša celotno sintakso JANUS ADE v obliki sintaksnih grafov (dodatek A), tako da je vselej omogočena takojšnja verifikacija napisanih stavkov. Članek opisuje tudi omejitve in razširitve jezika JANUS ADA glede na definicijo jezika ADA. V nadaljevanju članka bodo prikazani novi programski primeri, formalna sintaksa in vzdrževalne dopolnitve jezika (oziroma kompilatorja) JANUS ADA (nova verzija 1.4.4).

Programming in ADA I. This article describes the programming in ADA language using JANUS ADA as an ADA sublanguage. JANUS compiler can be executed on CP/M operating system for a variety of microcomputer systems. Examples shown in this article represent solutions of "every-day" problems and introduce ADA to a programmer in a simple manner. This article (Part I) deals with the JANUS ADA syntax using syntax graphs (Supplement A); in this way for each programming case the syntax verification by the programmer is posible. The article describes restrictions and extensions of the JANUS ADA in comparision with ADA standard definition. In the next part of the article further examples, formal syntax, and maintainance completions of JANUS ADA (Version 1.4.4) will be presented.

1. Nastanek jezika ADA

ADA je programirni jezik za raznovrstno uporabo, kot so npr. numerični izračuni, sistemsko programiranje in paralelno izvrševanje programov v realnem času. Jezik je bil poimenovan po Adi Avgusti Lovelace, hčerki pesnika Byrona in sodelavki Charlesa Babbaga, pionirki računalništva v 19. stoletju. Jezik ADA so razvili v Parizu po naročilu obrambnega ministrstva ZDA v letih 1975 - 1980.

V zadnjem letu sta se pojavila tudi dva CP/M prevajalnika za jezik ADA: Janus (proizvajalec RR Software) in ADA (Supersoft). Prvi prevajalnik je reducirana različica jezika ADA, kjer je realiziranega približno 50% standarda za jezik ADA, drugi prevajalnik pa obsega realizacijo pod 50% standarda. V tem članku bomo obravnavali primere in sintakso za prevajalnik Janus.

Jezik ADA so načrtovali s tremi cilji: učinkovitost, zanesljivost in lahkotno vzdrževanje, vendar s spoznanjem, da bodo ADA programe pisali poklicni programerji (določena težavnostna stopnja programiranja). Čitljivost je poudarjena na račun lahkotnosti pisanja. Programske spremenljivke morajo biti eksplicitno deklarirane in njihov tip je določen, tako da obstaja zaščita proti avtomatični konverziji tipov. Ta lastnost zagotavlja namensko uporabo objektov. Čitljivost programov je podobno kot pri Pascalu zagotovljena s konstrukti v angleškem jeziku. Poseben poudarek je bil dan neobsežnosti jezika (to je razvidno tudi iz sintaksnih grafov v dodatku A tega članka).

ADA ima vrsto konstruktov iz drugih programirnih jezikov, kot so npr. Euclid, Lis, Mesa, Modula in Sue. Vsi ti jeziki so bili izvedeni iz Pascala. ADA je pravi proizvod jezikovnega načrtovalnega projekta in je v tem pomenu vrhunski jezikovni dosežek.

Na koncept ADE sta vplivala tudi jezika ALGOL 68 in Simula, pa tudi Alphard in Clu, toda ne v tolikšni meri kot pascalske izpeljanke.

Vsak ADA program je sestavljen iz zaporedja višjih programskih enot, ki se lahko vse posebej (separatno) prevajajo. Takšne programske enote so dejansko podprogrami, ki določajo izvršljive algoritme, nadalje tkim. paketni moduli (package), ki določajo množice objektov ali moduli opravil (tasks), ki določajo hkratne (paralelne) procese.

Od CP/M prevajalnikov za ADO si bomo podrobneje ogledali le enega, in sicer Janus. Jezik ADA je moč prevajati tudi z enostavnejšimi prevajalniki, ki temeljijo na uporabi rekurzivnega sestopa.

Proti uporabi jezika ADA pa se oglašajo tudi pacifisti, trdeč, da bodo ADO uporabljali za programiranje vojnih udarnih raket. ADA je spričo tega kot nov in visoko zmogljiv jezik tudi nevaren, čeprav ni namenjen le programiranju jedrskega orožja.

Janus je določena implementacija jezika ADA ter

je zaokrožen sistem za razvoj ADA programov. Sestavljajo ga prevajalnik, zbirnik, povezovalnik in inverzni zbirnik. Ta sistem se izvaja na operacijskem sistemu CP/M 2.x s procesorji 8080A ali 8085 ali Z80. Janus zahteva 56k zlogov hitrega pomnilnika in vsaj eno enoto z upogljivim diskom.

2. Primeri programov v ADI

Spoznavanje in učenje novega jezika je vselej izziv samemu sebi (preverjanje lastne sposobnosti). Spočetka je vse novo in določene konstrukte je potrebno še razumeti. Počasi se oblikujejo pojmi, ki se jih postopoma oprijemamo in na katerih gradimo prihodnje znanje. Na srečo imajo programirni jeziki določene skupne lastnosti in tudi ADA ima podobnost z vrsto drugih jezikov. Začnimo kar s primeri.

2.1. Uporaba Eratostenovega sita

S programom, ki opisuje algoritem Eratostenovega sita za izračun vseh praštevil od 3 do 16000, lahko ocenjujemo kvaliteto prevajalnika Janus ((2)), s tem da merimo čas izvajanja prevedenega programa. Ta metoda izračuna praštevil se izogne deljenju ter je izredno hitra, ker uporablja znanje o številih, ki ne morejo biti praštevila (npr. soda števila in mnogokratniki praštevil). Knuth ((1)) je že pred leti napisal program za tak izračun.

V listi 1 imamo zbirko z imenom PRIMES.PKG (tu je PKG pripona za ADA program, izvedena iz besede PACKAGE), pod to zbirko pa vidimo izvedbo ukazne zbirke PRIMES.COM, ki smo jo dobili s prevajalnikom. Program v listi 1 je paketno telo, je kratek in enostaven, tako da z njegovim razumevanjem nimamo posebnih težav. Na začetku paketnega telesa imamo deklaracije, tem pa sledi blok paketnega telesa (BEGIN, END, glej sintaksni graf za package body v dodatku A). Stavki, ki se v tem bloku pojavljajo, so PUT, FOR,

```
B>TYPE PRIMES.PKG
PRAGMA RANGECHECK(OFF); PRAGMA DEBUG(OFF);
PACKAGE BODY PRIMES IS
--      FROM BYTE SEPT. 81 BENCHMARK EVALUATION
    SIZE : CONSTANT := 8190;

    FLAGS : ARRAY(0..SIZE) OF BOOLEAN;
    COUNT,K,PRIME : INTEGER;

BEGIN
        PUT("10 ITERATIONS"); NEW_LINE;
TEN :   FOR ITER IN 1..10 LOOP
            COUNT := 0;
CLEAR :     FOR I IN 0..SIZE LOOP
                FLAGS(I) := TRUE;
            END LOOP CLEAR;
PRIME :     FOR I IN 0 .. SIZE LOOP
                IF FLAGS(I) THEN
                    PRIME := I + I + 3;
                    K := I + PRIME;
                    WHILE K <= SIZE LOOP
                        FLAGS(K) := FALSE;
                        K := K + PRIME;
                    END LOOP;
                    COUNT := COUNT + 1;
                    PUT("PRIME"); NEW_LINE;
                END IF;
            END LOOP PRIME;
        END LOOP TEN;
        PUT(COUNT); PUT("PRIMES"); NEW_LINE;
END PRIMES;
B>
B>PRIMES
10 ITERATIONS
1899PRIMES

B>
```

Lista 1. Ta program v jeziku ADA izračuna 1899 praštevil, kot se vidi iz izvajanja tega programa (ukaz PRIMES) na koncu te liste

```
A>TYPE DROBIZ.PKG
PRAGMA RANGECHECK(OFF); PRAGMA DEBUG(OFF); PRAGMA CONDCOMP(ON);
PACKAGE BODY DROBIZ IS
-- Ta  program vcita 8 celih stevil,  ki predstavljajo stevila ko-
-- vancev po 5, 10, 20, 50, 100, 200, 500 in 1000 par.
-- Program izpise celotno vrednost kovancev v dinarjih in parah.

   STET_KOV, CEL_VRED, STEV_PAR, STEV_DIN :  INTEGER;

   VRED_KOV : ARRAY (1..8) OF INTEGER;

BEGIN
   VRED_KOV(1) := 5; VRED_KOV(2) := 10; VRED_KOV(3) := 20;
   VRED_KOV(4) := 50; VRED_KOV(5) := 100; VRED_KOV(6) := 200;
   VRED_KOV(7) := 500; VRED_KOV(8) := 1000;
   CEL_VRED := 0;
   PUT("VPISI STEVILA POSAMEZNIH KOVANCEV"); NEW_LINE; NEW_LINE;

   FOR NASL_KOV IN 1 .. 8 LOOP
@     PUT("KOVANCI PO "); PUT(VRED_KOV(NASL_KOV),4);
@     PUT(" PAR: ");
      GET(STET_KOV); NEW_LINE;
      CEL_VRED := CEL_VRED + VRED_KOV(NASL_KOV)*STET_KOV;
   END LOOP;

   STEV_DIN := CEL_VRED / 100;
   STEV_PAR := CEL_VRED REM 100;

   NEW_LINE; PUT("CELOTNA VREDNOST ZNASA DIN "); PUT(STEV_DIN);
   IF STEV_PAR < 10 THEN
      PUT(",0"); PUT(STEV_PAR);
   ELSE
      PUT(","); PUT(STEV_PAR);
   END IF;

END DROBIZ;
```

Lista 2. ADA program DROBIZ.PKG v tej listi včitava števila posameznih kovancev (od 5 par do 10 din). Vsebuje dve vrstici, ki sta na začetku označeni z znakom '@' in ker ima na začetku PRAGMA CONDCOMP(ON); se tudi ti dve vrstici prevedeta in kasneje izvajata (glej listo 3).

Za FOR zanko včitavanja vhodnih podatkov se izračunavata posebej dinarski in parski del rezultata, ki se kasneje sestavita v celotno vrednost z ustreznimi PUT stavki na koncu tega programa.

V prvi vrstici programa imamo tri PRAGME, ki so navodila za prevajalnik, sam program pa ima obliko PACKAGE BODY (glej sintakso tega konstrukta v dodatku A na koncu članka). Program uporablja osemmestno polje VRED_KOV (deveta vrstica), v katerem so shranjene vrednosti kovancev, ki se določijo v vrsticah za rezervirano besedo BEGIN. Ta način določevanja vrednosti elementov polja seveda ni najbolj pripraven.

IF, WHILE, prireditveni stavki in procedura NEW_LINE. Programer, ki obvlada jezik Pascal, z razumevanjem programa v listi 1 ne bi smel imeti težav.

Kot vidimo iz liste 1, je ta program sestavljen iz zaporedja pragma, pragma, paketno telo. Z ukazom tipa PRAGMA damo prevajalniku določeno navodilo (v našem primeru izključimo opciji RANGECHECK in DEBUG), paketno telo pa je podobno podprogramskemu telesu in določa notranje (operacijske) podrobnosti.

Kadar imamo na začetku vrstice znak '@', lahko s posebnim stikalom to vrstico vključimo v prevajalni postopek; v našem primeru je ostala ta vrstica izključena, kot je razvidno iz izvajanja programa v listi 1.

2.2. Štetje drobiža

Ta program, ki ga imamo v listi 2, včitava vhodne podatke, sicer pa nima izrazitih posebnosti (glede na program v listi 1).

V deklarativnem delu programa imamo 4 spremenljivke tipe celo število (integer) in polje VRED_KOV z 8 elementi (indeksi 1 do 8). Ti elementi so celoštevilske konstante in predstavljajo vrednosti različnih kovancev (v parah). Preostali del programa je razumljiv sam po sebi; v stavku

```
STEV_PAR := CEL_VRED rem 100;
```

pa se izračunava celoštevilski ostanek (rem) pri deljenju CEL_VRED s 100.

```
A>DROBIZ
VPISI STEVILA POSAMEZNIH KOVANCEV

KOVANCI PO     5 PAR: 4
KOVANCI PO    10 PAR: 5
KOVANCI PO    20 PAR: 6
KOVANCI PO    50 PAR: 7
KOVANCI PO   100 PAR: 8
KOVANCI PO   200 PAR: 9
KOVANCI PO   500 PAR: 10
KOVANCI PO  1000 PAR: 3

CELOTNA VREDNOST ZNASA DIN 111,40
A>
```

Lista 3. Ta lista kaže izvajanje programa .DROBIZ z vhodnimi podatki in rezultatom

V listi 3 je prikazano izvajanje zbirke DROBIŽ .COM. Tu se izvajajo tudi stavki v vrsticah, označenih z '@', saj imamo na začetku programa v listi 2

```
PRAGMA CONDCOMP(ON);
```

V programu liste 2 bi lahko uvedli tudi tip kovanca, in sicer

```
type KOVANEC is (petpar, desetpar, dvajsetpar,
     petdesetpar, dinar, dvadin, petdin,
     desetdin);
```

Polje VRED_KOV bi v tem primeru definirali takole:

```
VRED_KOV: constant array
     (petpar .. desetdin) of INTEGER :=
     (petpar => 05, desetpar => 10,
      dvajsetpar => 20, petdesetpar => 50,
      dinar => 100, dvadin => 200,
      petdin => 500, desetdin => 1000);
```

2.3. Uporaba podprogramov

Uporaba posebnih procedur (podprogramov, subrutin) za vhod in izhod lahko ima več prednosti. V proceduro lahko programiramo želene formate in njihova specifikacije, ki jih uporabljamo potem avtomatično.

Nekateri podprogrami so v jeziku ADA definirani vnaprej (tako kot predpisuje standardno poročilo o jeziku ADA), vendar je implementacija zahtevanih vnaprejšnjih definicij podprogramov odvisna od posameznih prevajalnikov za jezik ADA. Priročni proceduri pri izdajanju teksta sta npr.

```
SET_COL(M);        --pomakni se v stolpec M
SPACE_OVER(N);     --pomakni se za N stolpcev v
                   -- desno
```

Če teh procedur ni v prevajalniški knjižnici, si pomagamo na dva načina. Sami definiramo obe proceduri s procedurnima deklaracijama ali pa nadomestimo učinke teh procedur z ustreznimi PUT stavki, kot je razvidno iz programa CENIK.PKG v listi 4 (vstavimo ustrezno število presledkov).

Program v listi 4 je sestavljen iz treh procedurnih deklaracij (TISK_NASL_INFO, STOLPNI_NASLOV in TISK_CEN) in glavnega programa, kjer se te tri procedure pokličejo. V listi 5 je prikazano izvajanje zbirke CENIK.COM, ki smo jo dobili s prevodom zbirke CENIK.PKG v listi 4.

Pri uporabi stavka (procedure) PUT imamo tri različne procedure za izpis, katerih uporaba je implicitno določena s tipom parametra (INTEGER, FLOAT, STRING). Ustrezna PUT procedura se izbira s pomočjo ugotavljanja enakosti tipov procedurnega in pozivnega argumenta. Procedura PUT ima lahko tudi dva parametra, pri čemer drugi parameter določa število izpisanih mest (vodeče ničle ze zamenjajo s presledki; glej listo 5).

3. Sintaksa podjezika ADA za prevajalnik JANUS

Že podnaslov pove, da lahko JANUS prevajalnik sprejme le podjezik (in določeno razširitev) jezika ADA, kot ga določa ADA jezikovno poročilo (glej slovstvo ((3)), ((6)) na koncu članka). Prvo vprašanje, ki ga v tej zvezi postavljamo, je seveda, kateri jezikovni konstrukti so pri JANUS implementaciji izvzeti (nedopustni, nelegalni) in kaj je k JANUS ADI dodanega.

Razlike med ADO in JANUS ADO so najprej razvidne iz sintaksne liste (dodatek B v drugem delu članka) oziroma iz sintaksnih diagramov (dodatek A). JANUS ADA ima glede na ADO omejitve pa tudi razširitve. Implementacijske omejitve JANUSa glede na ADO so tele:

1. Dinamična polja z razsežnostjo večjo od 1 niso podprta.
2. Največje število vgnezdenih podprogramov, blokov in zank je 25.
3. Največje število vgnezdenih procedur ali funkcij je 10.
4. Največje vgnezdenje variantnih delov v zapisu je 10.
5. Največje število preštevnih konstant v preštevnem tipu (enumeration type) je 100.
6. Največje vgnezdenje CASE stavkov je 10.
7. Dostopni (access) tipi niso implementirani.
8. Tile jezikovni pridevki niso implementirani:
   base, image, value, delta, actual_delta, bits, small, large, machine_rounds, digits, mantissa, emax, epsilon, first(), last(), machine_emax, machine_emin,

```
A>TYPE CENIK.PKG
PRAGMA RANGECHECK(OFF); PRAGMA DEBUG(OFF);
PACKAGE BODY CENIK IS
-- Ta program sestavi cenik po vnaprej predpisanem vzorcu, tako da
-- poklice tri podprograme (procedure).

   PROCEDURE TISK_NASL_INFO IS
   BEGIN
      NEW_LINE; NEW_LINE;
      PUT("                CENIK"); NEW_LINE;
      PUT("                -----");

      NEW_LINE; NEW_LINE;
      PUT("       KOD IZDELKA : IKE-1954"); NEW_LINE;
      PUT("       IZDELEK     : REGULATOR"); NEW_LINE;
      PUT("       CENA ENOTE  : ASCH 13,75"); NEW_LINE;
   END;

   PROCEDURE STOLPNI_NASLOV IS
   BEGIN
      NEW_LINE; NEW_LINE;
      PUT("      KOLICINA");PUT("            CENA");
         NEW_LINE;
      PUT("      --------");PUT("            ----");
         NEW_LINE;
   END;

   PROCEDURE TISK_CEN IS
      CENA_EN: CONSTANT INTEGER := 1375;
      CENA, SILINGI, GROSI : INTEGER;
   BEGIN
      FOR SKUP IN 1 .. 4 LOOP
         NEW_LINE;
         FOR KOL IN (SKUP*5-4) .. (SKUP*5) LOOP
            CENA := KOL*CENA_EN;
            SILINGI := CENA / 100;
            GROSI   := CENA REM 100;

            PUT("          "); PUT(KOL,2); PUT("          S");
               PUT (SILINGI,3); PUT(",");

            IF (GROSI < 10) THEN
               PUT("0"); PUT(GROSI,1); NEW_LINE;
            ELSE
               PUT(GROSI,2); NEW_LINE;
            END IF;
         END LOOP;
      END LOOP;
   END TISK_CEN;

BEGIN   --   Glavni program, ki poziva podprograme.
   TISK_NASL_INFO;
   STOLPNI_NASLOV;
   TISK_CEN;
END CENIK;
```

Lista 4. Ta ADA program je sestavljen iz treh procedurnih deklaracij, in sicer za izpis naslova cenika, za izpis dveh stolpnih naslovov ter končno za izpis cen v odvisnosti od količine izdelkov, kot kaže lista 5. Glavni program na koncu liste je sestavljen iz pozivov teh treh procedur.

Glede na oblike izpisov (lista 5) bi bilo moč v tem programu uporabiti bolj učinkovite vgrajene funkcije, ki pa žal v JANUS ADI niso implementirane. Taki funkciji bi bili NEW_LINE(n), kjer je n parameter za n vrstic in funkcija SET_COL(m), kjer je m parameter za m-ti stolpec. V programu smo pomanjkanje teh dveh funkcij nadomestili z zaporedjem funkcij NEW_LINE ter z ustrezno funkcijo PUT (glej osmo, deveto in deseto vrstico liste).

Spremenljivka SKUP v prvi FOR zanki določa štiri cenovne skupine (od 1 do 20 v listi 5) in spremenljivka KOL v drugi FOR zanki procedure TISK_CEN po pet elementov v vsaki skupini. Interval za spremenljivko KOL se tako spreminja v odvisnosti od vrednosti spremenljivke SKUP.

```
      machine_radix,          machine_mantissa,
      machine_overflows,   length,   length(),
      range,  range(),  constrained,  position,
      first_bit,     last_bit,    storage_size,
      terminated, priority, failure, count.
```

9. Agregati niso implementirani.
10. Preobremenjevanje (overlaoding) ni podprto.
11. Alokatorji niso podprti.
12. Rezinaste prireditve za polja niso implementirane.
13. Izpustitev dejanskih parametrov ni dovoljena.
14. Nepozicijski dejanski parametri niso dopustni.
15. RENAMES deklaracija ni implementirana.
16. Opravila (tasks) in moduli niso implementirani.
17. Optimizacije uporabnik ne more specificirati.
18. Izjemnostna obdelava ni implementirana, izjeme (exceptions) torej niso podprte.
19. Generični paketi niso implementirani.
20. Reprezentacijske specifikacije niso implementirane.
21. Nestična preštevanja niso implementirana.
22. Naključni V/I dostop ni podprt.

```
                    CENIK
                    -----

KOD IZDELKA : IKE-1954
IZDELEK     : REGULATOR
CENA ENOTE  : ASCH 13,75

KOLICINA            CENA
--------            ----

       1            S 13,75
       2            S 27,50
       3            S 41,25
       4            S 55,00
       5            S 68,75

       6            S 82,50
       7            S 96,25
       8            $110,00
       9            $123,75
      10            $137,50

      11            $151,25
      12            $165,00
      13            $178,75
      14            $192,50
      15            $206,25

      16            $220,00
      17            $233,75
      18            $247,50
      19            $261,25
      20            $275,00
```

Lista 5. Ta lista prikazuje izvajanje programa CENIK iz liste 4, ko je bil ta preveden (v zbirko CENIK.COM) s prevajalnikom JANUS

23. Tele jezikovno definirane pragme so brez učinka:
controlled, pack, priority, suppress, inline, interface, memory_size, optimize, storage_unit, system.

JANUS ima tudi razširitve glede na ADO. Pogojno prevajanje omogoča vključevanje in izključevanje programskih vrstic v postopek prevajanja. Alternativi se izbereta pri pozivu prevajalnika. Ta lastnost je koristna pri preizkušanju programa in odpravljanju napak (te vrstice imajo znak '*' na svojem začetku).

ASM stavek omogoča vstavitev poljubnih podatkov v prevedeni kod (ki ga je proizvedel JANUS prevajalnik).

IN in OUT stavek omogočata hitri konzolni V/I pri preizkušanju programov. Njuni funkciji lahko podvojimo z GET in PUT pozivi.

V dodatku A so zbrani sintaksni grafi (jezikovne definicije) za JANUS prevajalnik, verzija 1.4.3. Pri pisanju programov in pri učenju jezika ADA lahko te grafe uporabljamo in se tako izognemu prevelikemu številu napak v programu. V nadaljevanju članka bomo opisali nekatere izboljšave (dopolnitve) jezika JANUS ADA, ki so se pojavile v verziji 1.4.4.

Slovstvo k prvemu delu

((1)) D.E.Knuth: The Art of Computer Programming. Vol. 2: Semi-Numerical Algorithms. Založba Addison-Wesley, 1969.

((2)) J.Gilbreath: A High-Level Language Benchmark. Byte, Sep 1981, 180-198.

((3)) Reference Manual for the Ada Programming Language. US Dept of Defense, July 1980.

((4)) P.Wegner: Programming with Ada. Prentice-Hall, 1980.

((5)) I.C.Pyle: The Ada Programming Language. Prentice-Hall, 1981.

((6)) H.Ledgard: Ada - An Introduction, Ada Reference Manual. Springer-Verlag, 1980.

((7)) Janus System User Manuals. RR Software, P.O.Box 1512, Madison, Wisconsin 53701.

((8)) M.Exel: Programiranje sprotnih in vgnezdenih sistemov: procesi v Adi. Informatica 5(1981), štev.2, 8-18.

((9)) I.C.Pyle: Using Ada for Specification and Design. Informatica 5(1981), štev.3, 4-10.

Dodatek A

V dodatku A imamo sintaksne grafe za gramatiko jezika JANUS ADA, kot je opisana v ((7)). Sintaksni graf je slika gramatičnega pravila in za vsako pravilo imamo po en graf. Formalna sintaksa jezika JANUS ADA bo opisana v dodatku B.

Razlike med standardno ADO in JANUS ADO so še vedno velike. Težava je v tem, da vrsta konstruktov standardne ade še ni implementirana v JANUS ADI. Oba proizvajalca ADA kompilatorjev za mikroračunalnike obljubljata, da je njun cilj implementacija celotne ADE. Če smo se učili ADO iz knjig, npr. iz ((3, 4, 5, 6)), moramo biti previdni, ker bo sicer kompilator JANUS javljal veliko število napak. Torej je priporočljivo, da najprej pogledamo, kateri konstrukti jezika ADA v JANUSu niso implementirani (točke 1 do 23 v podpoglavju 3 tega članka).

Sintaksni grafi se pojavljajo v vrstnem redu, kot so zapisana sintaksna pravila v sintaksni listi (dodatek B). Ta vrstni red je določen s priročnikom, kjer se posamezni jezikovni konstrukti obravnavajo oziroma pojasnjujejo. V drugem delu članka bomo zgradili kratek priročnik, tako da bo učenje novega jezika lažje.

Abecedni red sintaksnih kategorij je tale:

accuracy_constraint, actual_parameters, adding_operator, address_specification, argument, array_type_definition, asm_statemant, assignment_statement, attribute

base, based_integer, basic_loop, block

case_statement, character_literals, character_string, choice, compilation, component_declaration, component_list, compound_statement, condition, context_specification, constraint

decimal_number, declaration, declarative_item, declarative_part, derived_type_definition, discrete_range

enumeration_literal, enumeration_type_definition, exception_choice, exception_declaration, exception_handler, exit_statement, exponent, exponentiating_operator, expression, extended_digit

factor, formal_part, function_call

goto_statement

identifier, identifier_list, if_statement, index_constraint, indexed_component, inout_statement, integer, integer_type_definition, iteration_clause

label, letter, letter_or_digit, literal, logical_operator, loop_parameter, loop_statement

mode, multiplying_operator

name, null_statement, number_declaration, numeric_literal

object_declaration

package_body, package_declaration, package_specification, parameter_declaration, pragma, primary, private_type_definition, procedure_call, program_component

qualified_expression

range, range_constraint, real_type_definition, record_body, record_type_definition, relation, relational_operator, representation_specification, return_statement

selected_component, sequence_of_statements, simple_expression, simple_statement, statement, subprogram_body, subprogram_declaration, subprogram_specification, subtype_declaration, subtype_indication

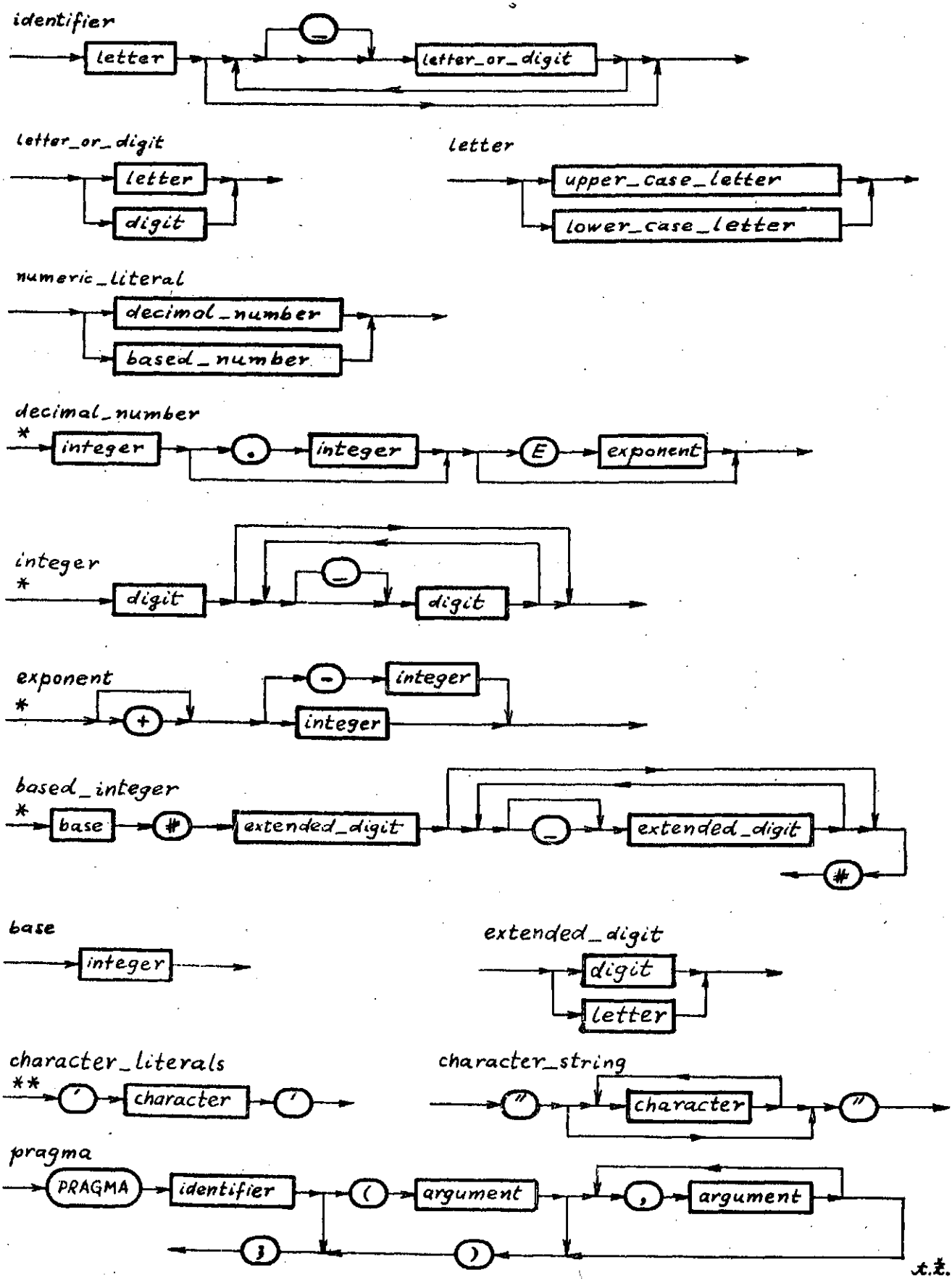
term, type_conversion, type_declaration, type_definition, type_mark

unary_operator, use_clause

variant

with_clause

SINTAKSNI GRAFI JEZIKA JANUS ADA



t.š.

argument
*
identifier
character_string
number

declaration
*
object_declaration
type_declaration
subtype_declaration
number_declaration
subprogram_declaration
package_declaration

object_declaration
*
identifier_list : CONSTANT subtype_indication := expression
array_definition
;

number_declaration
identifier_list : CONSTANT := literal_expression ;

type_declaration
*
TYPE identifier IS type_definition ;

identifier_list
, identifier

type_definition
*
enumeration_type_definition
integer_type_definition
real_type_definition
array_type_definition
record_type_definition
derived_type_definition
private_type_definition

subtype_declaration
SUBTYPE identifier IS subtype_indication ;

subtype_indication
type_mark constraint

type_mark
type_name
subtype_name

constraint
*
range_constraint

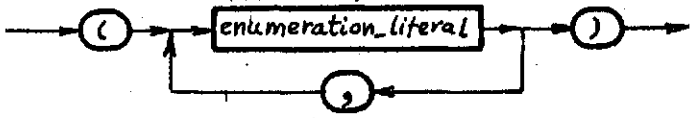
derived_type_definition
*
NEW subtype_idication ;

s.š.

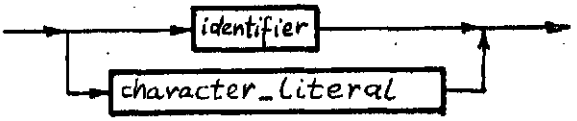
range_constraint

range

enumeration_type_definition



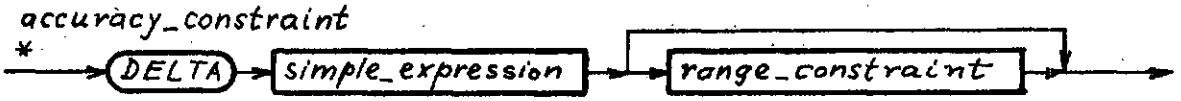
enumeration_literal



integer_type_definition

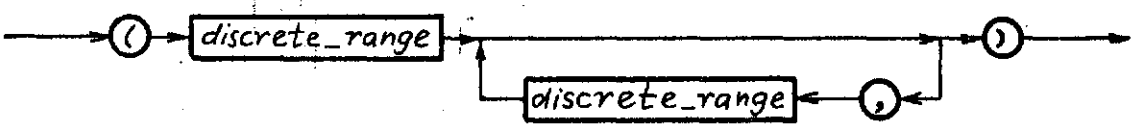
real_type_definition

accuracy_constraint

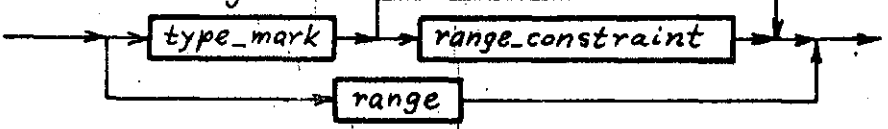


array_type_definition

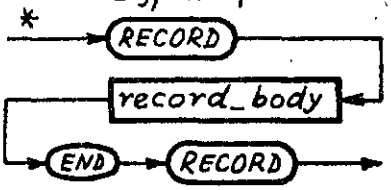
index_constraint



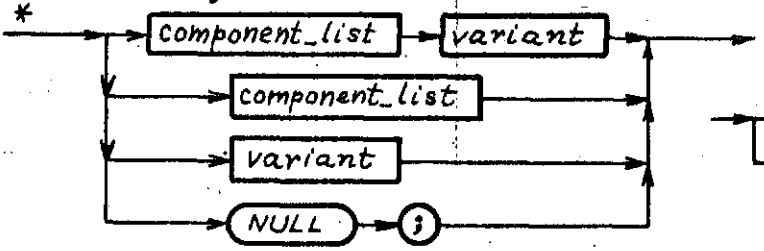
discrete_range



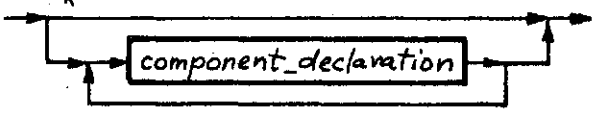
record_type_definition



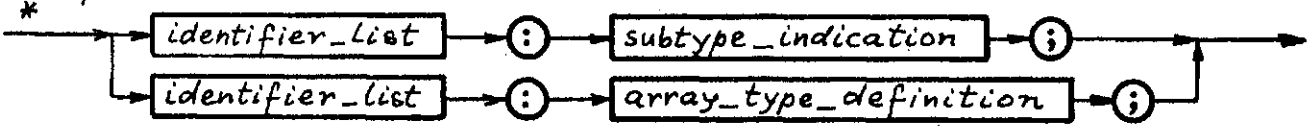
record_body



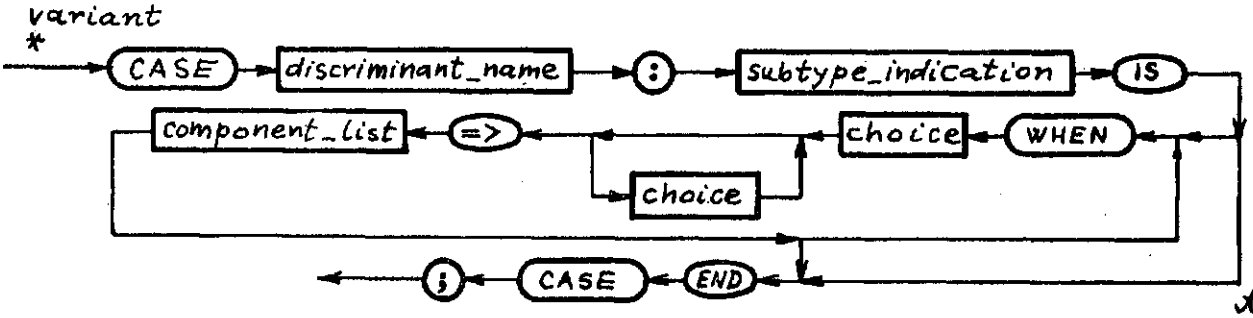
component_list



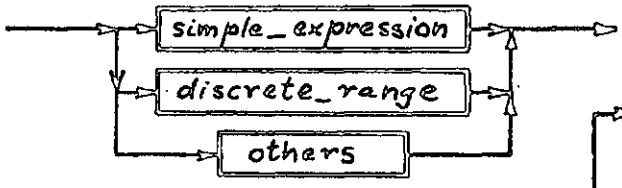
component_declaration

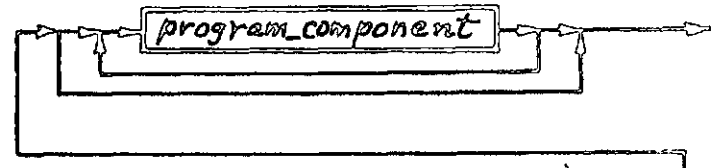


variant



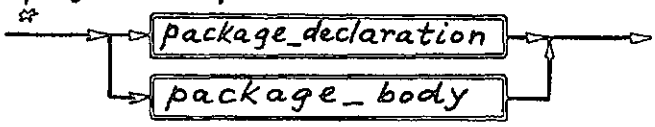
choice



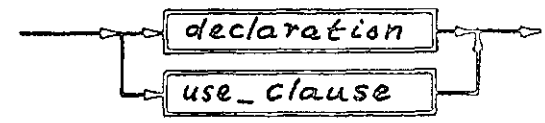


declarative_part

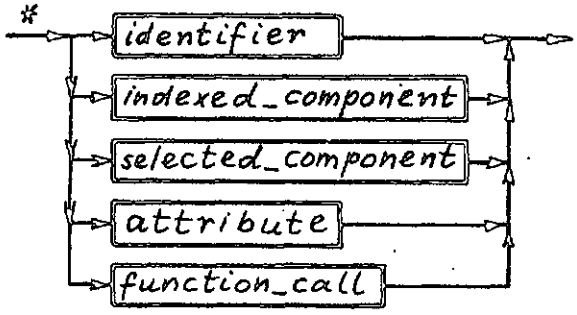
program_component



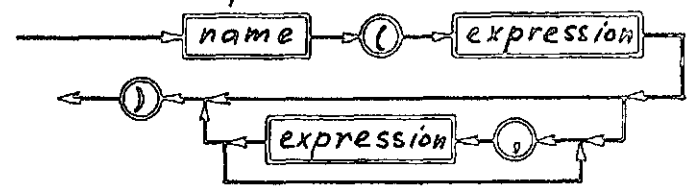
declarative_item



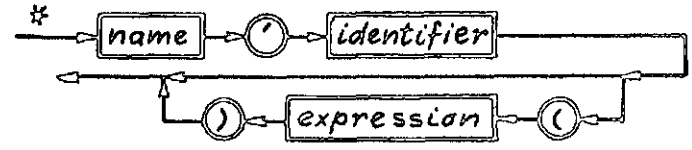
name



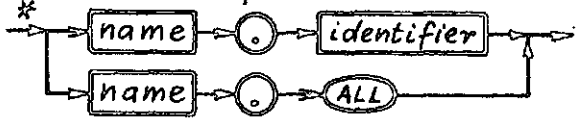
indexed_component



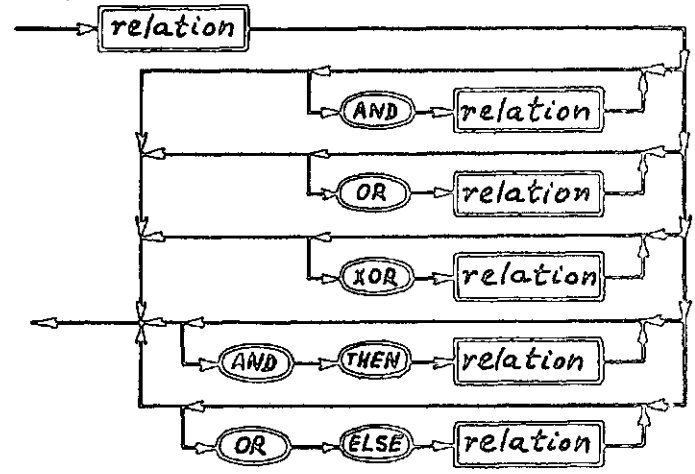
attribute



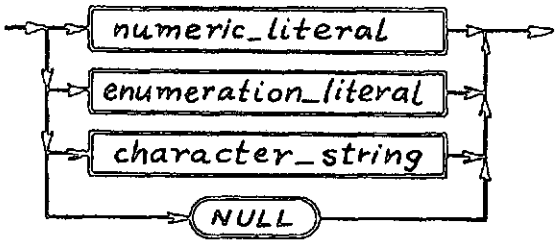
selected_component



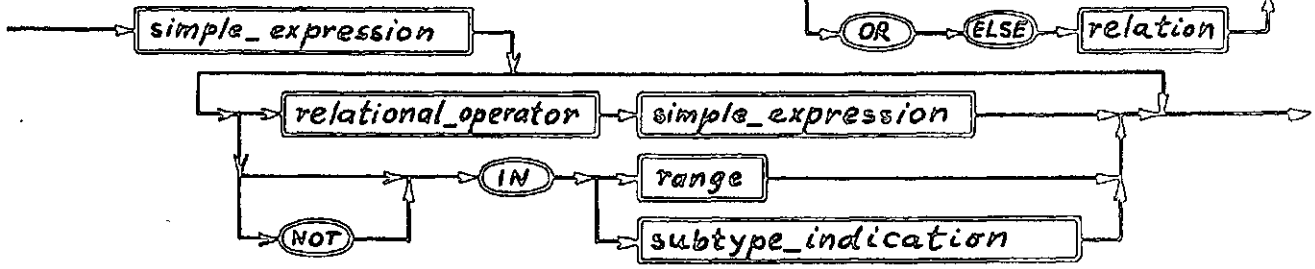
expression



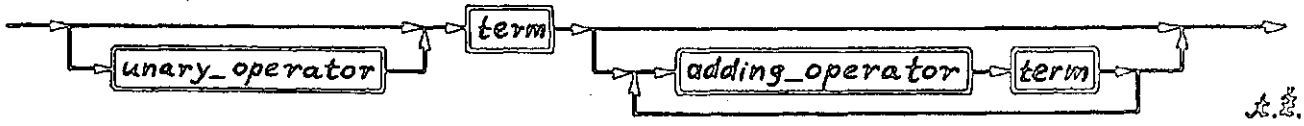
literal



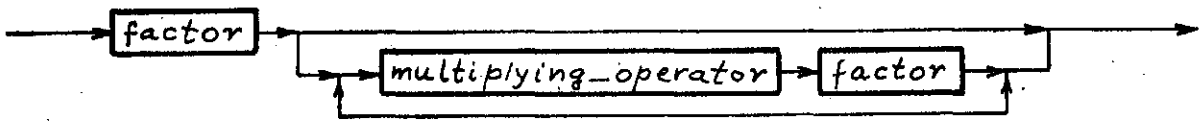
relation



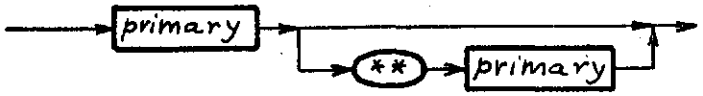
simple_expression



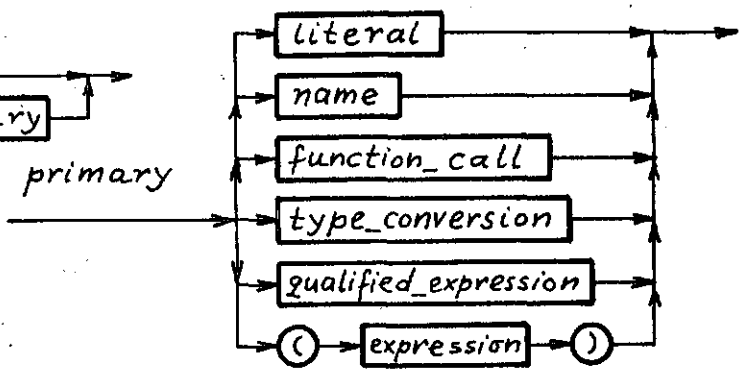
term



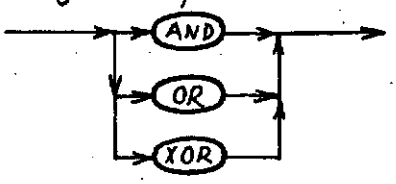
factor



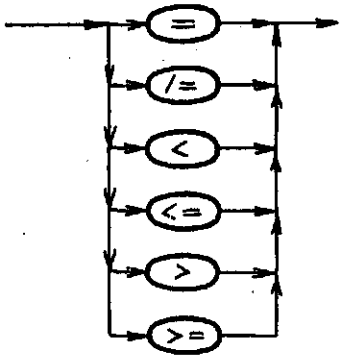
primary



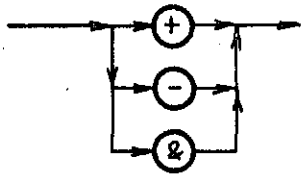
logical_operator



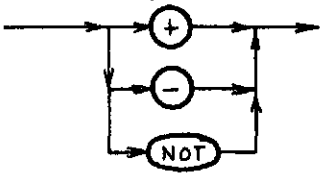
relational_operator



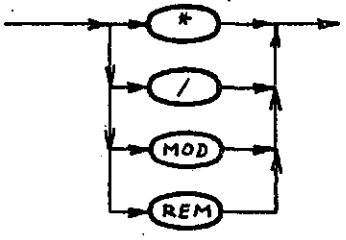
adding_operator



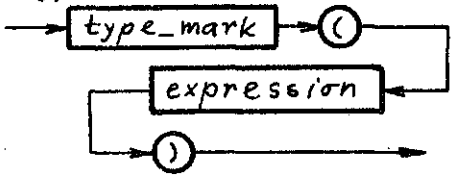
unary_operator



multiplying_operator

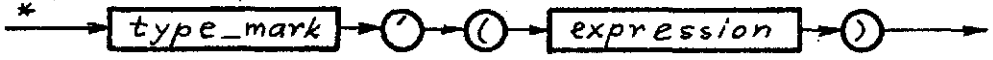


type_conversion

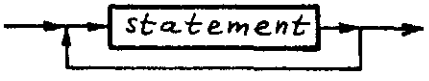


exponential_operator

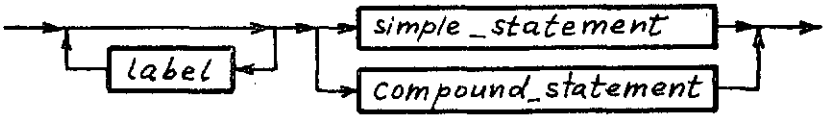
qualified_expression



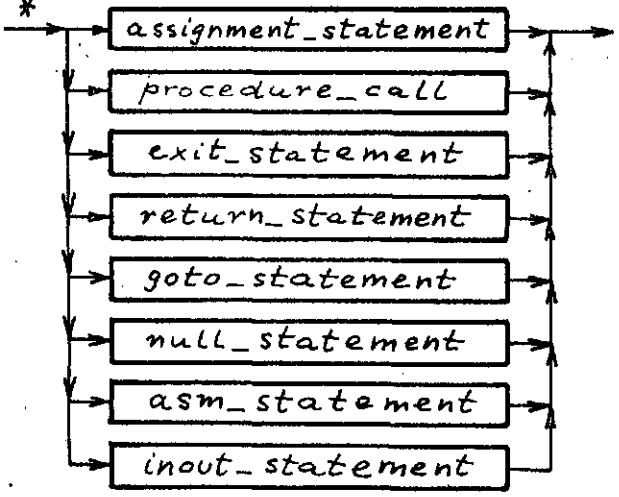
sequence_of_statements



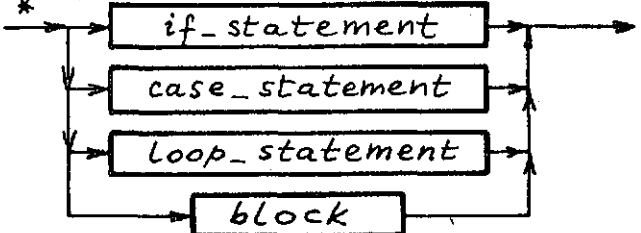
statement



simple_statement

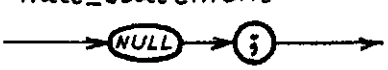


compound_statement



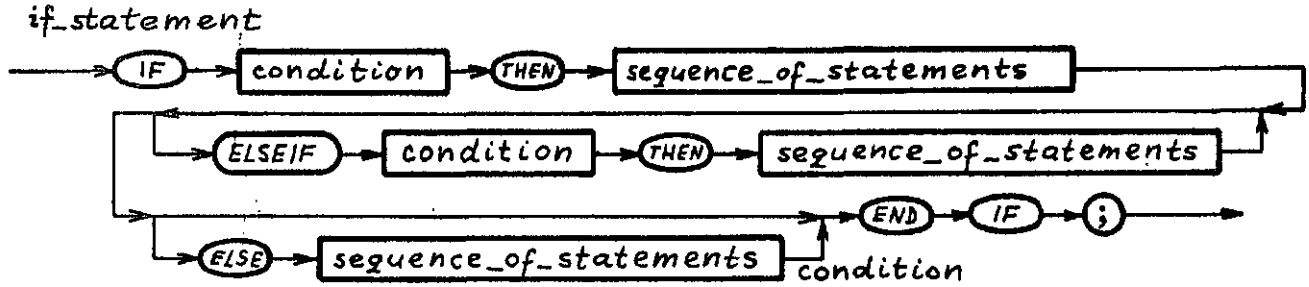
label

null_statement

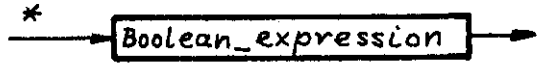


assignment_statement

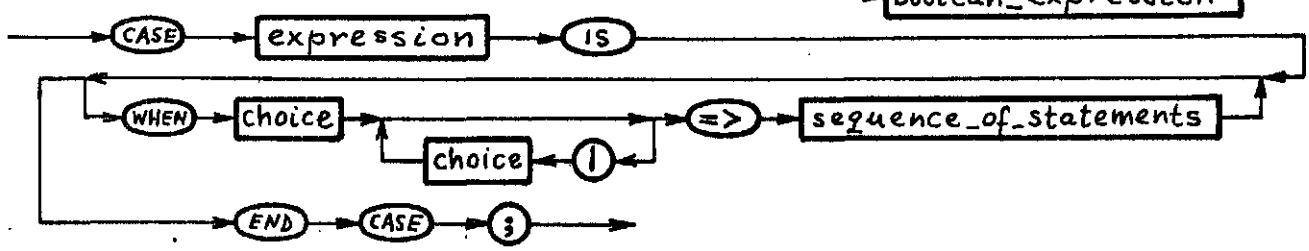
if_statement



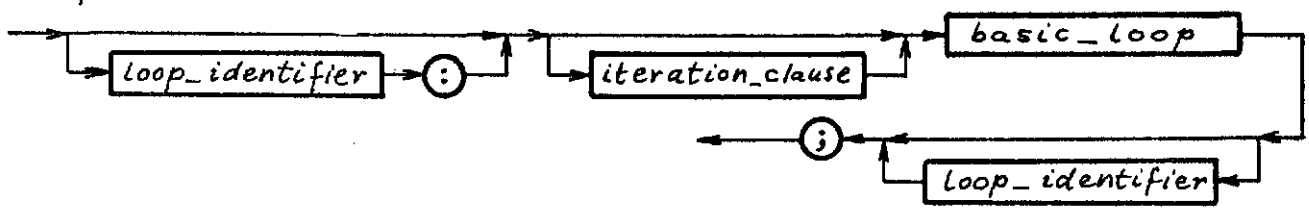
condition



case_statement

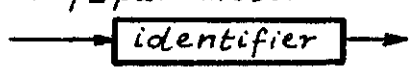


loop_statement

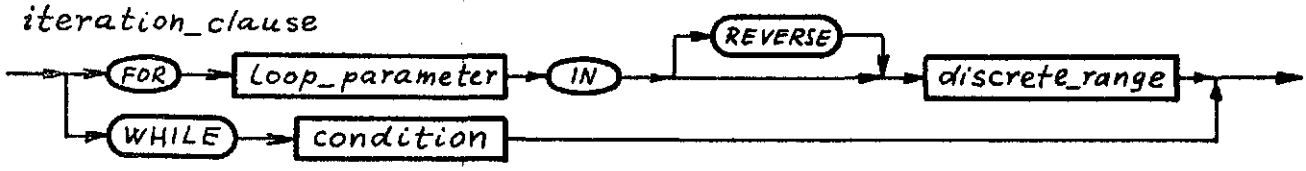


basic_loop

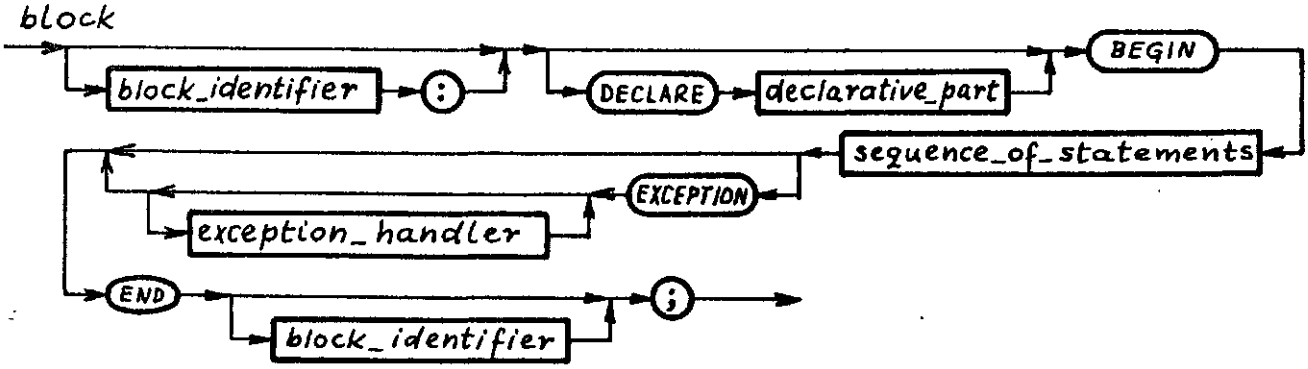
loop_parameter



iteration_clause



block

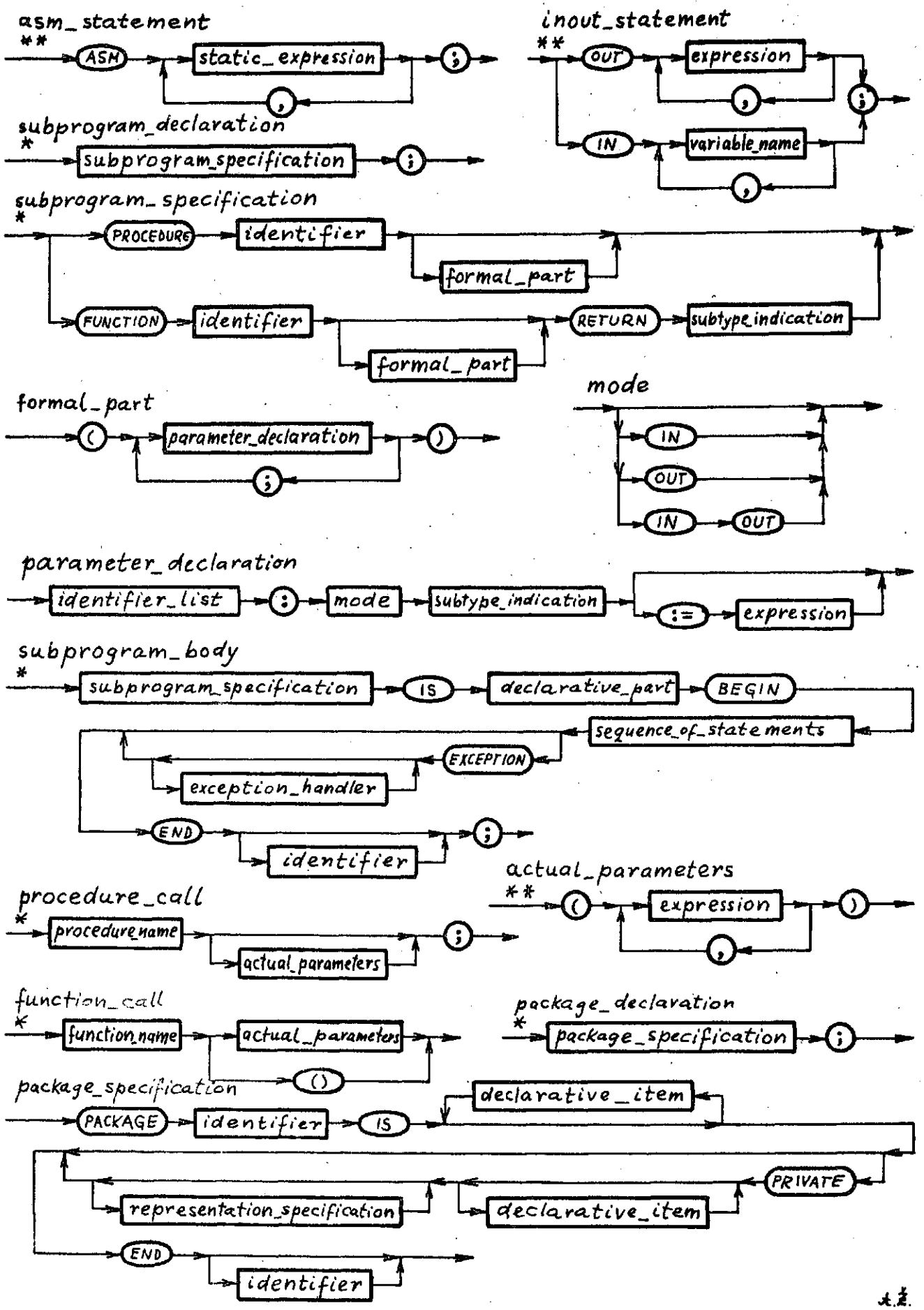


exit_statement

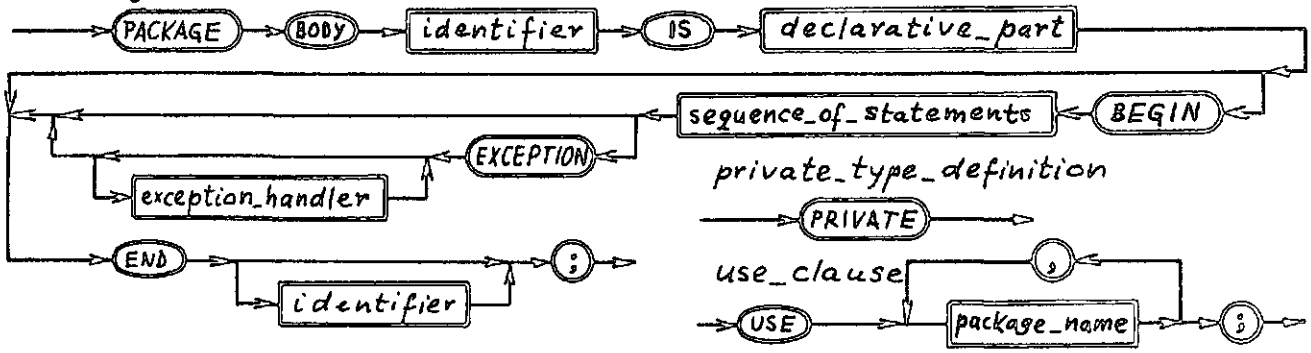
A.Ž.

return_statement

goto_statement


asm_statement
**
ASM
static_expression
;
,
inout_statement
**
OUT
expression
,
;
IN
variable_name
,
subprogram_declaration
*
subprogram_specification
;
subprogram_specification
*
PROCEDURE
identifier
formal_part
FUNCTION
identifier
formal_part
RETURN
subtype_indication
formal_part
(
parameter_declaration
;
)
mode
IN
OUT
IN
OUT
parameter_declaration
identifier_list
:
mode
subtype_indication
:=
expression
subprogram_body
*
subprogram_specification
IS
declarative_part
BEGIN
sequence_of_statements
EXCEPTION
exception_handler
END
identifier
;
procedure_call
*
procedure_name
actual_parameters
;
actual_parameters
**
(
expression
,
)
function_call
*
function_name
actual_parameters
()
package_declaration
*
package_specification
;
package_specification
PACKAGE
identifier
IS
declarative_item
PRIVATE
representation_specification
declarative_item
END
identifier

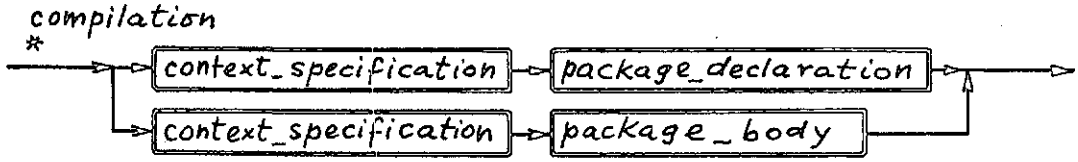
package_body



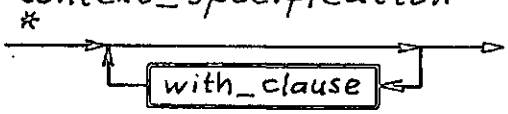
private_type_definition

use_clause

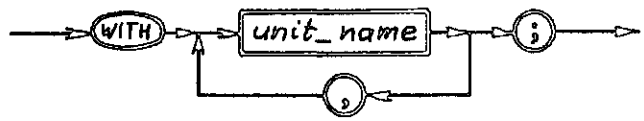
compilation



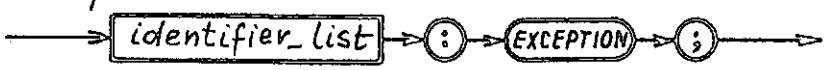
context_specification



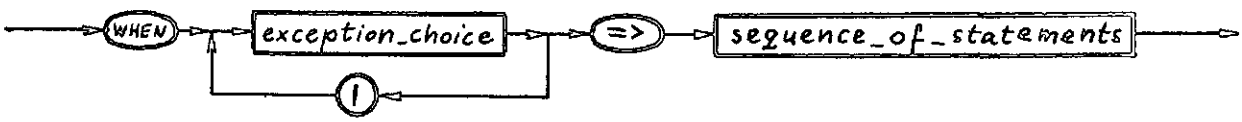
with_clause



exception_declaration

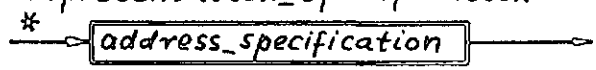


exception_handler

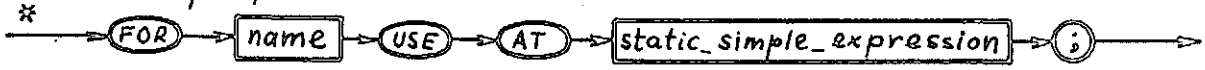


exception_choice

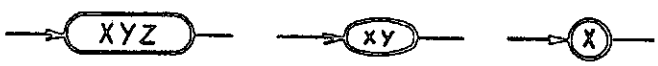
representation_specification



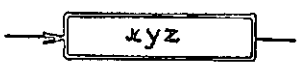
address_specification



LEGENDA:

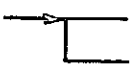


terminali XYZ, XY, X :
(terminalni simboli, rezervirane besede, operatorji, ločilni znaki)

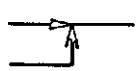


neterminal xyz : [definiran na drugem mestu]
(sintaksna kategorija, sintaksna spremenljivka)

konec definicije neterminala : smer konstrukcije elementa sintaksne kategorije



alternativna vejitev



stičišče alternativ

A.Ž.

KOMUNIKACIJA SEKVENCIJALNIH PROCESA U RAČUNARSKIM SISTEMIMA SA VIŠE MIKROKOMPJUTERA

NIKOLA HADJINA,
VELIMIR CVITAŠ

SVEUČILIŠNI RAČUNSKI CENTAR ZAGREB, JUGOSLAVIJA

UDK: 681.3:519.685

Jedan od najznačajnijih problema u postupku realizacije računarskih sistema sa više mikrokompjutera, je postupak komunikacije i sinhronizacije više paralelnih procesa. U radu će biti dano rješenje uvođenjem specijalnih komandi, programskih procedura, koje se ugrađuju na nivou nadzornog sustava (aplikacije) mikroprocesorskog sistema. Realizacija tog postupka bit će provedena za multimikroprocesorske sisteme sa zajedničkom memorijom.

COMMUNICATION OF SEQUENTIAL PROCESSES IN THE MULTI-MICROCOMPUTER SYSTEMS:
One of the most important problems which have to be solved in the multimicrocomputer systems design are communication and sinhronization procedures for parallel processing. One approach by introducing of special commands, which will be implemented on monitor or application level, is presented. Implementation of these commands on the multimicrocomputer system with common memory is given.

1. UVOD

Brz razvoj tehnologije integriranih krugova dovodi do novih pristupa u sklopovskoj i programskoj realizaciji računarskih sistema sa više mikrokompjutera. Nastoje se pronaći rješenja kako povezati te komponente te kako ih nadopuniti odgovarajućom programskom podrškom da bi one bile što efikasnije i pouzdanije u eksploataciji. Postoji nekoliko interesantnih arhitektura od kojih su najznačajnije mreža mikrokompjutera preko standardnih kanala, te multimikroprocesorski sistemi realizirani preko zajedničke memorije. Da bi se takovi računarski sistemi mogli što efikasnije koristiti potrebno je osigurati komunikaciju i sinhronizaciju sekvencijalnih procesa.

Ključni element u razvoju programskih komponenti za takove računarske sisteme su viši programski jezici pogodni za programiranje paralelnih procesa. Prijedlozi za realizaciju takovih programskih jezika (Hoare i Hansen) postoje.

Raširena metoda za komunikaciju procesa je ažuriranje i ispitivanje zajedničke memorije. Na žalost ova metoda može dovesti do neispravnosti u radu za neka sklopovska rješenja, što je dakako povezano sa skupoćom izvedbe. Predložene su mnoge metode za sinhronizaciju kao npr. semafori (Dijkstra), događaji (PL/I), uvjetni kritični odsječci (Hoare), monitori i repovi (Hansen), putevi (Campbell) i dr.

Cilj je ovoga rada da pokuša pronaći što jednostavnija rješenja za gornje probleme, a koja se lako implementiraju na multimikrkompjuterske sisteme.

Primjena multimikrokompjuterskih sistema uočljiva je najviše u komunikaciono orjentiranim računarskim sistemima (terminalski koncentratori, statistički multipleksori i dr.), u realizaciji snažnih centralnih procesora upotrebom mrežne arhitekture procesora (ILIAC IV), te u realizaciji lokalnih mreža na principu komutacije paketa, gdje mikrokompjuteri preuzimaju ulogu usmjeravanja poruka u mreži računala, tj. realizaciju logičke veze upotrebom fizičkih linija. Ključnu stavku u rješavanju ovih problema ima i sam operacioni sistem kome treba posvetiti posebnu pažnju.

2. DEFINIRANJE PROBLEMA I MOGUĆA RJEŠENJA

Kao što se iz dosad iznesenog može zaključiti mogu se uvesti apstraktni pojmovi procesa i repova. Pod procesima podrazumjevamo programske produkte koji obavljaju određeni posao u okviru kompleksnog računarskog sistema npr., kontrolu ulazno/izlaznih linija, kanala i dr., a pod repovima podrazumjevamo puteve poruka između procesa. Dakle jedna poruka može biti zaprimljena od jednog procesa, a posredstvom repa može biti prenesena drugim repovima. Završni proces prenosi stvarno poruku na fizičku liniju ili kanal. Prema tome naš osnovni nadzorni sustav mora omogućiti prenos poruka (spremnika) između procesa. On mora također raspoređivati procese, omogućiti stvaranje i unošenje procesa i repova, on mora upravljati sa korištenjem spremnika, zajedničkom memorijom i drugim sredstvima koja se koriste od strane više procesa. Realizacija gore navedenih funkcija usko je vezana na utrošak sistemskog vremena, pa izboru algoritama, po kojima se funkcije izvode, treba posvetiti veliku pažnju. Najvažniji problemi s kojima se treba suočiti su: 1) međusobno isključivanje procesa, čime se osigurava da dva procesa koji se izvode paralelno ne pristupaju istim strukturama podataka (spremnici, repovi, tabele i dr.) istovremeno; 2) komunikacija između paralelnih procesa; 3) sinhronizacija paralelnih procesa.

Neka od mogućih rješenja navedena su u ovom radu.

2.1. MONITORI

Korištenje monitora [4] povezano je sa postojanjem programskih jezika kao CONNCURENT PASCAL na mikroprocesorskim sistemima, što najčešće nije slučaj.

2.2. IMPLEMENTACIJA VLASTITIH PROGRAMSKIH PROCEDURA

Implementacijom vlastitih programskih procedura potrebno je riješiti pitanje međusobnog isključivanja, sinhronizacije i komunikacije paralelnih procesa. Međusobno isključivanje se realizira sa dvije programske procedure za testiranje i postavljanje (TS) ključeva nad kritičnim sekcijama, te procedure za otklanjanje ključa (CTS). Kritična sekcija je sredstvo koje se koristi od strane više procesa (npr. tabele, repovi, spremnici i dr. u zajedničkoj memoriji).

IMPLEMENTACIJA ULAZNO/IZLAZNIH PROCEDURA

Prema [2] uvodi se jednostavan oblik ulazno/izlaznih naredbi koje se koriste za komunikaciju između paralelnih procesa.

Ova komunikacija između dva procesa se zbiva kada jedan proces pozove drugi kao odredište za izlaz, a drugi pozove prvi kao izvor za ulaz. U ovom slučaju izlazna veličina iz prvog procesa se kopira sa prvog procesa u drugi proces. Ne postoji automatsko "bufferiranje." Općenito ulazna ili izlazna komanda se zadržavaju sve dok nije spreman odgovarajući izlaz odnosno ulaz.

Takovo kašnjenje je nevidljivo za proces koji je zadržan .

SINTAKSA ULAZNO/IZLAZNIH NAREDBI

Može se prikazati na slijedeći način u BNF notaciji:

<ulazna naredba> :: = <izvor> ? <ciljna varijabla>
<izlazna naredba> :: = <odredište> ! <izraz>
<izvor> :: = <ime procesa>
<odredište> :: = <ime procesa>
<ime procesa> :: = <identifikator>

Ulazno/izlazne naredbe specificiraju komunikaciju između dva konkurentna sekvencijalna procesa. Takvi procesi mogu biti implementirani u sklopovima kao uređaji posebne namjene (čitač, štampač) ili mogu biti formirani u obliku nezavisnih (asinhronih) programa koji se mogu realizirati paralelnim naredbama (COBEGIN, COEND) ili predstavljaju programe koji se izvode na dediciranim procesorima.

Komunikacija između dva procesa se zbiva:

1. Kada ulazna komanda u jednom procesu specificira kao svoj izvor ime drugog procesa,
2. Kada izlazna komanda u drugom procesu kao svoje odredište specificira ime prvog procesa,
3. Kada se ciljna varijabla ulazne komande slaže sa veličinama označenim u izrazu izlazne komande.

Po ovim uvjetima, kaže se, ulazna i izlazna komanda međusobno korespondiraju. Takve dvije komande se mogu izvoditi istovremeno, s tim da se vrijednost izraza izlazne komande pridruži ciljnoj varijabli ulazne komande.

Ulazna komanda je u grešci ako se njezin izvor terminira.. Izlazna komanda je u grešci ako se njezino odredište terminira, ili ako je njezin izraz nedefiniran.

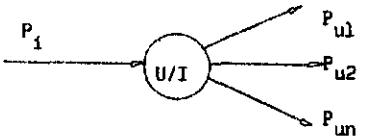
Zahtjevi na sinhronizaciju ulazno/izlaznih naredbi svode se na uvođenje kašnjenja na naredbu koja je prva spremna. Kašnjenje se prekida kada je odgovarajuća (korespodentna) naredba u drugom procesu također spremna, ili kada drugi proces završava. U ovom posljednjem slučaju prva naredba je u grešci. Kašnjenje može beskonačno trajati ukoliko niti jedna ulazna ili izlazna naredba međusobno ne korespondiraju. To je pojava potpunog zastoja.

Implementacija ulazno/izlaznih naredbi provedena je za multimikroprocesorski sistem sa zajedničkom memorijom sa specijalnom namjenom za korištenje ulazno izlaznih spremnika.

OPIS PROBLEMA

Osnovni problem u ovom slučaju je bio da jedan ili više procesa koristi podatke u U/I spremniku čim su raspoloživi.

Potrebno je zadovoljiti sinhronizacionu shemu.



U sinhronizacionoj shemi svi procesi (P_i i P_u) simuliraju se na U/I spremniku. Proces P_i izvodi izlaznu naredbu čije je odredište definirano u izvornom dijelu ulazne naredbe procesa (P_{ul}, P_{u2} - P_{un})

FORMAT ULAZNO IZLAZNIH NAREDBI

IZLAZNA NAREDBA:

OUTPUT P_j! P_i BUF_i

gdje je:

P_j - identifikator procesa za koga se puni ulazno/izlazni spremnik BUF_i (odredište)

P_i - identifikator procesa koji puni U/I spremnik BUF_i (proces koji izvodi naredbu OUTPUT)

BUF_i - logičko ime U/I spremnika

ULAZNA NAREDBA :

INPUT P_i ? P_j BUF_i

gdje je:

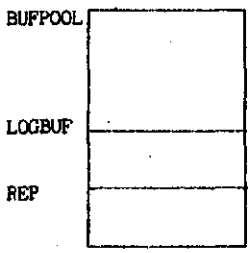
P_i - identifikator procesa koji puni ulazno/izlazni spremnik BUF_i (izvor)

P_j - identifikator koji čeka na ulaz podatka iz U/I spremnika BUF_i (proces koji izvodi naredbu INPUT)

BUF_i - logičko ime U/I spremnika

STRUKTURA PODATAKA

Ove naredbe izvode se nad slijedećom strukturom podataka koji su smješteni u zajedničkoj memoriji.

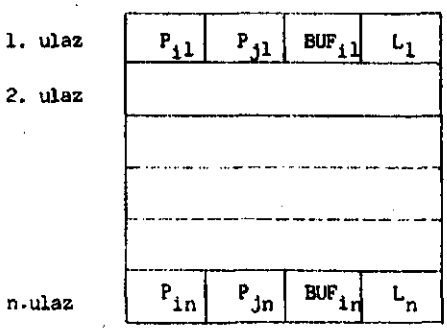


BUFPOOL: Prostor ulazno/izlaznih spremnika koji se inicijalizira po formatu kako je dano u [6]

LOGBUF: Tabela logičkih brojeva spremnika logički broj određuje indeks u tabeli na kojoj se nalazi adresa spremnika u BUFPOOL

REP: Rep za sinhronizaciju sa fomatom.

REP



Svaki ulaz u REP-u za sinhronizaciju sastoji se od 4 byta sa slijedećim značenjem:

P_i - identifikator procesa koji proizvodi podatak (izvor)

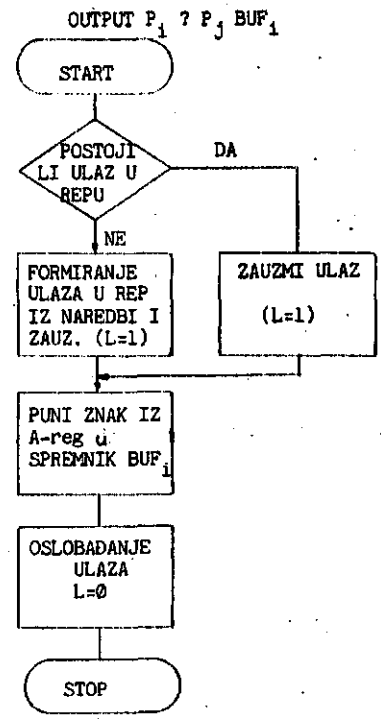
P_j - identifikator procesa koji koristi podatak (odredište)

BUF_i - logičko ime ulazno/izlaznog spremnika

TOK IZVOĐENJA NAREDBI

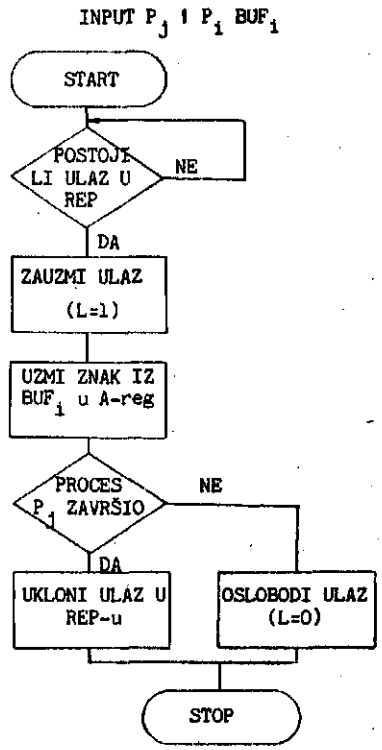
Ulazno/izlazne naredbe (INPUT i OUTPUT) implementirane su upotrebom MACRO naredbi za mikroprocesor Z80.

IZLAZNA NAREDBA



Izvođenjem ove naredbe unosi se znak iz A-registra u spremnik BUF_i upotrebom programa za beskonačni ulaz, kako je opisano u [6].

ULAZNA NAREDBA



Uzimanje znaka iz spremnika vrši se po proceduri opisanoj u [6].

IMPLEMENTACIJA NAREDBI

Naredbe su implementirane MACRO pozivom i to skupom instrukcija INTEL-8080 a izvode se na mikroprocesoru Z80.

```
OUTPUT    MACRO     BUFON PJ  PI
          LXI       D, BUFON
          MVI       C, PJ
          MVI       B, PI
          CALL      OUTPQ
          ENDM

INPUT     MACRO     BUFON  PI  PJ
          LXI       H, BUFON
          MVI       C, PI
          MVI       B, PJ
          CALL      INPQ
          ENDM
```

Subrutine INPQ i OUTPQ obavljaju funkcije naznačene u dijagramu toka za ulaznu i izlaznu naredbu.

ZAKLJUČAK

Uvedene ulazno/izlazne naredbe ugrađene su u programsku podršku dvoprocesorskog mikro sistema čija je funkcija bila koncentriranje asinhronih ulaza (terminala) na sinhroni ulaz računala. Tu su vršeni eksperimenti rukovanja ulazno/izlaznim spremnicima od najjednostavnijeg načina (ulazno/izlazni spremnici pridruženi svakom terminalu posebno), do primjene zajedničkog prostora ulazno/izlaznih spremnika i uvedenih sinhronizacionih ulazno/izlaznih naredbi.

Na sličan način moguće je upotrebom uvedenih programskih procedura provesti sinhronizaciju i komunikaciju sekvencijalnih procesa u mreži mikrokompjutera. Ulogu spremnika (medija za prenos podataka) tada preuzimaju kanali kojima su povezani mikrokompjuteri.

LITERATURA:

1. Dijkstra, E. W. Co-operating Sequential processes, Programming Languages, F. Genuys, Academic Press, New York, 1968.
2. C. A. R. Hoare, Communicating Sequential Processes, Communications of ACM, No 8, Vol 21, 1978.
3. A. Pawlina, Software Mechanisms for Inter-Process Communication in Class of Multiple Microprocessor Systems, Microprocessing and Microprogramming 7, 1981.
4. C. A. R. Hoare, Monitors: An Operating System Structuring Concept, Comm. ACM 17, 10 (1974).
5. P. Brinch-Hansen, Operating System Principles, Prentice-Hall, New Jersey, 1973.
6. V. Cvitaš, N. Hadjina, Modularni višeprocesorski mikro sistemi, 4. Međunarodni simpozij "Kompjuter na Sveučilištu, Cavtat, 1982.

SINTAKTIČKA ANALIZA JEZIKA SA SVOJSTVIMA

ZDRAVKO DOVEDAN

TVA KOV JNA, ZAGREB

UDK: 681.3.0.6:519.682.1

Jezik sa svojstvima je onaj u kojem rečenice, pored sintaktičke strukture, posjeduju i odredjena semantička svojstva. Na primjer, Algol 60 i Pascal su jezici sa svojstvima.
U ovom radu prezentirana je jedna metoda za provodjenje sintaktičke analize takvih jezika. Zasnovana je na primjeni prepoznavača u kojem je kontrola konačnog stanja komponovana od dijagrama prelaza i skupa akcija.

PARSING PROPERTY LANGUAGES. Property language is one in which, besides syntactic structure, sentences possess some semantics properties. For example Algol 60 and Pascal are property languages.
In this work a parsing method for property languages is presented. The method is based on use of a recognizer. The finite state control of the recognizer is composed of a transition diagram and set of actions.

1. UVOD

U teoriji formalnih jezika poznato je nekoliko metoda za provodjenje sintaktičke analize: s vrha, odozdo, upravljana tabelom, sa rekurzivnim spuštanjem, itd. Svaka od njih primjenjuje se za odredjene klase bezkontekstnih gramatika. Tako, na primjer, sintaktička analiza s vrha (top-down) može se primjeniti kod gramatika koje nisu lijevo rekurzivne, a sintaktička analiza sa rekurzivnim spuštanjem kod gramatika tipa LL(k).

Sintaktička analiza jezika generiranih tzv. "gramatikama sa svojstvima" malo je obradjena u literaturi. Jedan model, baziran na proširenju osnovne bezkontekstne gramatike tabelom svojstava, dan je u [2]. Medjutim, dosta je kompleksan i nepodesan za uklapanje u postojeće metode za provodjenje sintaktičke analize.

Postupak za provodjenje sintaktičke analize jezika sa svojstvima, kojeg ćemo izložiti u ovom radu, zasnovan je na primjeni prepoznavača jezika sa svojstvima. Kontrola konačnog stanja takvog prepoznavača dobivena je proširenjem znanja dijagrama prelaza (u odnosu na onaj koji se primjenjuje kod konačnih automata).

U prvom dijelu rada uveden je generator jezika sa svojstvima. Zatim je definiran prepoznavač takvog jezika. Na kraju je, na primjeru jednostavnog jezika cjelobrojnih aritmetičkih izraza, ilustrirana jedna od mogućih primjena predložene metode.

2. OSNOVNE DEFINICIJE I TERMINOLOGIJA

Zbog neusklađenosti terminologije, najprije uvodimo neke osnovne definicije teorije formalnih jezika, a prema [1] i [3].

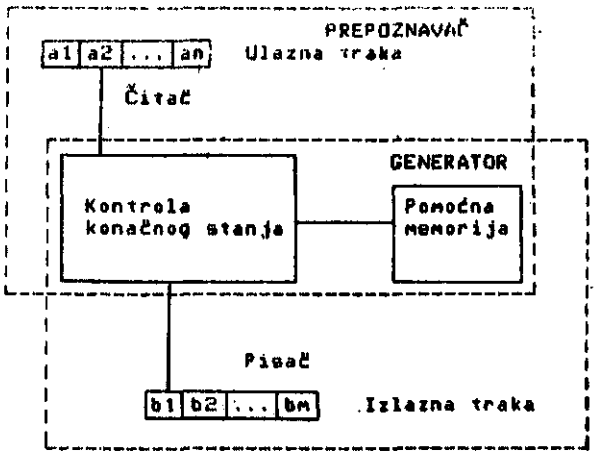
Alfabet A je konačan skup znakova. Znak je jedinstven, nedjeljiv element, kao što su: a,b, c,...,A,B,...,0,1,...,+,-,(,), itd. Ako se znakovi alfabeta A poredjaju jedan do drugog, dobije se niz (povorka) znakova. Definira se i niz koji u sebi ne sadrži nijedan znak i naziva se prazan niz. Označavat ćemo ga sa e. Dužina niza je broj znakova sadržanih u njemu. Prazan niz ima dužinu 0. Niz aa...a, koji sadrži n znakova a, piše se a^n. Specijalno je $a^0 = e$.

Često se u praksi promatraju nizovi znakova koji se mogu smatrati jedinstvenim nedjeljivim cjelinama. Takvi nizovi nazivaju se simboli ili riječi. Shodno ovoj definiciji i sam alfabet može se promatrati kao skup simbola dužine 1. Obično se simboli dijele u klase, kao na primjer: rezervirane riječi (**begin**, **if**, **continue**, itd), identifikatore (X, Area, Y2, itd.), imena funkcija (sin, cos, log, itd.), konstante (102, 9, 873, itd.) i specijalne simbole (+, -, =, /, *, itd.). Skup svih simbola definiranih nad alfabetom A označavat cemo sa V i nazivati rječnik. Sa V^* označavat ćemo skup svih nizova simbola nad V.

Jezik L nad rječnikom V je bilo koji podskup od V^*, tj. $L \subseteq V^*$. Najčešće je to beskonačan ili dosta veliki skup. Zbog toga, bilo bi teško i nepraktično definirati neki jezik navodeći eksplicitno sve njegove elemente. Jedna od najznačajnijih i najčešće primjenjivanih metoda za specificiranje jezika su gramatike. Slobodno govoreći, gramatika je skup pravila ("produkcija") koja opisuju sintaksu jezika, odnosno odredjuju koje povorke iz V^* pripadaju nekom jeziku L nad V. Postupak kojim se ispituje da li neka povorka w iz V^* pripada nekom jeziku L, koristeći pri tome pravila gramatike, naziva se sintaktička analiza. Povorka simbola w koja je u L naziva se rečenica.

Druga metoda za specificiranje jezika pretpostavlja proceduru koja na ulazu prihvata povorku w iz V^* i poslije konačno mnogo izračunavanja utvrdjuje da li je w element od L. Takva procedura (mašina, mehanizam) naziva se prepoznavač (recognizer), sl.2.1, a sastoji se od ula

zne trake, čitača, kontrole konačnog stanja i pomoćne memorije. Procedura koja umjesto ulazne trake i čitača ima izlaznu traku i pisač naziva se generator, a prepoznavač i generator zajedno nazivaju se pretvarač (transducer).



Sl. 2.1 - Pretvarač.

Specijalan slučaj pretvarača je konačni pretvarač. On nema pomoćne memorije, a kontrola konačnog stanja može se zadati dijagramom prelaza.

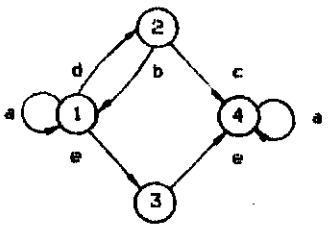
Definicija 2.1

Dijagram prelaza konačnog pretvarača je petorka $T = (Q,V,\Delta,q0,F)$, gdje su:

- Q — konačan skup stanja,
- V — rječnik; elemente rječnika (simbole) nazivat ćemo prelazi,
- Δ — funkcija prelaza: $\Delta: (Q,V) \to P(Q)$, gdje je $P(Q)$ particija skupa Q,
- $q0 \in Q$ — početno stanje i
- $F \subseteq Q$ — skup konačnih stanja.

Primjer 2.1

Neka je $Q = (1,2,3,4)$, $V = (a,b,c,d)$, $q0 = 1$ i $F = (4)$. Dijagram prelaza $T = (Q,V,\Delta,q0,F)$ predstavljen je sa:



	a	b	c	d	e
1	1			2	3
2		1	4		
3					4
4	4				

Značenje funkcije prelaza vidljivo je iz dijagrama.

♦

Dijagram prelaza može se predstaviti i u obliku tabele prelaza, kao što je pokazano u prethodnom primjeru.

3. GENERATOR JEZIKA SA SVOJSTVIMA

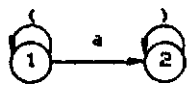
Generator u kojem je kontrola konačnog stanja u potpunosti zadana dijagramom prelaza T i koji, prema tome, nema pomoćne memorije, generira tzv. "regularan jezik", [1]. Drugim riječima, regularan jezik je onaj koji sadrži samo povorke $w = s1 \ldots sn$ iz V^* za koje vrijedi:

$$q1 = \Delta(q0,s1)$$
$$\vdots$$
$$qk = \Delta(qj,sn)$$

gdje su qi, $i=1,\ldots,k$, stanja iz Q, a $q0$ je početno i qk jedno od konačnih stanja, $qk \in F$.

Primjer 3.1

Generator zadan dijagramom prelaza:



generira jezik $L = \{ (^m a)^n : m \geq 0, n \geq 0 \}$.

♦

Kontrolu konačnog stanja generatora jezika sa svojstvima podjelit ćemo u dva dijela: dijagram prelaza i skup akcija. Dijagramom prelaza T zadaju se sve moguće promjene stanja, što je definirano funkcijom Δ. Akcija je postupak koji, u ovisnosti od tekućeg stanja qi i informacija dobivenih od pomoćne memorije, vrši restrikciju mogućih prelaza iz stanja qi, što je zadano funkcijom prelaza Δ, u trenutno ostvarive prelaze. Ako je prelaz iz stanja qi u stanje qj prelazom sk uvijek ostvariv, smatrat ćemo da je takvom prelazu pridružena prazna akcija. Generator regularnih jezika primjer je u kojem su uvijek svi prelazi ostvarivi (otuda generator regularnih jezika nema potrebe za pomoćnom memorijom).

Definicija 3.1

Jezik sa svojstvima je jezik čiji je generator zadan sa:

- kontrolom konačnog stanja, koja se sastoji od dijagrama prelaza $T = (Q,V,\Delta,q0,F)$, i skupom akcija pridruženih svakom paru (qi,sj), $qi \in Q$, $sj \in V$ za koji je definirana funkcija prelaza Δ i barem jedna akcija je neprazna,
- pomoćnom memorijom.

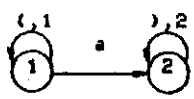
♦

Kontrolu konačnog stanja jezika sa svojstvima i dalje ćemo predstavljati dijagramom prelaza, ali ćemo svakom prelazu dodati kôd njemu pridružene akcije sa značenjem:

- 0 (ili izostavljeno) - prazna akcija,
- 1,2,... - neprazna akcija.

Primjer 3.2

Generator jezika $L = \{(^n a)^n : 0 \leq n \leq 10\}$ ima dijagram prelaza:



gdje akcija 1 znači: "brojati prelaze sa "(""; a akcija 2: "rečenica jezika je generirana kada se načini onoliko prelaza sa ")" koliko je bilo načinjeno sa "(""". Pomoćna memorija će sadržava ti informaciju o broju načinjenih prelaza sa "("; Generator ovog jezika može se zadati slije dećim algoritmom:

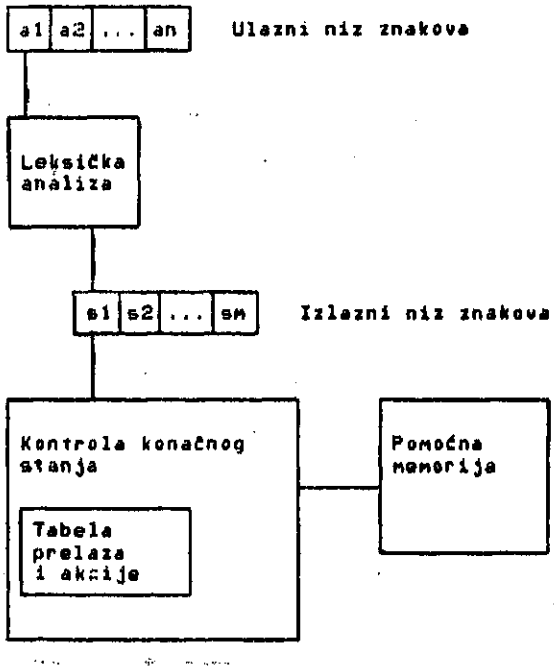
```
{ Generiranje jezika L={a,(a),((a)),...} };
br := 0; brmax := 10;
while br < brmax do
  begin
    i := 1;
    while i < br do
      begin
        string(i) := "("; i := i+1
      end;
      string(i) := "a";
      i := 1;
      while i < br do
        begin
          string(i) := ")"; i := i+1
        end;
        write(string); br := br+1
  end
```

4. PREPOZNAVAČ JEZIKA SA SVOJSTVIMA

Kod generatora jezika sa svojstvima akcijom je za svako stanje dijagrama prelaza bio odredjen skup ostvarivih prelaza, što je u svakom trenutku bilo uvjetovano informacijama dobivenim od pomoćne memorije.

U slučaju prepoznavača jezika sa svojstvima postavljamo pitanje: "Da li za simbol sj ∈ V postoji ostvariv prelaz iz tekućeg qi u neko stanje qk?". Prvi dio odgovora na ovo pitanje dat će nam dijagram prelaza. Ako je qk=δ(qi,sj) definirano, prelaz je moguć, a akcija pridružena paru (qi,sj) odredit će da li je ostvariv.

Model prepoznavača jezika sa svojstvima dan je na sl. 4.2:



Sl. 4.2 – Prepoznavač jezika sa svojstvima.

Vidimo da prepoznavaču prethodi jedan pretvarač (leksička analiza) koji prihvata povorku znakova na ulazu i na izlazu daje povorku simbola, odnosno povorku cjelobrojnih kodova pridruženih simbolima jezika i klasama identifikatora, što su istovremeno stupci tabele prelaza prepoznavača.

Ako je sym kod učitanog simbola, state tre-nutno tekuće stanje kontrole konačnog stanja, moguć prelaz zadan je sa:

```
trans = ^(state,sym)
```

a akcija kodom na mjestu (state,sym) tabele akcija. Tada se kontrola konačnog stanja prepoznavača jezika sa svojstvima može općenito napisati kao:

```
eoi := false; state := 1;
while not eoi do
  begin
    { sym: get sym(string);
      trans: trans tab(state,sym);
      action: act tab(state,sym) };
    if trans = 0
       then write("Synt. error");
       eoi := true
    else if action = 0
            then state := trans
         else { Izvodi se akcija sa
                kodom "action" }
  end
```

Ovdje je "get sym" leksička analiza, a "trans tab" i "act tab" tabele prelaza i akcije. Program terminira ili ako dosegne jedno od konačnih stanja, što je zadano jednom od nepraznih akcija, ili pri otkrivanju sintaktičke greške.

5. PRIMJER

Treba provesti sintaktičku analizu jezika sa svojstvima cjelobrojnih aritmetičkih izraza danih skupom produkcija (napisanih u BNF-u, [1]):

```
⟨izraz⟩ ::= ⟨term⟩|⟨term⟩+⟨term⟩
  ⟨term⟩ ::= ⟨faktor⟩|⟨term⟩*⟨faktor⟩|
    ⟨faktor⟩ ::= ⟨p-varijabla⟩|⟨konstanta⟩|
                 ⟨a-varijabla⟩(⟨izraz⟩)|
                 ⟨a-varijabla⟩.⟨atribut⟩|
                 (⟨izraz⟩)
      ⟨atribut⟩ ::= dom|lob|hib|low|high
```

gdje su ⟨p-varijabla⟩ i ⟨a-varijabla⟩ identifikatori primitivne i varijable sa strukturom polja (array), respektivno. Primjer poverke u ovom jeziku je: a(a.lob+1)*b.

Pretpostavka je da se leksičkom analizom odredjuje svojstvo identifikatora, a poverke znamenki dobija svojstvo konstante. Generator jezika definiranog gornjom gramatikom ima slijedeću tabelu prelaza:

	c	p	a	.	+	*	(	)	at	;
1	2	2	4		3		1 1			
2					3	3		2 2		1 3
3	2	2	4				1 1			
4				5			1 1			
5									3	

gdje su:

c	konstante,
p	primitivne varijable,
a	array varijable
at	atributi i
;	simbol koji označava kraj izraza.

U istoj tabeli dani su i kodovi akcija (kao eksponenti). Na mjestima gdje su izostavljeni podrazumjeva se da su jednaki 0.

Program za provodjenje sintaktičke analize dan je sa:

```
eoi := false; synter := false;
state := 1;
while not (eoi V synter) do
  begin
    { sym: get sym(string);
      trans: trans tab(state,sym);
      action: act tab(state,sym) };
    if trans = 0
       then write( "Synt. error");
       synter := true
    else
    if action = 0 then state := trans
    else
    if action = 1
       then ibr := ibr+1; state := trans
    else
    if action = 2
       then
       if ibr > 0 then ibr := ibr-1;
                       state := trans
       else write("Synt.error");
            synter := true
    else
    if action = 3
       then
       if ibr = 0 then eoi := true
       else write("Synt.error");
            synter := true;
  end;
  if synter
     then write("Input string is not in L")
  else write("Input string is in L")
```

6. ZAKLJUČAK

Predložena metoda provodjenja sintaktičke analize praktično je bila primjenjena u realizaciji Dijkstrinog jezika, prezentiranog u [4], koji spada u klasu jezika sa svojstvima.

Na kraju treba dati odgovor na pitanje: Koja su ograničenja u primjenljivosti predloženog postupka? Za sada možemo reći da je to mogućnost sprovodjenja direktne leksičke analize. Drugim riječima, da bi se predloženi postupak mogao primjeniti u sintaktičkoj analizi nekog jezika, neophodno je da svi simboli jezika imaju jedinstveno značenje, tj. učitavanjem nekog simbola ne smije postojati neizvjesnost da li je učitana rezervirana riječ, identifikator ili ime neke funkcije.

Literatura

1. AHO V.A., ULLMAN D.J.: "The Theory of Parsing, Translation and Compiling", Vol.I: Parsing, Prentice-Hall, 1972.
2. - " - : "The Theory of Parsing, Translation and Compaling", Vol. II: Compiling, Prentice-Hall, 1973.
3. - " - : "Principles of Compiler Design", Addison-Wesley Publishing Company, 1977.
4. DIJKSTRA E.W.: "A Discipline of Programming", Prentice-Hall, 1976.

SPREMLJANJE RAČUNALNIŠKIH SISTEMOV V REALNEM ČASU

B. KASTELIC,
S. KLANČAR,
Š. URANKAR

UDK: 681.326.0

INSTITUT „JOŽEF STEFAN", JAMOVA 39, LJUBLJANA
ISKRA–TELEMATIKA, TOZD ATC, KRANJ

Članek opisuje spremljanje (monitoring) delovanja računalniških sistemov v realnem času. V splošnem je opisano zunanje in notranje spremljanje računalniških sistemov. Zunanje spremljanje omogoča dodaten spremljevalni računalnik, ki preko posebnega povezovalnega vezja spremlja delovanje osnovnega računalnika. Notranje spremljanje je realizirano z nekaj dodatne materialne in programske opreme v samam računalniku. Deluje na principu dodeljevanja računalniškega časa delovnim in spremljevalnim programom. Opisan je primer notranjega spremljanja računalnika v realnem času v telefonski centrali SI 2000.

REAL TIME MONITORING OF COMPUTER SYSTEMS - The paper describes in general real time external and internal monitoring of computer systems. External monitoring is performed by use of other monitor computer, which probes the operation of the examined computer system. Internal monitoring is performed with built-in hardware and software in the examined computer system. This monitoring is executed by apportion of computer time to the working and monitoring programs. The performing of internal monitoring is illustrated by monitoring of computer controlled telephone exchange System ISKRA 2000.

1. UVOD

Razmah računalniških sistemov, ki delujejo v realnem času (take računalnike lahko imenujemo tudi sprotni računalniki) je botroval razvoju orodja, ki omogoča spremljanje (monitoring) materialne, predvsem pa programske opreme računalniškega sistema med delovanjem le-tega v realnem času.

Sprotni računalniški sistemi imajo običajno vgrajene mehanizme, ki ob detekciji napake avtomatsko sprožijo obnovitev sistema. Take sisteme prištevamo med sisteme, ki so neobčutljivi na napake (fault tolerance). Toda poleg omenjenih mehanizmov, mora imeti tudi operater možnost spremljanja in poseganja v delovanje sistema. Namen našega članka ni opisati delovanje sistema neobčutljivega na napake, ampak bomo govorili o spremljanju in poseganju operaterja v tak sprotni sistem, seveda medtem ko sistem svojo funkcijo v okolju na katerega je priključen normalno opravlja. Taka orodja omogoču operaterju oziroma razvijalcu programske opreme neposredno kontrolo delovanja sistema v fazi razvoja in pozneje nadzor nad sistemom ob lokalnih okvarah, ki jih sistem sam ni sposoben odpraviti niti javiti.

2. SPREMLJANJE RAČUNALNIŠKIH SISTEMOV

Zahteve neobčutljivosti sistema na napake ter možnost spremljanja in poseganja operaterja v sprotni računalniški sistem so zelo ozko povezane. Te zahteve so lahko realizirane na dva načina. Računalnik lahko sam z nekaj dodatne materialne in programske opreme omogoča operaterju spremljanje svojega delovanja v realnem času, ali pa obstaja poleg osnovnega računalniškega sistema še neki dodaten spremljevalni računalnik, ki nadzira delovanje in prav tako omogoča operaterju spremljanje osnovnega sistema. Govorimo o notranjem in zunanjem spremljanju računalniških sistemov.

Za zunanje spremljanje računalnika je torej značilno, da imamo poleg osnovnega računalnika (object machine) še dodatni spremljevalni računalnik (monitor machine), ki je povezan preko posebnega povezovalnega vezja z osnovnim računalnikom. Spremljevalni računalniki so praviloma mnogo manjši od osnovnega sistema in opravljajo funkcijo vzdrževalnega procesorja z naborom spremljevalnih funkcij za osnovni sistem. Razen tega lahko delujejo kot periferni procesor, komunikacijski procesor, diagnostični procesor (ne v realnem času) in kot povezovalna enota s centralnim diagnostičnim računalnikom.

Zunanje spremljanje računalniških sistemov je običajno uporabljeno pri računalniških sistemin, kjer sta zelo pomembni zanesljivost in hitrost. Pri sistemih, kjer si lahko privoščimo uporabo dela računalniškega časa za spremljanje samega računalnika, ob tem da se računalnik še vedno v dovoljenih časovnih zakasnitvah odziva okolju s katerim komunicira, lahko uvedemo notranje spremljanje. To je realizirano na principu preklapljanja računalniškega časa med delovnimi programi in programi, ki omogočajo

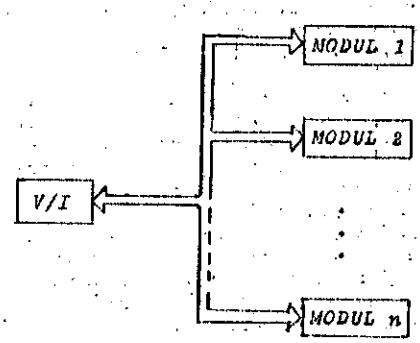
spremljanje računalnika. Te programe bomo krajše imenovali spremljevalni programi. Pri sunanjem spremljanju računalnika smo napravo, ki je omogočala spremljanje, imenovali spremljevalni računalnik, pri notranjem pa jo bomo spremljevalna konsola.

V nadaljevanju članka bomo opisali primer notranjega spremljanja računalnika na Iskrinem mikroračunalniku TK6800 s poudarkom na možnostin, ki jih ima operater. Realisacija neobčutljivosti na napake istega računalnika je bila že opisana v članku (3).

3. SPREMLJEVALNA KONZOLA

Mikroračunalnik TK6800 je grajen modularno in je sestavljen iz več plošč (CPU, RAM, ROM,...), ki skupaj s specialnimi telefonskimi ploščami tvorijo en modul telefonske centrale SI 2000. Spremljanje računalnika pa omogoča vhodno/izhodna enota (CRT, TTY) in dodatno vezje, ki se nahaja na posebni plošči imenovani CSLT. To skupaj imenujemo spremljevalna konsola. CSLT plošča je na eni strani priključena na sistemsko vodilo računalnika na drugi pa na vhodno/izhodno enoto spremljevalne konsole, oziroma na konsolno vodilo. Konsolno vodilo omogoča spremljanje vseh modulov telefonske centrale preko ene V/I enote.

Programska in materialna oprema za spremljanje računalnika v realnem času se nahaja v vseh modulih, skupna je edino vhodno/izhodna enota. Moduli so serijsko vezani preko ACIA vmesnikov na V/I enoto. To pomeni, da vsak poslani znak s V/I enote sprejmejo vsi moduli povezani na konsolno vodilo. Na poslani znak se vedno odzove samo tisti modul, s katerim je vzpostavljena povezava. V določenem trenutku je vzpostavljena komunikacijska pot s samo enim modulom. Ostali moduli sicer sprejemajo oddajane znake s V/I enote, vendar se nanje ne odzivajo. Hitrost prenosa podatkov po konzolnem vodilu je 300 bitov/sekundo.



SLIKA 1: Moduli so povezani preko konzolnega vodila s vhodno/izhodno enoto.

Vsak modul je lahko v dveh fazah delovanja. V prvi je takrat, ko komunikacijska pot med modulom in V/I enoto spremljevalne konzole ni vzpostavljena. Modul v tej fazi samo pričakuje ukaz za preklop v drugo fazo delovanja. V trenutku ko se komunikacijska pot med modulom in V/I enoto vzpostavi, preide modul v drugo fazo delovanja in izpiše na V/I enoti svojo identifikacijsko oznako in sporočilo o stanju modula (RUN, HALT, TEST,...). Operater lahko šele sedaj uporablja spremljevalne funkcije za spremljanje poteka delovnih programov, ki tečejo v dotičnem modulu.

```
L01 RUN

.<
```

SLIKA 2: Izhodno sporočilo na V/I enoti spremljevalne konsole po vzpostavitvi povezave z modulom L01.

3.1. SPREMLJEVALNE FUNKCIJE

Spremljevalne funkcije omogočajo operaterju spremljanje materialne in programske opreme modula (mikroračunalnika) in poseganje v izvajnje delovnih programov v modulu. Obstajata dve vrsti spremljevalnih funkcij, ki tečejo na dveh različnih nivojih. Istočasno se lahko izvajata dve spremljevalni funkciji in sicer ena na prvem in druga na drugem nivoju.

S spremljevalnimi funkcijami prvega nivoja spremljamo oziroma sledimo na več načinov potek delovanja programov in spreminjanje podatkov na določenih lokacijah pomnilnika. Te funkcije so vezane na delovanje CSLT plošče in so neposredno odvisne od prekinitev, ki jih generira prekinitveno vezje te plošče.

- Sledenje programa je realizirano s pomočjo vezja na CSLT plošči, ki omogoča shranjevanje podatkov z adresnega in podatkovnega vodila v posebni sledilni pomnilnik. Programsko je potrebno nastaviti le ustrezne registre CSLT plošče in na konou izpisati rezultate sledenja iz sledilnega pomnilnika. V sledilnem pomnilniku je prostora za 128 podatkov z adresnega in podatkovnega vodila. Ločimo tri vrste in dva načina sledenja programa. Sledimo lahko vse veljavne podatke na obeh vodilih, samo naslove in operacijsko kodo instrukcij ali pa samo naslovo in operacijsko

```
PAGE  001

00001                    *
00002                    * EXAMPLE
00003                    *


00005           S12A   A MEMORY EQU     $1[illegible]

00007A 4200                      ORG     $4200

00009A 4200 8E 4CF0   A BEGIN    LDS     #$4CF0
00010A 4203 4F                   CLRA
00011A 4204 5F                LOOP1 CLRB
00012A 4205 CE 612A   A          LDX     #MEMORY
00013A 4208 A7 00     A LOOP2    STAA    0,X
00014A 420A 37                   PSHB
00015A 420B C6 45     A          LDAB    #'E
00016A 420D F7 612A   A          STAB    MEMORY
00017A 4210 8D 0C 421F           BSR     OUT
00018A 4212 33                   PULB
00019A 4213 08                   INX
00020A 4214 5C                   INCB
00021A 4215 C1 03     A          CMPB    #3
00022A 4217 26 FF 4208           BNE     LOOP2
00023A 4219 8E 4FF0   A          LDS     #$4FF0
00024A 421C 20 E6 4204           BRA     LOOP1

00026A 421E 01                OUT    NOP
00027A 421F 39                   RTS
00028                            END
TOTAL ERRORS 00000
```

SLIKA 3: Delovni program na katerem so bili narejeni podani primeri spremljevalnih funkcij.

kode skočnih instrukcij (JMP, BRA, JSR, RTS,...). Sledimo na dva načina in sicer 128 podatkov pred prekinitvenim mestom ali pa 128 podatkov po prekinitvenem mestu. Po zaključku sledenja se rezultati sledenja izpišejo na V/I enoti. V kolikor je bila povezava s dotičnim modulom prekinjena, ali če teče v modulu funkcija drugega nivoja, se nastavi ustrezna zastavica, ki povzroči ob ponovni vzpostavitvi komunikacijske poti z modulom ali ob zaključku funkcije drugega nivoja, izpis sporočila na V/I enoti, da so v sledilnem pomnilniku že pripravljeni rezultati sledenja.

```
.<T1 4208
    ADDR CODE  T1 BP=4208
0D  4FFE  42
0C  4FEF  12
0B  4212  33
0A  4213  08
09  4FF0  01
08  4213  08
07  4214  5C
06  4214  5C
05  4215  C1
04  4215  C1
03  4216  03
02  4217  26
01  4218  EF
00  4208  A7  D5 02 00 612C 420A 4FF0
.<
```

SLIKA 4: Sledenje zadnjih 128 veljavnih podatkov na obeh vodilih do prekinitvenega mesta. Izpisanih je samo zadnjih 14 podatkov. V zadnji vrstici so izpisane še vsebine registrov mikroprocesorja na prekinitvenem mestu.

```
.<T4 4208
    ADDR CODE  T4 BP=4208
00  420A  37
01  420B  C6
02  420D  F7
03  4210  8D
04  421E  01
05  421F  39
06  4212  33
07  4213  08
08  4214  5C
09  4215  C1
0A  4217  26
0B  4219  8E
0C  421C  20
0D  4204  5F
0E  4205  CE
    ADDR CODE  T4 BP=4208
0F  4208  A7
10  420A  37
11  420B  C6
12  420D  F7
13  4210  8D
14  421E  01
15  421F  39
16  4212  33
17  4213  08
18  4214  5C
19  4215  C1
1A  4217  26
1B  4208  A7
1C  420A  37
1D  420B  C6
.<
```

SLIKA 5: Sledenje naslovov in operacijskih kod 128 instrukcij od prekinitvenega mesta naprej. Na sliki je zadnjih 30 podatkov.

- *Funkcija pogojnega sledenja točke programa ali podatka sledi dogajanje na določenem naslovu, to je na prekinitvenem mestu. Pred izvajanjem te funkcije lahko nastavimo zakasnitev sledenja, to je število prehodov preko prekinitvenega mesta preden se starta sledenje in pogoje ob katerih se starta sledenje. V vmesnem pomnilniku, kamor se shranjujejo rezultati sledenja, je prostora za največ 100 prehodov. Ob vsakem veljavnem prehodu preko prekinitvenega mesta se shrani v vmesni pomnilnik naslov in koda instrukcije, ki je operirala s prekinitvenim naslovom, vsebine registrov mikroprocesorja in še vsebine največ petih lokacij pomnilnika, katerih naslove predhodno nastavimo. Po zaključku spremljanja se rezultati, v kolikor je ostala povezava s dotičnim modulom vzpostavljena, prepišejo iz vmesnega pomnilnika na V/I enoto spremljevalne konzole.*

```
.<T 612A 9
.<
    ADDR CODE C  P  A   X    PC   SP  612A  BP=612A D=0000
00  4208  A7  D4 00 00 612A 420B 4FEF  00
01  420D  F7  I0 45 00 612A 421E 4FED  45
02  420D  F7  D1 45 00 612B 421E 4FED  45
03  420D  F7  D1 45 00 612C 421E 4FED  45
04  4208  A7  D4 00 00 612A 420B 4FEF  00
05  420D  F7  D0 45 00 612A 421E 4FED  45
06  420D  F7  D1 45 00 612B 421E 4FED  45
07  420D  F7  D1 45 00 612C 421E 4FED  45
08  4208  A7  D4 00 00 612A 420B 4FEF  00
.<
```

SLIKA 6: Pogojno sledenje podatka na naslovu s612A, 9 prehodov preko prekinitvenega mesta, brez pogojev in nakasnitve sledenja ter s izpisom podatka na naslovu s612A.

- *Funkcija sledenja programa po posamesnih instrukcijah ne omogoča delovanje računalnika v realnem času. Uporablja se lahko samo takrat, ko so delovni programi v neki točki prekinjeni. Omogoča nam, da izvedemo samo eno ali več naslednjih instrukcij delovnega programa. Na V/I enoti spremljevalne konzole se izpiše naslov in operacijska koda izvedene instrukcije, vsebina registrov mikroprocesorja po izvedbi instrukcije in nastavljeni podatki.*

```
*<N 15
ADDR CODE C  B  A   X    PC   SP  612A
421E  01  D1 45 00 612C 421F 4CED  45
421F  39  D1 45 00 612C 4212 4CEF  45
4212  33  D1 02 00 612C 4213 4CF0  45
4213  08  D1 02 00 612D 4214 4CF0  45
4214  5C  D1 03 00 612D 4215 4CF0  45
4215  C1  D4 03 00 612D 4217 4CF0  45
4217  26  D4 03 00 612D 4219 4CF0  45
4219  8E  D0 03 00 612D 421C 4FF0  45
421C  20  D0 03 00 612D 4204 4FF0  45
4204  5F  D4 00 00 612D 4205 4FF0  45
4205  CE  D0 00 00 612A 4208 4FF0  45
4208  A7  D4 00 00 612A 420A 4FF0  00
420A  37  D4 00 00 612A 420B 4FEF  00
420B  C6  D0 45 00 612A 420D 4FEF  00
420D  F7  D0 45 00 612A 4210 4FEF  45
ADDR CODE C  B  A   X    PC   SP  612A
4210  8D  D0 45 00 612A 421E 4FED  45
421E  01  D0 45 00 612A 421F 4FED  45
421F  39  D0 45 00 612A 4212 4FEF  45
4212  33  D0 00 00 612A 4213 4FF0  45
4213  08  D0 00 00 612B 4214 4FF0  45
4214  5C  D0 01 00 612B 4215 4FF0  45
*<
```

SLIKA 7: Ukaz za sledenje programa po posamesnih instrukcijah definira izvajanje naslednjih 31 instrukcij delovnega programa, nakar se delovni program zopet prekine.

Spremljevalne funkcije drugega nivoja večinoma prav tako ne ovirajo delovanja računalnika v realnem času. Omogočajo nam:

- nastavitev pogojev za funkcijo pogojnega sledenja točke programa ali podatka,
- nastavitev naslovov podatkov za funkciji pogojnega sledenja točke programa ali podatka in za sledenje programa po posamesnih instrukcijah,
- nastavitev zakasnitve sledenja,
- izpis/spremembo pomnilne lokacije,
- izpis vsebine pomnilnika, vmesnega pomnilika ali sledilnega pomnilnika,
- izpis simbola spremljevalne funkcije prvega nivoja, ki je v teku.

Poleg teh obstajajo še spremljevalne funkcije drugega nivoja, ki ne omogočajo delovanja računalnika v realnem času. Uporabljajo se lahko le takrat, ko se delovni programi ne izvajajo.

3.2. POSEBNA VHODNA SPOROČILA

Ko je z modulom vzpostavljena povezava, se ta odzove v vsakem trenutku na posebno vhodno sporočilo. Posebno vhodno sporočilo predstavlja en kontrolni znak in ima višjo prioriteto kot spremljevalne funkcije drugega nivoja, ki jih vedno prekine. Ta vhodna sporočila nam omogočajo prekinitev povezave z modulom, restart modula, prekinitev izvajanja spremljevalne funkcije prvega nivoja, ki je v teku ter ustavitev in ponovni start delovnih programov.

4. MATERIALNA OPREMA

CSLT plošča omogoča povezavo med V/I enoto in računalnikom ter spremljanje oziroma sledenje programov, ki tečejo v računalniku. Važnejši deli:

- prekinitveno (break point) vezje. Generira prekinitev (TRAP 0), ko se pojavi na adresnem vodilu znak naslov kot je v prekinitvenem registru.

- sledilni pomnilnik velikosti 3x128 zlogov. Vanj se shranjujejo rezultati sledenja.

- števni sistem za naslavljanje sledilnega pomnilnika

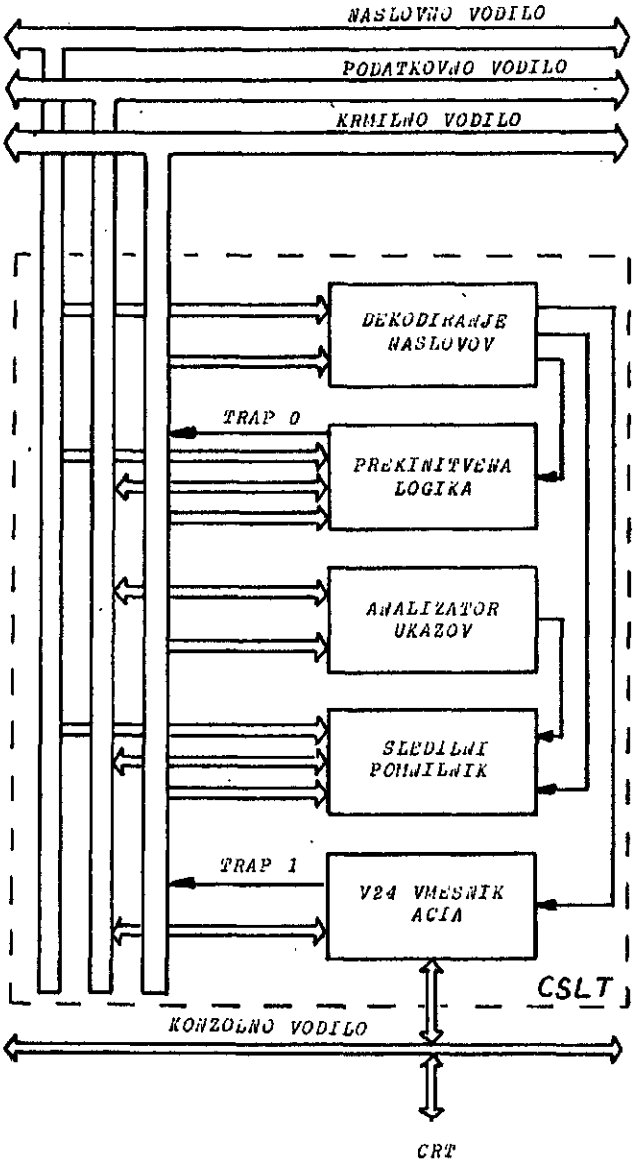
- analizator operacijskih kod, ki omogoča shranjevanje določenih podatkov v sledilni pomnilnik.

- V24 vmesnik z ACIA vezjem za komunikacijo z V/I enoto. Prekinitvena zahteva ACIA vezja, ki se generira za vsak sprejeti in oddani znak, je vezana na past TRAP 1.

Sledenje programa je omogočeno tako, da se med izvajanjem programa zapisuje vsebina adresnega in podatkovnega vodila neposredno brez sodelovanja procesorja v sledilni pomnilnik. Delovanje računalnika ni moteno. Po končanem sledenju se lahko vsebina sledilnega pomnilnika prešita. Obstajajo naslednje možnosti sledenja:

- sledenje programa do prekinitvenega (break point) naslova
- sledenje programa od prekinitvenega naslova
- sledenje ciklov na prekinitvenem naslovu
- iskanje neobstoječih operacijskih kod.

Pri sledenju do prekinitvenega naslova se vsebina adresnega in podatkovnega vodila shranjuje v sledilni pomnilnik dokler program ne naleti na prekinitveni naslov. Pri sledenju od prekinitvenega naslova se sledenje starta, ko program naleti na prekinitveni naslov in se prekine, ko je sledilni pomnilnik poln.



SLIKA 8: Bločna shema CSLT plošče, ki je na eni strani priključena na sistemsko vodilo računalnika, na drugi pa na konsolno vodilo.

Ločimo tri vrste sledenja:

- sledenja vseh veljavnih podatkov vključno z DMA cikli na obeh vodilih,
- sledenje samo operacijskih kod ukazov,
- sledenje samo operacijskih kod skočnih ukazov.

5. PROGRAMSKA OPREMA

Omenili smo že, da je notranje spremljanje računalnika v realnem času realizirano na principu preklapljanja računalniškega časa med delovnimi in spremljevalnimi programi. Spremljevalne programe torej izvaja isti procesor kot aplovne, zato le-ti zahtevajo svoj del spomina (ROM in RAM) v samem mikroračunalniku. V bistvu so spremljevalni programi servisne rutine za

prekinitve TRAP0 in TRAP1, ki jih generira CSLT ploščа.

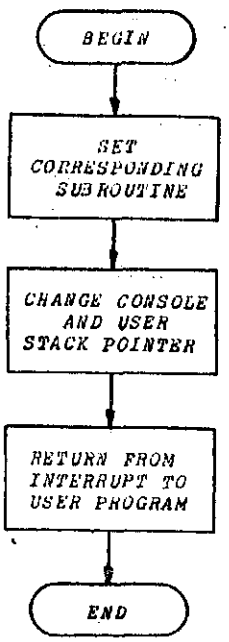
Dodeljevanje računalniškega časa spremljevalnim in delovnim programom v modulu je zasnovano na osnovi prekinitev, ki jih generira ACIA spremljevalne konzole. Pri hitrosti 300 bitov/sekundo se prekinitve generirajo, ob oddajanju znakov na V/I enoto, približno vsakih 33ms. V 33ms smejo spremljevalni programi uporabiti za svoje izvajanje največ 1ms računalniškega časa.

Z vhodno/izhodno enoto spremljevalne konzole komunicirajo samo spremljevalni programi. Delovni programi bi načeloma lahko komunicirali preko lastne V/I enote, vendar je v telefonskih centralah ne potrebujejo.

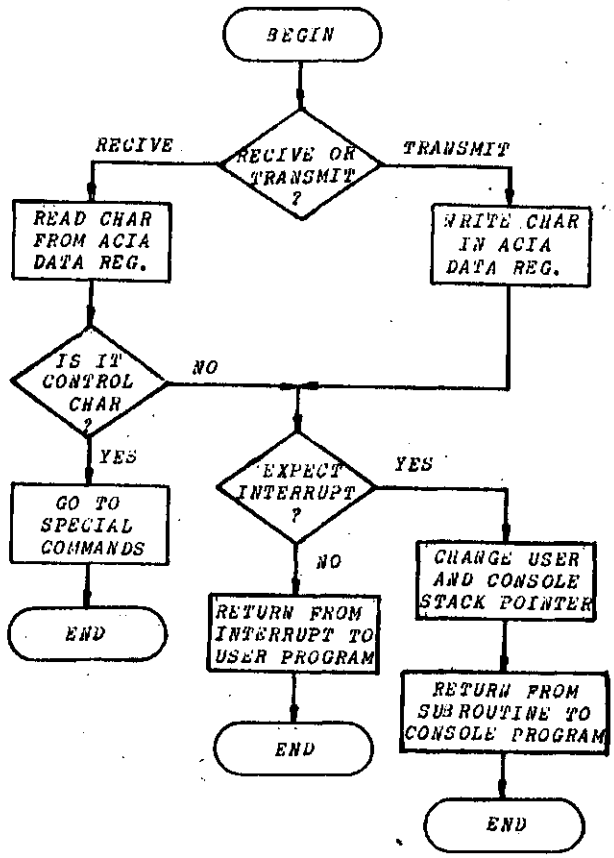
Jedro spremljevalnega programa opravlja naslednje naloge: preklaplja modul v posamezne faze delovanja in jih inicializira, omogoča dodeljevanje računalniškega časa spremljevalnim in delovnim programom, izpisuje izhodna sporočila o stanju modula, sprejema vhodna sporočila, išče naslove spremljevalnih funkcij v ukazni tabeli in omogoča nadzor nad njihovim izvajanjem.

Vsak modul je lahko v dveh fazah delovanja. V prvi fazi je takrat, ko komunikacijska pot med modulom in V/I enoto spremljevalne konzole ni vspostavljena. V tem primeru je ACIA spremljevalne konzole programirana tako, da je omogočena samo sprejemna prekinitev (receiver interrupt). Vsak sprejeti znak povzroči prekinitev delovnih programov. Servisna rutina sprejeti znak primerja s "control A". Če znak ni "control A" se izvajanje delovnih programov nadaljuje, sicer pa se nastavi primerjava naslednjih treh znakov (tip in dve heksadecimalni številki). V kolikor se oznaćba modula ujema s vpisanimi znaki, preide modul v drugo fazo delovanja. V prvi fazi se prekinjajo delovni programi za zelo kratek čas, saj je potrebno za izvajanje servisne rutine le 50us, pa tudi prekinitve, ki jih generirajo vpisani znaki, se pojavljajo v bistveno daljših intervalih od 33ms.

V drugi fazi delovanja je modul, ko je komunikacijska pot med modulom in V/I enoto spremljevalne konzole vspostavljena. V tej fazi ne moreta biti istočasno dva ali več modulov, povezanih na istó konzolno vodilo. ACIA spremljevalne konzole je programirana tako, da sta omogočeni sprejemna in oddajna prekinitev, ki prekinjata izvajanje delovnih programov. Oddajna prekinitev (transmit interrupt) se generira vsakih 33ms. Ta povzroči izvajanje servisne rutine in s tem preklop na spremljevalne programe. Oddajna prekinitev se servisira tako, da se v podatkovni register ACIA spremljevalne konzole zapiše znak, ki se bo oddal oz. izpisal na V/I enoto. V kolikor ni nobenega znaka za izpis, se vpiše v podatkovni register ničla ($00). Ta se ravno tako odda V/I enoti, vendar se na ekran ne izpiše ničesar. Po servisiranju oddajne prekinitve ostane približno 0,8 ms računalniškega časa za izvajanje spremljevalnega programa. Najkasneje po preteku tega časa se



SLIKA 9: Diagram poteka za V/I podprograme spremljevalnega programa. Po nastavitvi ustrezne rutine se zamenjata spremljevalni in delovni sklad. Ob povratku iz prekinitve se začne izvajati delovni program.



SLIKA 10: Diagram poteka za vhodno/izhodno servisno rutino. Ob prekinitvi, ki jo generira ACIA spremljevalne konzole (TRAP 1), se prečita iz podatkovnega registra podatek, če se je generirala sprejemna prekinitev, oziroma se zapiše v podatkovni register podatek, če se je generirala oddajna prekinitev. V primeru, da je bila prekinitev pričakovana, se zamenjata delovni in spremljevalni sklad in ob povratku iz podprograma se začne izvajati spremljevalni program. V primeru, da prekinitev ni bila pričakovana, se ob povratku iz prekinitve nadaljuje izvajanje delovnih programov. Pričakovana prekinitev pomeni, da se je pred prekinitvijo izvršila v spremljevalnem programu rutina, ki čaka na to prekinitev.

mora izvajanje programa preklopiti na delovni program. Vrnitev v delovne programe je realizirana na tri načine:

1. Spremljevalni program se prenehа izvajati, ko le-ta kliče podprogram za izpis znaka (OUTCH) na V/I enoto spremljevalne konsole. Izvajanje spremljevalnega programa se nadaljuje po približno 32us, ko se zaradi oddajne prekinitve prekine izvajanje delovnih programov.

2. Spremljevalni program se prenehа izvajati, ko le-ta kliče podprogram za sprejem znaka (INCH) z V/I enote spremljevalne konsole. Izvajanje spremljevalnega programa se nadaljuje šele takrat, ko se generira sprejemna prekinitev. Ker se med tem običajno nekajkrat generira oddajna prekinitev, se ob servisiranju te odda nič, izvajane pa se takoj vrne v delovne programe.

3. Spremljevalni program se prenehа izvajati, ko le-ta kliče podprogram za izpis ničle (SLEEP) na V/I enoto spremljevalne konsole. Na V/I enoti se ne izpiše ničesar, izvede se samo preklop na delovne programe. Ta način preklopa je uporabljen takrat, ko bi izvajanje spremljevalnega programa preseglo dovoljeno 1ms.

5.1. SPREMLJEVALNE FUNKCIJE

Spremljevalne funkcije prvega nivoja omogočajo spremljanje modula v realnem času. Neposredno so odvisne od pasti nič (TRAP 0), ki jo generira prekinitveno (Break point) vezje na CSLT plošči modula. Te funkcije se izvajajo v dveh korakih. Najprej se inicializira delovanje CSLT plošče in nastavi ustrezna servisna rutina na past 0. Ta inicializacija teče kot funkcija drugega nivoja in se praktično v trenutku izvrši. Drugi korak je servisiranje pasti 0, katero generira prekinitveno vezje. Servisiranje poteka dejansko neopazno, saj lahko operater med tem še vedno nadzoruje delovanje modula s funkcijami drugega nivoja. Če je povezava z modulom ostala vspostavljna in če ne teče istočasno v modulu funkcija drugega nivoja, se ob zaključku funkcije prvega nivoja resultati takoj izpišejo na V/I enoti spremljevalne konsole.

Spremljevalne funkcije drugega nivoja niso odvisne od nobenih "hardverskih" pogojev. Dejansko predstavljajo le servisiranje prekinitve (TRAP 1), ki jo generira ACIA spremljevalne konsole. Resultati teh funkcij se izpišejo takoj po startu.

5.2. DELOVNI (UPORABNIŠKI) PROGRAMI

Delovni programi so napisani neodvisno od spremljevalnih programov, vendar morajo vsekakor upoštevati nekatere prepovedi:

- prepovedan imajo dostop na področja pomnilnika (ROM in RAM), ki ga uporabljajo spremljevalni programi
- prepovedan imajo dostop do prekinitvenih vektorjev, ki jih uporabljajo spremljevalni programi
- prepovedan imajo dostop do vseh registrov CSLT plošče, vključno z registri ACIA spremljevalne konsole

6. SKLEP

Notranje spremljanje računalniških sistemov ima seveda mnoge pomanjkljivosti v primerjavi z zunanjim spremljanjem. Kar je uporabljen tudi procesor za izvajanje delovnih in spremljevalnih programov, zasedajo spremljevalni programi del spomina v računalniku. V našem konkretnem primeru je to kar 7k slogov ROMa in do 2k slogov RAMa. Seveda tudi kraja računalniškega časa delovnim programom ni vedno sanemarljiva. V našem primeru lahko spremljevalni programi uporabijo v kritičnih trenutkih do 3% računalniškega časa v 33ms intervalu. Dejansko je povprečna poraba računalniškega časa mnogo manjša (pod 0,5%). Velika prednost notranjega spremljanja računalnika pa je niska cena, ki v manj zahtevnih aplikacijah odtehta vse njegove pomanjkljivosti.

7. LITERATURA

(1) A.Avižienis: "Fault Tolerance by Means of External Monitoring of Computer Systems", AFIPS Conference Proceedings, Vol.50, 1981

(2) A.Avižienis: "Fault Tolerance: The Survival Attribute of Digital Systems", Proceedings of IEEE, Vol.66, No.10, 1978

(3) M.Kovačević: Sistemska obnova u uslovima realnog vremena, Informatica 6, 1981

(4) R.Murn: Postopki za povečanje zanesljivosti digitalnih sistemov, Informatica 3, 1980

SOME PROBLEMS OF IMAGE PROCESSING BY PARALLEL PROCESSOR CLIP

IVAN BRŮHA

UDK: 581.3.0.5

FACULTY OF ELECTRICAL ENGINEERING ČVUT, PRAHA CZECHOSLOVAKIA

The paper deals with problems of one implementation of the primitives extraction from an image by means of parallel processing techniques. The described algorithm, which is able to extract the primitives of polygonal objects, has been implemented on an emulator of the parallel processor CLIP.

The entire implementation of the ideas of the picture decomposition is, however, accompanied by some rather difficult problems, mainly by missing some essential visual information. The author discusses some suggestions for solving these disadvantages.

1. Image Processing

A robot needs some information about the world, it is moving in, for some actions in its environment. A visual information appears to be one of good possibilities. Fig. 1 shows a chart of the visual information processing. A 3-dimensional scene is taken by a camera and preprocessed. From arisen grey-level image, represented by a matrix of e.g. 64x64 elements, elementary primitives (e.g. vertices and edges) must be extracted. Then, a scene analysis follows, i.e. relations among objects of the scene are assigned, and the image processing is completed by an object recognition. This information is utilized by the robot for an action, e.g. for removing an object from its position to another.

The high-level image processing (scene analysis and object recognition) is realized by many methods in considerable details but the low-level image processing (primitives extraction) is the subject of research at present. The fundamental problem of the low-level image processing consists in that a large number of information contained in the grey-level image with, for instance, 16 grey levels stored in a large matrix must be reduced into small amount of primitives.

This paper deals with one algorithm for the extraction of vertices and edges of a polygonal world. The entire algorithm has been implemented as a program for parallel processor CLIP [1], [2]. The processor CLIP (Cell Logic Image Processor) operates with binary matrices (so-called bit-planes) and, in contradistinction to a sequential processor, the entire bit-plane is treated together, in parallel as a compact unit. The parallel processing of visual information enables that the time consumed for decomposition of a digitalized image be quite small so that the robot can process the visual information on-line.

The implemented algorithm of decomposition is divided into two parts:

- the first part extracts an outline image from a grey-level picture, represented by a matrix of size e.g. 64x64 elements, with e.g. 16 grey levels;
- the second part of the algorithm extracts vertices (with their coordinates) and edges (with corresponding pairs of vertices) [3], [6].

In the following, we go through the algorithm of decomposition. The algorithm is able to treat only polygonal objects. These objects symbolize a simple world which a robot can operate in.

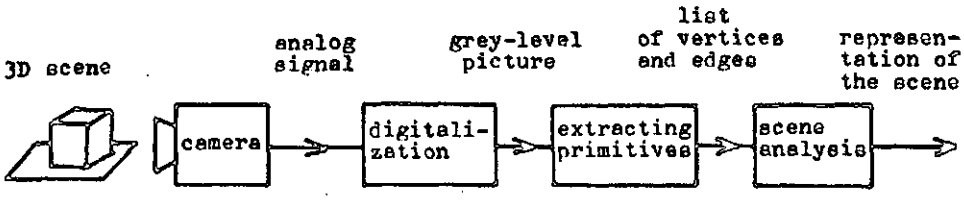


Fig. 1. Image processing.

2. Extracting an outline image

The outline image extraction from a grey-level picture is based on the fact that the outline corresponds to the spots with the largest magnitude of a gradient counted over the grey-level picture. This idea is relatively straightforward but represents the most difficult issue of the entire image processing.

The implemented algorithm has five steps:

1. Histogram assignment. The digitalized grey-level picture contains points with various magnitude of greyness so that the elements of the corresponding matrix are various numbers from 0 to say 15 . We can easily count the number of the elements of the matrix having a certain magnitude. If we consider them as a function of the magnitude of greyness we receive the so-called histogram of the given grey-level picture.

2. Removing a noise from the grey-level picture is performed by sharpening local maxima of the histogram. The grey levels which occur seldom are suppressed and, on the contrary, those which occur frequently are reinforced.

3. Computing a gradient as a relative difference of greyness of neighbouring cells is the fastest step because the parallel processor computes the gradient of all the cells together, all at once.

4. The outline image then corresponds to those spots of the grey-level picture whose magnitudes of the gradient are higher than a given threshold.

5. Shrinking the outline image. We obtain the image whose edges are represented by narrow lines.

The described algorithm is simple and comparatively fast but, as for quality of results, is not so perfect, which and other aspects are discussed in conclusions.

3. Extracting vertices and edges

In the first part of the algorithm for low-level image processing, we get an outline image which is stored in one bit-plane of the processor CLIP. We are going to search vertices and edges of the outline image. This is done by four steps [3], [6]:

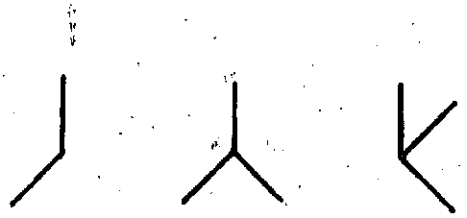
1. Extracting vertices. All vertices other than L-vertices (see Fig. 2) are extracted from the outline image. When the angle of an L-vertex is acute or right it can be found even in this step.

2. Extracting simple objects. The found vertices are removed from the outline image. Thus we obtain a set of the so-called simple objects which are either edges or a pair of edges with an L-vertex. These objects are separated and stored, each in different bit-plane.

3. Treating simple objects. The program finds out whether or not the simple objects are edges. If a simple object is recognized as an edge it is added to the list of edges. If not, the vertex is extracted from that simple object as well as the two edges. The described step can treat not only the simple objects with L-vertices but also those with Y- or U-vertices (see Fig. 2). But many experiments have revealed that the Y- and U-vertices are always extracted in the first step. This capability then means a sort of security if a vertex has been forgotten in the first step.

4. Coordinates of each vertex and the pairs of vertices to each edge are determined. The result of the entire algorithm of the decomposition is
   - the list of the found vertices with their coordinates,
   - the list of the edges with corresponding pairs of vertices.

The steps 2. to 4. do not contain any complicated procedures. The extraction of vertices is, however, rather complicated so that



L-vertex Y-vertex W-vertex

Fig. 2. Some types of vertices.

it would be explained in detail. The algorithm utilises the fact that vertices in an image are the points in which edges intersect, which means that they correspond to the points enclosed by the biggest number of excited cells, i.e. cells which contain 1 . So a density function $d_R(x)$ over the set of all cells can be constructed. Its value equals to the number of excited cells inside the circle with center x and radius R . On the basis of a large number of experiments, the radius $R=3$ has been chosen as the optimal one which can distinguish vertices and edges and, on the other hand, does not lose the local character of sight in a bit-plane of size up to about 200x200 . One can see that the L-vertices cannot be found generally in this step because every point close to an L-vertex has approximately the same number of neighbour excited cells.

The procedure for extracting vertices is as follows:

1. Compute the density d_j for the outline image and assign
   $\ell := 2^{*}L$
where L is number of bit-planes in which the density is stored. (Notice that this step is very fast thanks to the parallel procesor.)

2. Assign
   $\ell := \ell - 1$

3. Save cells with density equal to ℓ in a bit-plane A . If the bit-plane A is empty go to 2.

4. In the bit-plane A remove those excited cells which lie in surroundings of already found vertices.

5. If the bit-plane A does not contain an edge, consider the cells in A as new vertices, add them to already found vertices and go to 2.

6. Smooth the found vertices.

7. Create circles around the found vertices and compute the density d_1 for those cells of the outline image which lie inside these circles. Find more precise position of each vertex so that the new position of a vertex is a cell with maximal value of the density d_1 (computed separately for each vertex).

The algorithm for extracting vertices and edges from an outline image is quite simple and reliable as many experiments have verified.

4. E x a m p l e

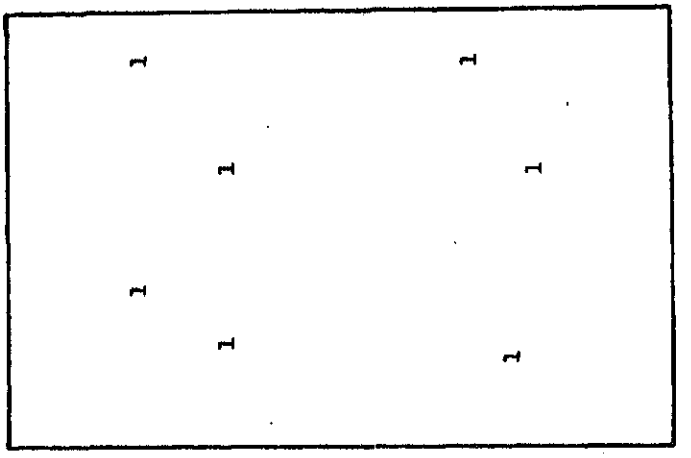
The entire algorithm for extracting vertices and edges from a grey-level picture has been implemented on a CLIP emulator written in FORTRAN. As bit-planes are treated in parallel, the time for entire treatment is not dependent on the size of bit-planes. The described method for decomposition is, to a considerable extend, independent on complexity of a given scene (i.e. number of vertices and edges). Experiments have revealed that the processor CLIP needs about 8000 instructions for decomposition of a scene with about 8 to 15 vertices and 8 to 15 edges. The hardware CLIP-4 [1] with about 10 μs per an instruction would consume about 80 ms which is enormously short time for the primitives extraction.

Fig. 3 shows a process of the decomposition of a simple scene created by single prism. Discussion of this example follows.

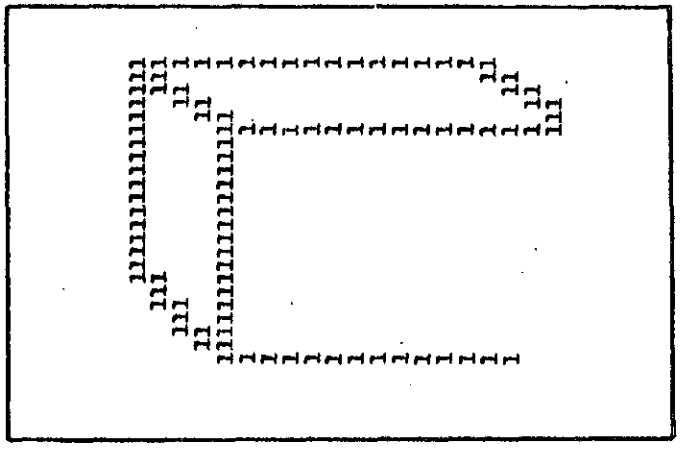
5. C o n c l u s i o n

Parallel approach represents fundamental turn in image processing. Large speed of processing, independence on size of bit-planes and only slight dependence on complexity of a scene speak clearly for the parallel approach. On the other hand, detailed analysis of the method and evaluating many experiments has revealed some negative features which either have general character or are typical just for the parallel approach. We discuss the most important ones:

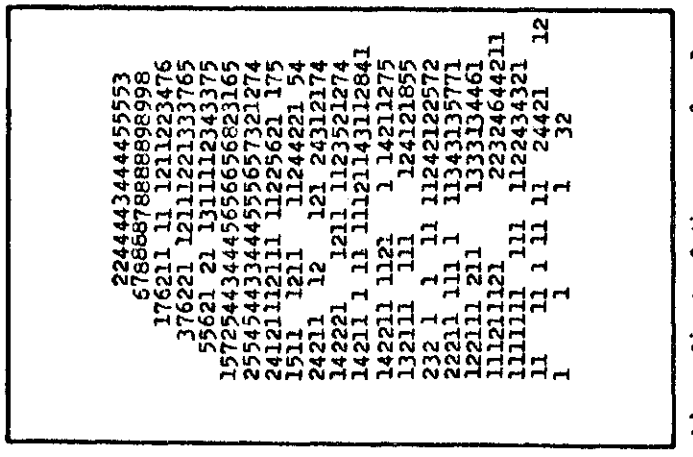
1. The greatest and essentially unremovable disadvantage is both the lack of information of one or more edges and too large noise in a grey-level picture (see Fig. 3a). It is brought about partly by lighting, insensibility of a camera or by the digitalization. But human eye is able without problems to perceive a given scene with a large noise. Human being, however, uses visual receptors together with his brain and, for creating a represen-



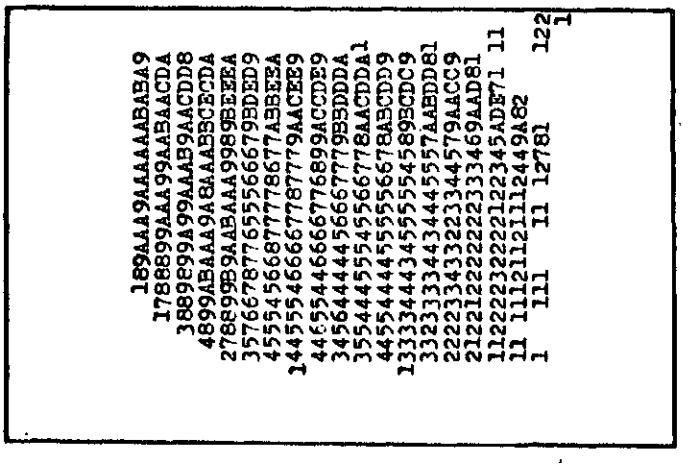
d) extracted vertices



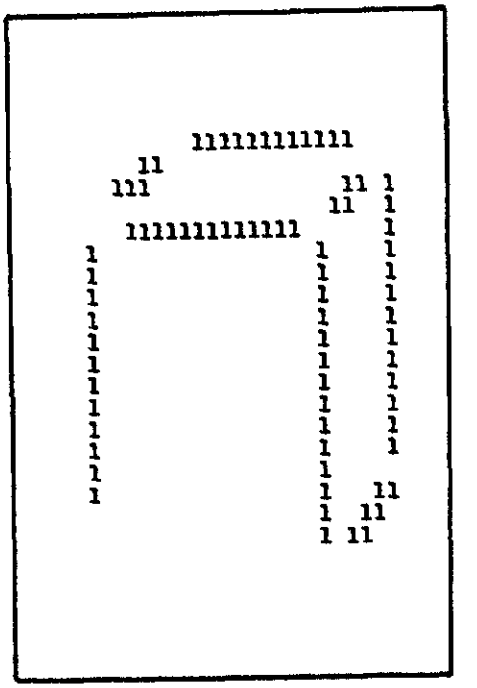
c) outline image



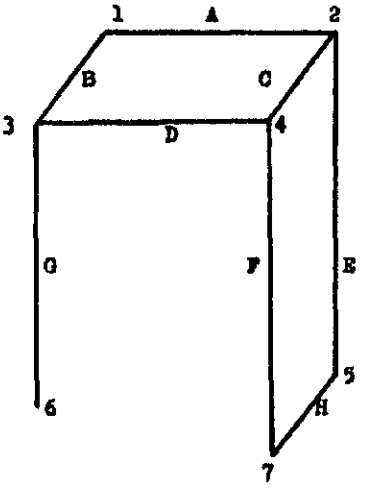
b) gradient of the grey-level picture



a) grey-level picture



e) extracted edges



f) representation of the scene

Fig. 3. An example of extracting vertices and edges from a grey-level picture. Cells with value 0 are depicted as spaces. Numbers in Fig. a,b) are hexadecimal, i.e. A, B, ..., F mean 10, 11, ..., 15 . Scale of all figures is 1:2 .

tation of a perceived scene, he also realizes a feed-back loop.

Not even the most perfect algorithm for decomposition of a grey-level picture is able to extract those edges which it has no information of. An error in extracted data must be discovered by the scene analysis (Fig.1). Waltz's algorithm for scene analysis [4] itself fails for unsufficient data without locating an error. An algorithm, extending Waltz's method, which is able not only to locate an error but also to complete an outline image by missing edges [5] has been developed. It is desirable to verify the added edges by a feed-back loop so that the algorithm of decomposition is recalled

- either for the same grey-level picture (with concentrating on an area of added edges)
- or for a new grey-level picture which arises from digitalizing the same scene, but lighted from the other side than the first snap.

2. The implemented algorithm for extracting an outline image has its weak place: removing a noise. The algorithm assumes that a face of a scene has approximately the same magnitude of greyness. Experiments, however, confirmed that reality does not agree with that assumption.

3. The assignment of an outline image from a gradient is not also convenient in many cases. The outline image is formed by those cells which have their gradient bigger than a threshold (which is constant for all scenes). Detailed analysis, however, demonstrates that this threshold should be changed even for different lines of the same image. It is not solved so far how to do it.

4. Architecture of parallel processor CLIP itself is not without comments. Parallel processor is suitable in the first steps of algorithm, when an input image is processed as one unit. The more we reach the result of processing, the more sequential character the processing has. During these final steps, the parallel processor treats only some small part of bit-planes and it sequentially performs approximately the same procedure for all the vertices, all the simple objects etc. It would be the best to construct a sequential-parallel processor which could treat bit-planes or similar formations both in parallel and in sequential manner.

References

[1] Brůha,I.: One method of searching vertices and edges by CLIP parallel processing. MIP-R-121, Machine Intel. Research Unit, Edinburgh University, 1977.

[2] Duff,M.J.B., Watson,D.M.: A parallel computer for array processing. Inform. Processing, Holland, 1974.

[3] Brůha,I.: Decomposition of outline images by means of parallel processor CLIP. (In Czech). Automatizace SNTL Praha,11, 1979.

[4] Waltz,D.L.: Generating semantic descriptions from drawings of scenes with shadows. MIT, AI-TR-271, 1972.

[5] Brůha,I., Woska,J.: Locating errors in an outline image representing a scene with polygons. (In Czech). Automatizace SNTL Praha, 1982.

[6] Brůha,I.: Some experiences with the CLIP parallel processing used for the picture decomposition. In: Informatica 78, Bled, XIII. Yugoslav Internl. Symp. on Inform. Processing, 1978.

ETHERNET – LOKALNA MREŽA PRIHODNOSTI

M. KOVAČEVIĆ,
D. PEČEK,
B. KASTELIC,
R. MURN

UDK: 681.324

ISKRA DELTA, LJUBLJANA
INSTITUT JOŽEF STEFAN, LJUBLJANA

Povezati ali ne? To danes ni več vprašanje, pač pa potreba. Vprašanje je le kako? Pričujoči sestavek bo prikazal lastnosti lokalne mreže imenovane ETHERNET, katere razvoj kaže na to, da bo postala standard.

ETHERNET - To connect or not to connect? It's no more the question nowaydays it's a necessity. But the question stays. How? The paper will show the characteristics, features and facilities of the local net - ETHERNET -, that has all the chanses to become standard.

UVOD

Deset let je minilo odkar so se prvič pojavili mikroprocesorji. Oprema in naprave s mikroračunalniško podporo so postale sestavni del življenja držav razvitega zahoda in Japonske. Navedimo v ilustracijo da je samo ameriški trg za terminale v letu 1979 prodal 3.3 M kosov in da v letu 1984 pričakujejo prodajo 7.6 M kosov. Takšna sredina sama po sebi narekuje gradnjo močnih komunikacijskih povezav. Tako globalnih kot lokalnih. V članku bomo prikazali rešitev problema lokalnih mrež, ki povezujejo med seboj elemente v nekem zaključenem okolju kot so naprimer poslovna stavba, tovarna, institucija, etc.

Do današnjega dne je bilo vloženih že precej naporov za standardizacijo lokalnih mrež. Ena prvih pobud je prišla s strani mednarodne organizacije za standarde(ISO). Odbor IEEE je izdelal vrsto priporočil za standardizacijo arhitekture, topologije, prenosne tehnologije in dostopa, ki so znana pod imenom PROJECT 802. Toda osnovni pogoj da neka ideja - predlog postane standard, so potrjeni testi v realnem svetu, izvedba mora biti enostavna in uživati mora podporo večine proizvajalcev.

Prvi korak v resničnem razvoju in implementaciji lokalnih mrež so storile tri tvrdke: DEC, XEROX in INTEL. XEROX je izdelal osnovni prototip mreže in ji dal ime ETHERNET, DEC je razvil celoten sistem povezovanja s posebnim povdarkom na prenosnih vezjih, INTEL pa je izdelal in realiziral projekt implementacije kompleksnih krmilnih in povezovalnih funkcij s mikroračunalniško tehnologijo in VLSI komponentami.

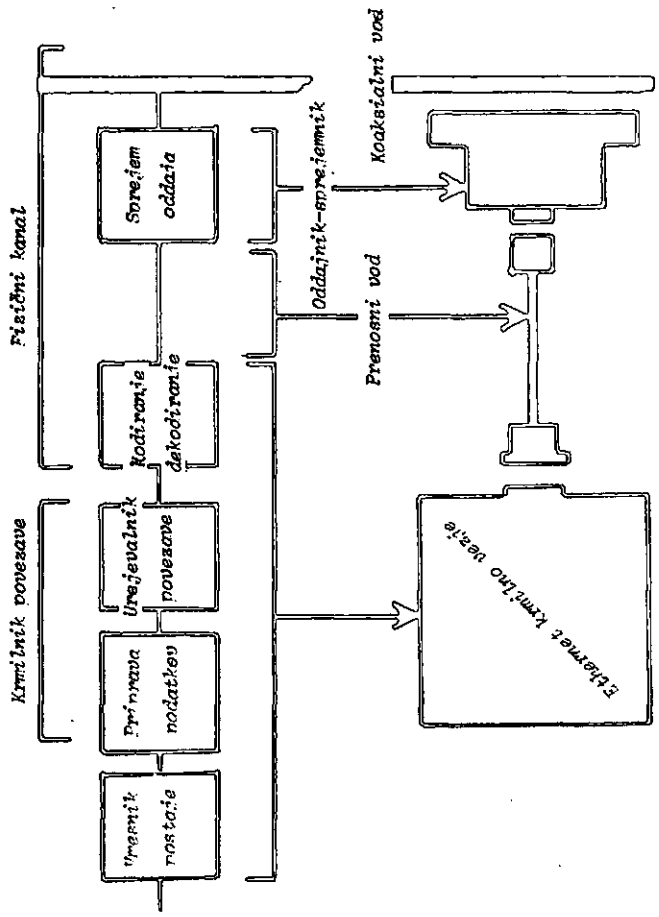
Združeno delo teh treh proizvajalcev predstavlja električne, logične in protokolske specifikacije mreže ETHERNET na dveh najnižjih nivojih. To sta linijski in fizični nivo.

V uvodu navedimo nekatere najbolj osnovne lastnosti mreže ETHERNET.

- *enostavna izvedba*
- *zanesljivost*
- *fleksibilnost naslavljanja*
- *velika prenosna hitrost*
- *majhna zakasnitev*
- *enostavno vzdrževanje*
- *izkoristek mreže je 90 odstoten*

1. ETHERNET - arhitektura

Arhitekturni model mreže je zasnovan na vrsti vmesnikov, ki se razlikujejo od tistih, ki jih srečujemo pri dejanski realizaciji s slike 1. Vmesniki dejanske realizacije morajo ustrezati zahtevam kompatibilnosti. Za to je odgovoren predvsem zadnji, najnižji nivo - sloj, fizični nivo, ki vsebuje dva pomembna vmesnika:



Slika 1.

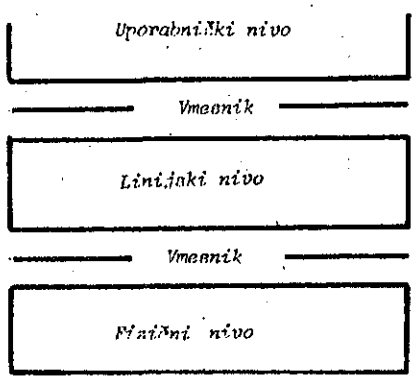
a) Vmesnik koaksialnega kabla
Komunikacije v mreži zahtevajo strogo upoštevanje kompatibilnosti vmesnika koaksialnega kabla s signali in fizikalnimi lastnostmi kabla.

b) Oddajno sprejemni vmesnik
Večina postaj v mreži bo locirana v večji ali manjši oddaljenosti od priključka na koaksialni kabel. Minimalna materialna oprema, oddajno sprejemni vmesnik, je locirana ob samem kablu, ostali del materialne opreme(krmilna vezja), pa v sami postaji.

Več plastna arhitektura

Glavni poudarek v arhitekturi ETHERNET je na dveh osnovnih plasteh - nivojih

Linijski nivo (DATA LINK LAYER) ni fizični nivo (PHISICAL LAYER). Vsi višji nivoji, ki komunicirajo z linijskim nivojem, so označeni kot plast - nivo odjemalcev (CLIENT LAYER). Slika 2. prikazuje večplastno strukturo mreže.
Prikazane plasti komunicirajo preko definiranih vmesnikov. Vmesnik med plastjo odjemalcev in linijsko plastjo omogoča pošiljanje in sprejemanje paketov ob zagotavljanju statusnih informacij, ki jih uporabljajo procedure za obnovo v višjih plasteh.
Vmesnik med linijsko plastjo in fizično plastjo pa ovrednoti signale, ki so potrebni za inicializacijo oddaje- sprejema paketa, detekcijo kolizij na vodilu, realizira pa tudi čakalno funkcijo za zadostitav časovnih zahtev.



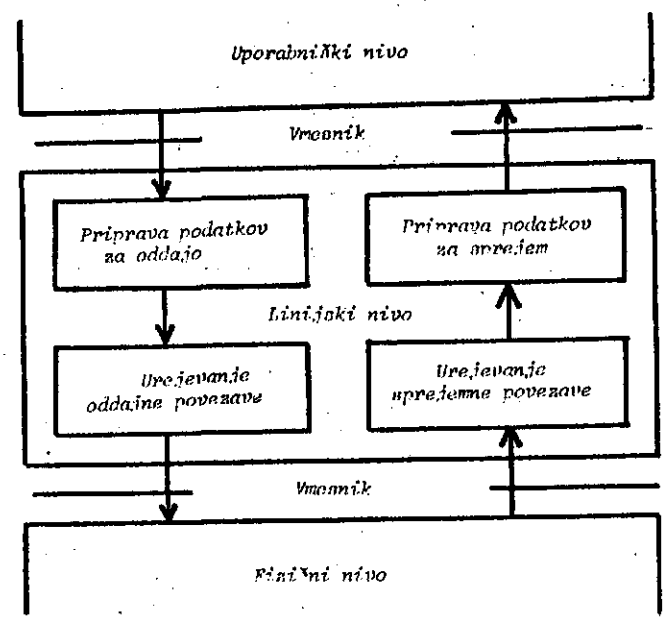
Slika 2.

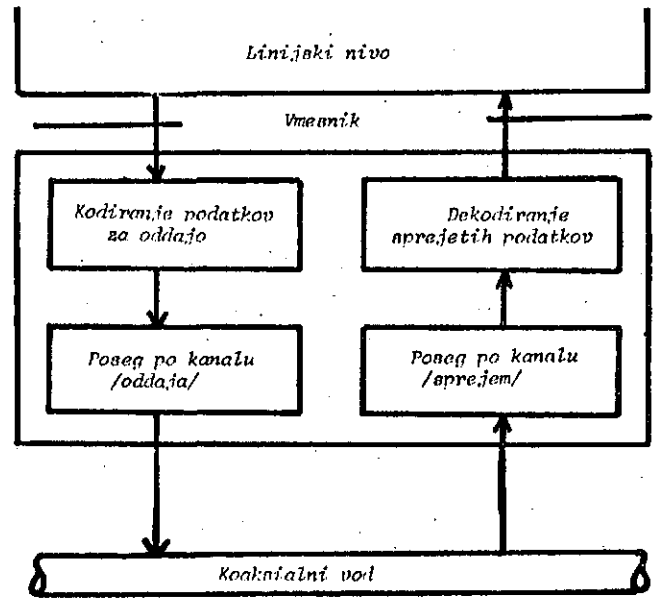
1.1 Linijska plast

Linijska plast je definirana tako, da kar najbolj ustreza specifikacijam, ki so podane v ISO modelu. Pripadata ji dve osnovni funkciji:
- formiranje podatkovnih paketov
- kontrola posega po prenosnem mediju

Slika 3. podrobneje razlaga obe osnovni funkciji.
Linijska plast zagotavlja povezovanje dveh ali več plasti višjega nivoja, ki želo komunicirati med seboj. Na ta način ustvarjena zveza se imenuje podatkovna zveza (data link). Materialna programska oprema, ki realizira to povezavo se imenuje krmilnik linijske povezave (data link controller), povezava pa sama fizično steče preko fizične plasti, ki jo imenujemo fizični kanal (physical channel).

1.3 Fizična plast





Slika 3.

Fizično plast predstavlja tako imenovani fizični kanal s kapaciteto 10 M bitov/sec in je priključen na koaksialni kabel. Na nivoju te plasti so specificirane vse pomembne fizikalno
- logične karakteristike mreže, kot so naprimer: kodiranje podatkov, časovni poteki, napetostni nivoji, etc. Podobno kot linijska plast ima tudi fizična plast dve osnovni funkciji, ki se neposredno vežeta na funkciji linijske plasti :
- prelitje formiranih podatkovnih paketov v impulze in dodajanje uvodnih bitov
- detekcija stanja kanala in posredovanje parametrov linijskemu nivoju.

1.3 Funkcionalni model mreže

Opis pričnimo s predpostavko da je iz uporabniškega nivoja prišla zahteva za oddajo paketa. Zahteva je posredovana linijski plasti. Na osnovi dobljenih podatkov se formira paket, ki je dopolnjen z uvodnimi biti, ki služijo za detekcijo napak. Paket potem preide v drug del plasti, ki ima nalogo povezovanja in ki kontrolira dostop do kanala. To pomeni: inogniti se oddajanju paketa, če je kanal zaseden. Ta funkcija je realizirana s otipavanjem linije CARRIER SENSE. Kakor hitro je kanal prost, steče oddajanje paketa. Linijska plast pošilja zaporedje bitov fizični plasti, ta pa pošlje formirane signale v koaksialni kabel. Ves čas oddaje teče preverjanje signalne linije (collision detect).

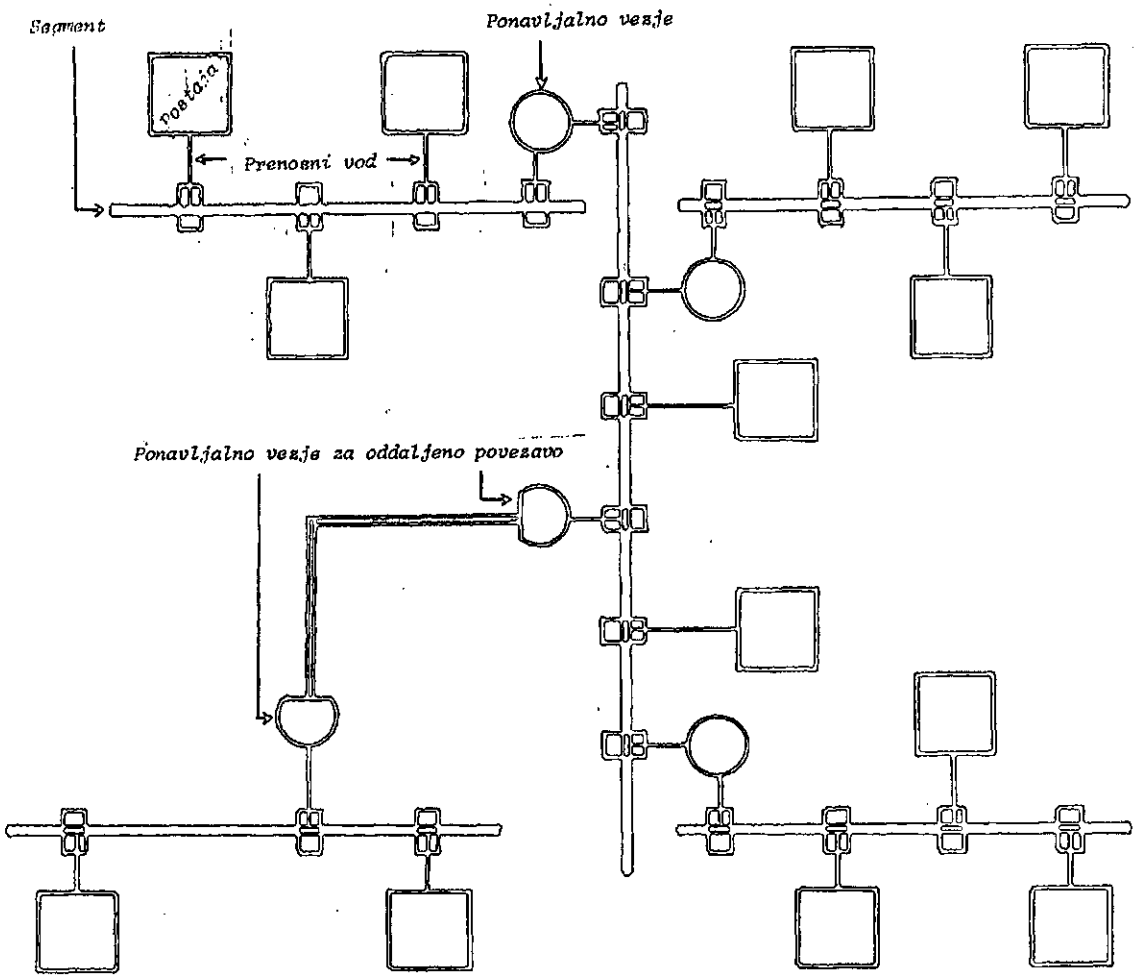
Na sprejemni strani se sprejemnik s pomočjo uvodnih bitov paketa sinhronizira z oddajnikom, sprejeti paket se dekodira in pretvori v linearno obliko in posreduje linijski plasti. Ta proces kontinuirano teče, dokler je linija CARRIER SENSE aktivirana. Deaktiviranje linije povzroči pregled ciljnega naslova paketa in če ta ustreza, se ta posreduje plasti odjemalcev, vključno s statusnimi informacijami, ki so generirane na osnovi kontrolnih bitov paketa.

2. KOLIZIJSKI POSTOPKI

Kolizija v mreži nastopi, če več postaj poskuša istočasno oddajati pakete. Vsaka postaja ima možnost detekcije kolizije v samem začetku oddajanja paketa (kolizijsko okno). (Postaja začne z oddajanjem. Zaradi končne hitrosti širjenja signalov po mreži, mora začetek oddajnega paketa doseči najbolj oddaljeno postajo, da je le - ta informirana o zasedenosti kanala. Čas, ki poteče od začetka oddajanja paketa do trenutka, ko najbolj oddaljena postaja ta paket prične sprejemati, se imenuje minimalno kolizijsko okno.) Če začetek oddanega paketa uspešno preide kolizijsko okno, postaja predpostavlja, da je zasedla kanal. Nadaljnje kolizije so preprečene, ker vsaka postaja čuti, da je kanal zaseden.

V primeru kolizije komponenta fizikalne plasti detektira interferenco na vodilu in aktivira kolizijski signal, obenem pa vsili vodilu določeno zaporedje bitov. Le - ti zagotovijo, da detektirajo kolizijo vse postaje. Oddajanje se ustavi, izbere se naključni čas čakanja za ponovni poiskus oddaje paketa.

Biti ki so rezultat kolizije se sprejmejo in detektirajo enako kot veljavni biti. Kolizija ne aktivira kolizijskega detekcijskega signala na sprejemni strani. Paketi sprejeti v pogojih kolizije se ločijo od veljavnih paketov v linijski plasti.



Slika 4.

6. Arhitektura mreže in njene omejitve s fizičnega nivoja

POSTAJA - Terminalno sprejemno/oddajno mesto, ki ga(jih) ETHERNET poveže v celoto. Maksimalno število postaj je omejeno na 1024.

SEGMENT - je pravilno zaključen koaksialni kabel, maksimalne dolžine 500 m.

PONAVLJALNO VEZJE - je vmesnik, s pomočjo katerega razširimo obseg topologije mreže, vendar sta lahko med dvema poljubnima postajama največ dva vmesnika.

PRENOSNI KABEL - je priključek, ki vodi od postaje do koaksialnega kabla in je lahko dolg največ 50 m.

PONAVLJALNO VEZJE ZA ODDALJENO POVEZAVO - je vezje, ki omogoča povezati dva relativno precej oddaljena dela mreže, vendar največ do razdalje 1000 m.

Zaključek

Ob zaključku prikaza lokalne mreže ETHERNET navedimo lastnosti, ki karakterizirajo mrežo, tako pozitivne, kot negativne.

a) Pozitivne

- enostavnost - Vsi elementi, ki bi lahko kakorkoli kompliciral lastnosti mreže in njihovo implementacijo so izključeni.

- nizka cena - Nenehno zniževanje cen postaj(terminalov, etc.) mora vplivati tudi na zniževanje cene za same povezave.

- kompatibilnost - Katerakoli implementacija mreže omogoča izmenjavo podatkov na nivoju linijske plasti.

- fleksibilnost naslavljanja - Mehanizem naslavljanja (v članku ni prikazan zaradi licenčnih pogojev omogoča naslavljanje posameznih objektov, skupine objektov, ali pa vseh objektov, ki so povezani v mrežo).

- enakopravnost - Vsi objekti mreže imajo v povprečju enakopraven dostop do mreže.

- napredek - Katerikoli objekt v mreži, ki deluje v skladu s protokoli(in to mora), na more preprečiti napredka drugih objektov.

- velika prenosna hitrost - Po mreži se prenašajo podatki s hitrostjo 10 M bitov sec.

- kratka zakasnitvena doba

- stabilnost - Mreža je stabilna tudi v primeru maksimalne obremenitve

- vzdrževanje - Konstrukcija mreže omogoča množico postopkov za vzdrževanje in planiranje mreže.

- večplastna arhitektura - Katerikoli načrt mreže mora biti specificiran v smislu večplastne arhitekture, tako, da so logični pogledi in razmišljanja ločeni na dva nivoja: linijski nivo in fizični nivo.

b)Negativne.

- Operacija „full duplex" - V katerem koli trenutku lahko prenašamo paket po mreži samo v eni smeri. Dvosmerni prenos je realiziran v smislu hitrih izmenjav paketov, ne pa kot „full duplex".

- kontrola napak - Indikacija napak je omejena na naslednje tipe:
  - odkrivanje napak posameznih bitov paketa v kanalu.
  - odkrivanje napak zaradi kolizij.

Odkrivanje in ukrepanje v primeru katerihkoli drugih napak je prepuščeno višjim nivojem arhitekture(plasti odjemalcev).

- varnost - Protokol podatkovne povezave ne predvideva možnosti za kodiranje podatkov v smislu varnosti. Če želimo varen prenos v tem smislu, je to potrebno storiti v plasti odjemalcev.

- fleksibilnost prenosne hitrosti - V mreži je možna samo ena prenosna hitrost - 10 M bit/sec.

- prioriteta - Postajam v mreži ni možno predpisati različnih prioritet.

- kvarni uporabnik - Mreža sama ni zavarovana pred posegi t.i. kvarnih uporabnikov na nivoju podatkovne plasti.

SPREMNA BESEDA

Namen članka je bil seznaniti bralca s malce bolj podrobnimi prijemi pri realizaciji danes že popularne mreže ETHERNET, ki vzbuja veliko zanimanja pri vseh potencialnih odjemalcih. Vse kaže, da bo postala svetovno priznan standard za lokalne mreže.
In kakšen je fizični izgled bistva mreže, t.j., kakšna je fizična forma vezij, ki realizirajo funkcije s slike 1? To sta dve dvostransko tiskani vezji, velikosti dvojnega evropskega formata. Že v letošnjem letu pa tvrdka INTEL najavlja novo VLSI integrirano vezje ETHERNET CONTROLLER, z razvojno oznako 82CXDX, ki bo združevala tako rekoč vse funkcije s slike 1.

Literatura:

CRAN.C.,TAFT E.A.,,PRACTICAL CONSIDERATIONS IN ETHERNET LOCAL INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, JANUAR 1980, HAWAII."

2)SHOCH J.F.,HUPP J.A.,,MEASURED PERFORMANCE OF AN ETHERNET LOCAL NETWORK". LOCAL AREA COMUNICSTIONS NETWORK SYMPOSIUM. MAJ 1979, BOSTON.

3) METCALFE R.M., BOGGS D.R.,,ETHERNET: DISTRIBUITED PACKED SWITCHING FOR LOCAL COMPUTER NETWORKS".COMUNATIONS OF THE ACM 19 7, JULIJ 1976.

4) PUBLIKACIJA TVRDKE INTEL: „THE ETHERNET" VERSION 1.0 SEPTEMBER, 1980.

ON AN IMPLEMENTATION OF THE POP – 2 LANGUAGE

IVAN BRŮHA

UDK: 681.3.0.6 POP-2:519.682

FACULTY OF ELECTRICAL ENGINEERING ČVUT
PRAHA, CZECHOSLOVAKIA

Problems of one implementation of the POP-2 programming language for the PDP-11 computers are presented, especially the representation of data structures and the way of compiling. Representation of POP-2 data structures has been solved together with proposal of the garbage collector in order to utilize the heap as much as possible. It is shown how POP-2 functions and statements are compiled. The PDP-11 instructions such as MOV , JSR , JMP , CMP , BNE , BEQ , TRAP etc. are direct used in the function body. The paper concludes by comparing three known implementations of POP-2/PDP-11, namely one from Prague, the others from Great Britain.

1. Introduction

The purpose of this paper is to reveal the problems relating to a concrete implementation of the POP-2 programming language for the computers PDP-11.

The reader is supposed to be in general familiar with the notion of data structures. Therefore, we will not concern ourselves in theoretical aspects but will get acquainted with the problems of a concrete implementation of data structures. The reader need not know nearly anything about the POP-2 language. Notice only that POP-2 was developed as a powerful language for non-numerical applications and Artificial Intelligence at Edinburgh University in 1969-70.

Because of better comprehension, notice that the POP-2 language is stack-oriented. It uses two stacks:

- the system stack, where the system itself stores information such as return addresses, current values of locals of functions etc.,
- the user stack, where a user pushes values of variables; this stack is also utilized for passing values to formals of functions and for inserting results of functions.

The POP-2 language for PDP-11 is implemented as a compiler. In additional, POP-2 is a conversational language allowing the user to communicate with his program and vice versa. For that reason, the compiler as well as compiled functions must be stored in the core when the system is running. The core is divided into

- the system and user stacks,
- the POP-2 system itself (lexical and syntax analysers, standard functions, garbage collector etc.),
- the heap (user's memory) for storing information about data structures and compiled functions.

2. Data structures of the POP-2 language and their representation for PDP-11

2.1. Simple and compound items of the language

The objects on which one can operate and which one can obtain as results are called items. They are divided into two distinct classes:

1) simple items or numbers, which consist of integers and reals,
2) compound items or data structures, which consist of records, strips and functions.

The POP-2 system must retain information not only about components of data structures, but also about their type, length etc. Because of simple orientation in the memory, this in-

formation is stored in a continuous area of memory, called memory cell. Address of the memory cell is the address of its first byte.

All references to a data structure are realized as a pointer to the memory cell which represents the given data structure. We must strictly distinguish the representation of data structures by means of memory cells and that by pointers to memory cells. A data structure as an item stored in the stack or a component of another data structure is represented by a pointer only.

In case of PDP-11, the information of data structures is stored in words so that the address of a memory cell is always the address of a word. The right-most bit of any pointer to a data structure is, thus, equal to 0 . The numbers as the POP-2 items should consequently have their right-most bits equal to 1 in order to be distinguished from the compound items.

On the other hand, we would futher have to distinguish integers and reals, for which another bit should be necessary. Moreover, reals must be stored in at least 4 bytes.

The above considerations have led to the following representations of the POP-2 items (Fig. 1):

1) An item is the data structure if its right-most bit is equal to 0 , i.e. it is a pointer.

2) An item is an integer if its right-most bit is equal to 1 . The value of the integer itself is placed in the remaining 15 bits (including the sign bit). The extent of POP-2 integers is thus half in comparison with PDP-11 integers, but this is not any disadvantage because the main purpose of POP-2 is not treating integers.

3) Reals are considered as pseudo-data structures: the value of a real is placed in two words of a memory cell, and the real as the POP-2 item is represented by the pointer to the corresponding memory cell. Data structures and reals are distinguished by means of a header of memory cell (see below).

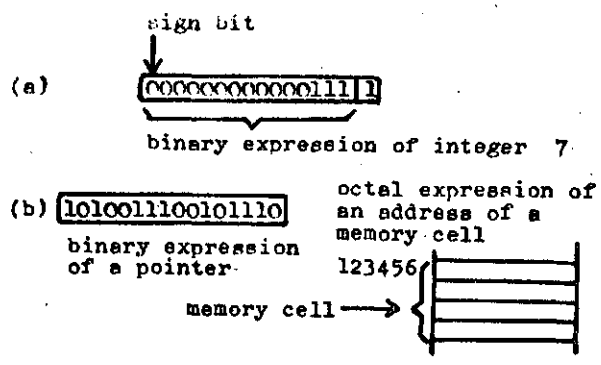


Fig. 1. Representation of (a) integers, (b) data structures.

Besides the components of a data structure, the memory cell, which represents the given data structure, must contain this futher information:

a) number of components,
b) size of components, i.e. the number of bits in which the given component is stored,
c) data name of the data structure class which the given data structure belongs to,
d) type of the data structure, i.e. whether it is a record, strip etc.,
e) size of the memory cell, i.e. number of words the cell occupies.

Some of the above information is contained in the memory cell implicitly, which depends on the type of the structure. The type of data structure and the size of the memory cell must, however, be involved always and easily accessible, because this information is necessary for the garbage collector. This demand has led to introducing so-called header of a memory cell. It is the first word of the memory cell and contains:

1) key of the memory cell,
2) size of the memory cell (1 to 2047 words),
3) GC-bit, utilized by the garbage collector (normally 0),

see Fig. 2. The type of a structure can be assigned either by the key or by the key and size of the memory cell.

2.2. Pairs and lists

Pairs create a standard record class. They are derived from the LISP language. The pair consists of two components, called front and back , each stored in one word [1],[2]. The pair can't be represented by a memory cell of size 2, as in case of LISP, because each memory cell must have its header. The memory cell

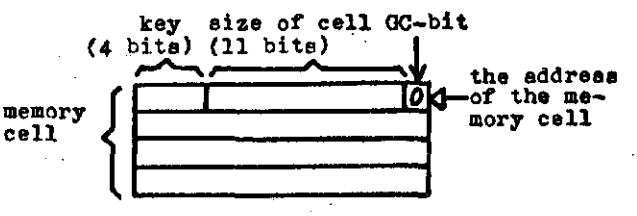


Fig. 2. Header of a memory cell.

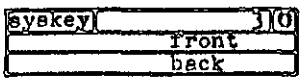


Fig. 3. Representation of a pair (syskey=10_8 is the system key).

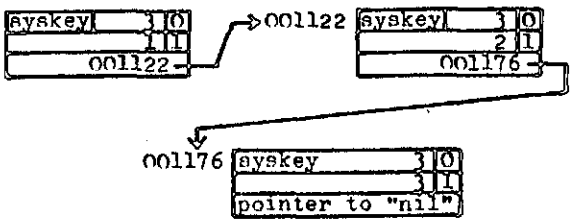


Fig. 4. Representation of the list [1 2 3] .

of a pair has size 3, see Fig. 3.

Similarly to LISP, a list is represented by a chain of pairs, with the item "nil" at its end; front of each pair is equal to an element of the list, back is the link to next pair of the chain. An example of list representation is at Fig. 4.

2.3. Words

Word of POP-2 language*) is a succession of letters and digits, beginning with a letter, or signs as + - ? , and its length is at most 8 . Words of POP-2 mean not only data structures but also identifiers. Therefore, their representation is rather complicated than in case of pairs. The memory cell of a POP-2 word (Fig. 5) contains these data:

1) value of a variable associated with the given identifier,
2) link of a dictionary chain,
3) value meaning,
4) type of identifier,
5) number of characters of the word,
6) characters of the word.

Now some comments about that:

1. The POP-2 system must have an access to any identifier at any time. That enables so-called dictionary, which links all declared identifiers and words into 8 chains according to their lengths. Therefore, the memory cell of a word must contain the link of a dictionary chain, i.e. pointer to sequent word in the given chain.

2. Each word can posses another value, called meaning , which is similar to P-lists of LISP's atoms.

3. Each identifier obtains some syntactic property when it is declared. This property with additional information is stored as the type of identifier. According to this type, the system recognizes syntax words, standard identifiers, user's identifiers, macros, operations, canceled identifiers etc.

2.4. Records

Pairs and words are standard record classes. The user has possibility to introduce his own record classes. A record class is the set of records which have
1) the same data name,
2) the same number of components and the components in the given order have the same sizes.

E.g. the statement
recordclass per (name:6,age:7,sex:1);

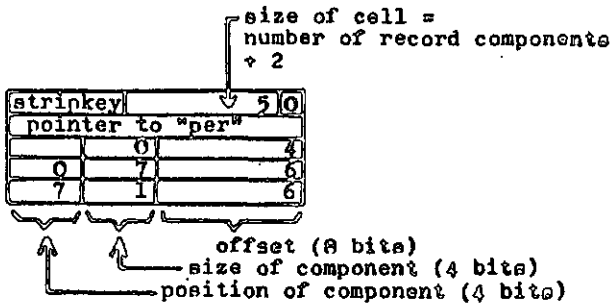


Fig. 6. The descriptors cell (stripkey=16_8).

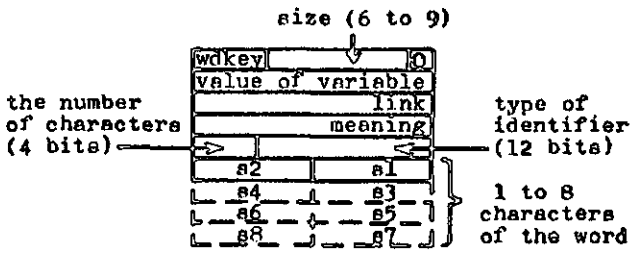


Fig. 5. Representation of a word (wdkey=13_8).

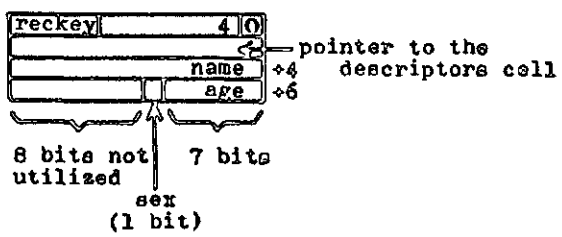


Fig. 7. Memory cell of a record of the class per (reckey=17_8).

*) Word of POP-2 and word of memory are two different terms but one can distinguish them from context.

introduces the record class with the data name "per" , whose records have 3 components (the size 0 indicates that the given component is stored in the entire memory word):

1. component has size 0 , called name ,
2. 7 age ,
3. 1 sex .

When a new record class is introduced, so-called descriptors cell is formed; it contains the data name of the record class and descriptors of all components. Each descriptor is composed
- by word offset of the memory cell where the given component is stored,
- by size of the component,
- in case of size > 0 also by the component position, i.e. number of bits on the right side of the component.

The descriptor cell of the above record class is at Fig. 6.

The memory cell of a record contains components as well as the pointer to the descriptors cell. A record of the class per is at Fig. 7.

2.5. Strips

If we want to handle items of the same size and not to distinguish their single components by different names, strips can be used. A strip class is the set of strips which have
1) the same data name,
2) the same component size,
but their lengths (number of components) can be different.

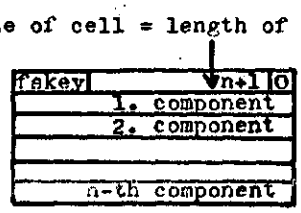


Fig. 8. Representation of a full strip of length n ($fskey=14_8$).

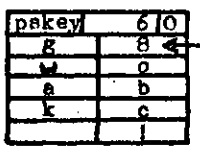
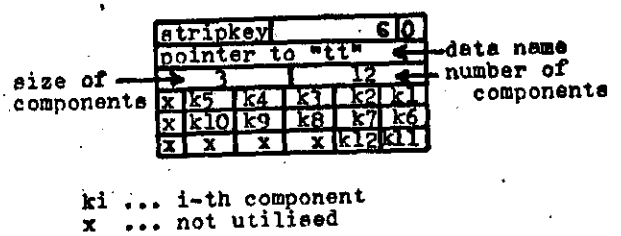


Fig. 9. Representation of the string 'go back!' ($pskey=15_8$).



ki ... i-th component
x ... not utilised

Fig. 10. Representation of a strip of the class tt , lenght 12 (stripkey $=16_8$).

Two strip classes are standard in POP-2. The first is full strips with component size 0. Their representation is at Fig. 8. The second is stringe with component size 8. Their components are interpreted as ASCII code of characters. An example of a string is at Fig. 9.

The user can define his own strip classes with component size > 0. E.g. the statement

stripclass tt:3;

introduces the strip class with data name "tt" and component size 3. In contradistinction to records, no descriptors cell is formed. All information is retained in memory cells of strips. Fig. 10 shows the representation of a strip of class tt , length 12.

2.6. Functions

The POP-2/PDP-11 language is implemented as a compiler. POP-2 functions are translated into machine code and must be comprehensibly stored in memory cells as records and strips.

Besides the function body, the memory cell for the function contains this additional information:

1) The user can, besides the function body, attach a special value fnprops (function properties) to any function. Fnprops means the same situation as meaning of POP-2 words. Default value is the function name.

2) To each function there can correspond another function, called updater, which has the same function name; it does not select but modifies components of data structures. The updater is also stored in the memory cell. Its default value is the dummy function "nil" .

3) The memory cell of a function contains also addresses of locals and formals of the function.

That is why, the POP-2 function is both a data structure (fnprops, updater) and a control structure (compiled function body). The shape of their memory cells and the way of compiling functions are described in Chapter 3.

2.7. Other types of memory cells

Besides above types of memory cells, there exist other types of cells, explicitly for references (standard data structure), compiled instructions, label names and their values, backtracking, buffers for lexical analyser and buffers for input/output. This variety of memory cells brings about that the garbage collector of the POP-2 system must be rather sophisticated.

3. W a y o f c o m p i l i n g

The memory cell of a function is created in two steps. In the first step, the text of the function is translated into a certain metalanguage; the instructions of the metalanguage are ordered in conformity with their precedences. In the second step, the system constructs a memory cell of corresponding size according to the number of the metalanguage instructions, and fills it by the machine code obtained by translating the metalanguage instructions. Notice that the first step is machine independent, only a code generator must be written for concrete type of the computer.

Fig. 11 shows both the metalanguage instructions and their translation into the PDP-11 assembler. For better understanding, notice the use of the registers in the implemented POP-2 system:

- r3 contains the address of the memory cell of the function which is being executed,
- r4 contains PDP-11's 1, i.e. POP-2's 0 or false (see representation of POP-2 integers),
- r5 is the user stack pointer,
- sp is the system stack pointer.

We can see from the shape of compiled instructions that each instruction of the metalanguage occupies two words in the memory cell, type of the instruction is in the first word and usually an address is in the second one. This order is made possible thanks to the PDP-11 assembler. The garbage collector can orientate itself easily within the memory cells of functions.

Example. The conditional

```
if x>1 then alpha(x) else beta(x) close->z;
```

is compiled as follows:

metalanguage		assembler	
	JEQ IFLAB		CMP (R5)+,R4
			BNE .+6
			JMP IFLAB(R3)
	PUSH X		MOV @#X,-(R5)
	CALL ALPHA		TRAP RCALL
			.WORD ALPHA
	J CLOSELAB		JMP CLOSELAB(R3)
IFLAB:	PUSH X	IFLAB:	MOV @#X,-(R5)
	CALL BETA		TRAP RCALL
			.WORD BETA
CLOSELAB:	POP Z	CLOSELAB:	MOV (R5)+,@#Z

Example. The memory cell of the function

```
function ann(x,y)=>r;
  vars a1;
  x*x + y*y->a1;
  sqrt(a1)->r
end
```

is shown at Fig. 12. Here x and y are formals, r is an output local, a1 is a local.

The first instruction of each function is always jsr r3,fnent . The subroutine fnent inserts current value of register r3 and values of all formals and locals into the system stack, assigns new values from the user stack to the formals, and gets the address of memory cell of the new function into the register r3. The subroutine fnout , on the contrary, removes the original values of formals and locals and assigns them to those variables, and puts the original address into the register r3 .

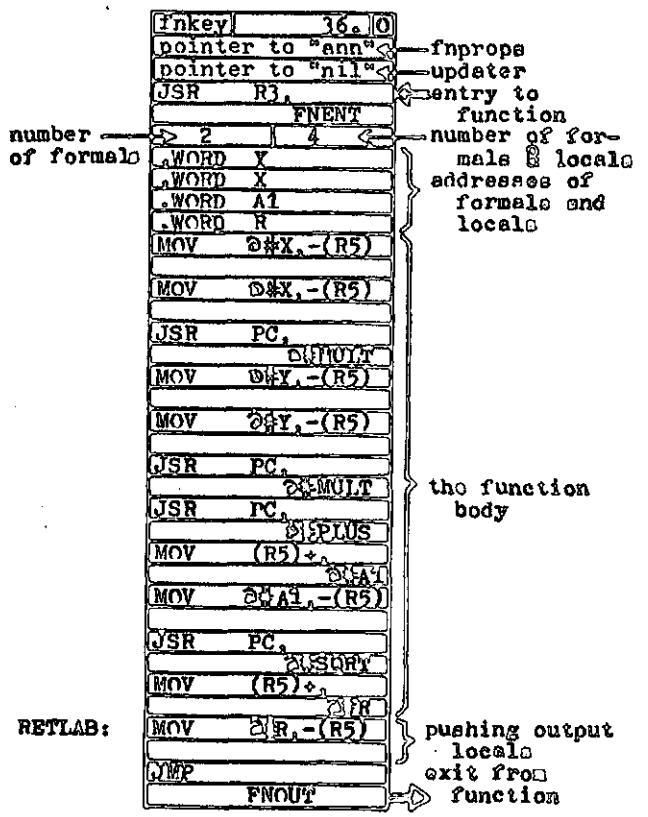


Fig. 12. Representation of the function ann ($fnkey=01_8$).

statement of POP-2	explanation	meta-language	assembler of PDP-11
a;	push the value of the variable a onto the user stack	PUSH A	MOV @#A,-(R5)
const;	push a number or a structure constant onto the user stack	PUSHI CONST	MOV #CONST,-(R5)
->a;	pop the item from the user stack and assign it to the variable a	POP A	MOV (R5)+,@#A
->m;	assign a function from the top of the user stack to the macro/operation m	CHPOP A[1]	TRAP RCHPOP[2] .WORD M
log()	call a standard function	CALLS LOG	JSR PC,@#LOG
alpha()	call a user's function	CALL ALPHA	TRAP RCALL[2] .WORD ALPHA
->alpha()	call the updater of a user's function	CALLU ALPHA	TRAP RCALLU[2] .WORD ALPHA
ftr()	call a FORTRAN subroutine within a POP-2 program	FCALL FTR	TRAP RFCALL[2] .WORD FTR
goto lab	go to the label lab or:	J LAB[4] JB LAB[4]	JMP LAB(R3) [3] TRAP RJB [2] .WORD LAB [3]
return	exit from a function	RETURN	JMP FNOUT [5]
switch ...	switch	SWITCH S	TRAP RSWITCH[2] .WORD S [6]
{if ...} then	conditional jump within if/while expression	JEQ IFLAB [7]	CMP (R5)+,R4 BNE .+6 JMP IFLAB(R3)
{unless...} then	conditional jump within unless/until expression	JNE IFLAB [7]	CMP (R5)+,R4 BEQ .+6 JMP IFLAB(R3)
... and...	conditional jump within and-expression	JZL EE [8]	TRAP RJZL[2] .WORD EE
... or ...	conditional jump within or-expression	JNZL EE [8]	TRAP RJNZL[2] .WORD EE

1) Only functions can be assigned to macros or operations, therefore a check must be carried out.
2) RCHPOP , RCALL , RCALLU etc. are relative addresses of subroutines, which perform prescribed actions, referred to the address which the code jumps into when executing TRAP instruction.
3) LAB means the relative address of destination, referred to the beginning of memory cell.
4) This statement is used when the label occurs before the goto-statement in the function body. Therefore, underflow and overflow of the user stack must be checked.
5) If the compiled function has output locals, return is transleted as JMP RETLAB(R3) where RETLAB labels the instructions for pushing the output locals.
6) S is the number of labels following the word switch .
7) IFLAB is pseudo-label behind close , or else , if it exists.
8) If the expression before and (or) is equal to false (true), it is useless to evaluate following expressions, that is why the code jumps to the end EE of the entire compound expression.

Fig. 11. Survey of instructions of the metalanguage and their translation into the assembler of PDP-11.

version from	Brighton	Edinburgh	Prague
author year of implementation	S.Hardy 1977	W.Clocksin 1979	I.Brůha 1981
operating system	UNIX	UNIX, RT-11 [4]	RSX-11M, RT-11
system POP-2 occupies length of stacks user's memory	12 K words variable 20 K words[2]	12 K words variable 20 K words[2] 16 K words[1]	9 K words 1+1,5 Kwords 21 K words 17 K words[1]
backtracking symbolic arithmetic	no no	no yes	yes no
implementation of PUSH instruction: number of words occupied in function body number of executed instructions	 2 4	 1 7	 2 1
implementation of CALLS instruction: number of words occupied in function body number of executed instructions	 2 7	 1 13	 2 1
implementation of CALL instruction: number of words occupied in function body number of executed instructions	 2 40	 1 46	 2 11
size of the memory cell of a function [3]	10+2v+2i+2r	10+v+u+i+r	8+v+2i+2r
average number of executed instructions per one instruction of the metalanguage	10	16	2

1) the first item is valid for RSX-11M or UNIX operation system, the other for RT-11.
2) this value means the magnitude of both stacks and the user's memory, in all.
3) symbol i means the number of instructions in the function body,
   v the number of locals and formals,
   r the number of output locals,
   u the number of all other variables (including standard ones), numbers and structure constants, occuring in the function body.
4) the version for RT-11 has been modified by Ilona Bellos.

Fig. 15. Comparison of three implementations of POP-2 language for PDP-11.

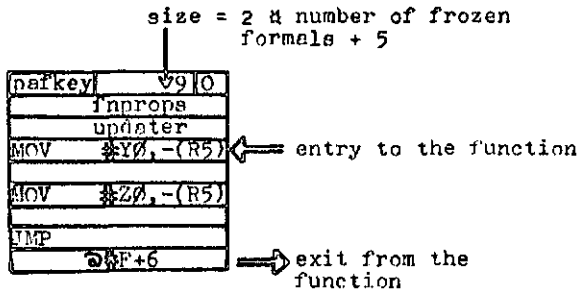


Fig. 13. Representation of a closure (pafkey=02_8).

The POP-2 language enables to freeze one or more formals of functions. The result of freezing formals is a function, called the closure, which is coincident with the original one but some of their formals have fixed, so called frozen values.

Let e.g. function f(x,y,z) have 3 formals; the statement

f(%y0,z0%)

creates a closure, say g, with one formal so that

$(\forall x)\quad g(x) = f(x,y0,z0)$.

The memory cell which represents the closure g is shown at Fig. 13.

4. Implementation of POP-2 system

Representation of POP-2 structures has been solved together with proposal of the garbage collector. I could not choose a concrete type of the garbage collector among all known types, because some of them are based on unrealizable presumptions or do not comply with the above representation of structures. I had to take the following aspects into account:

1) The system stack is not infinite. Therefore, there could arise some difficulties when going through the structures of vast depth.

2) The garbage collector cannot mark single components in a memory cell because in case of PDP-11 the address covers the entire word. It can mark the memory cell as a unit by the GC-bit in its header.

3) The garbage collector must be so versatile to treat memory cells of various sizes and various types of data structures.

4) The garbage collector must be fast because the user's memory can be filled up very easily by the information which is necessary to be saved only for short time or which cannot be called by the system or the user after executing some statements.

The above considerations have led to the following system of garbage collection. It passes the heap (user's memory) four times in all (so-called 4-passage garbage collector):

1.passage: mark those memory cells which can be referred by the user or the system (to any depth of structures);

2.passage: compute shifts for each marked cell;

3.passage: change pointers to all marked memory cells in accordance with the computed shifts;

4.passage: shift all marked memory cells to the bottom of the heap.

The other parts of the POP-2 system are rather simple in comparison with the garbage collector and proposal of the data structures representation.

Lexical analyser gives succesive text items as its result. It must look ahead at most three characters to find out the end of a text item.

Syntax analyser compiles a text from an opening syntax word such as ([if lambda until the corresponding closing word such as)] close end , and calls itself recursively. It is based on the principle of operating precedence analysis.

Fig. 14 shows the survey of parts of POP-2 system, together with the memory they occupy and the time consumed for their proposal and implementation.

5. Conclusion

Problems around one implementation of the POP-2 programming language for the PDP-11 com-

part of POP-2 system	occupies core [K words]	time for implementing [man-month]
lexical analyser	0.62	0.5
syntax analyser and compiler	1.72	1.5
standard functions	3.08	2
input/output	0.92	0.5
memory management	0.54	2.5 ✲)
initiation and error reporting	0.28	0.5
constants, system variables and words	1.91	
the entire system	9.07	7.5

✲) together with proposal of the structure representation

Fig. 14. Parts of POP-2 system.

puters have been presented, especially the representation of data structures and the way of compiling. We conclude by comparing three known implementations of POP-2/PDP-11, namely one from Prague, the other from Great Britain, see Fig. 15.

1. Edinburgh's version [3],[4] is the most memory-saving, but is the slowest. Call of a function is controlled by the system interpreter, which carries out the instructions stored in single words of the memory cell of the called function; the first byte is type of the instruction and the second one represents the operand, better said, its order in a special header of the cell. Execution of functions is, therefore, getting very slow.

Brighton's version uses an interpreter as well, but the first word is an address of a subroutine which carries out the given instruction, the second word is direct the operand. Size of memory cells is approximately double but the execution lasts shorter time.

As for Prague's POP-2, the size of memory cells is a bit shorter than in case of Brighton's one, but execution is the fastest. Only this version directly utilizes the instructions of machine code in memory cells. It is also faster because it does not check the stacks overflow and underflow during each PUSH , POP , CALL etc. instructions, but only when it enters or exits a function. This version has also been written with the aspect that it be changed simply for the VAX-11 computer. One could get a fast version where the problems about sizes of memory cells would be negligible in comparison with the implementation for PDP-11.

2. Dynamic partition of the free memory among data structures and the stacks, which is implemented in Edinburgh's and Brighton's versions, stands for most efficient utilizing of the free memory. On the other hand, the routines for memory management are longer and more complicated.

3. Symbolic arithmetic stands for an important tool for problem solving. Nevertheless, the execution of arithmetic operations over numbers slows down. It is better to preserve the standard arithmetic and to change it only by the user if necessary. On the contrary, I am convinced that the state-saving system and backtracking should be implemented as a standard tool of any programming language for artifial intelligence.

References

[1] R. Popplestone, R. Burstall, M. Collins: Programming in POP-2. Edinburgh University, 1970.

[2] I. Bruha: The POP-2 Programming Language - a conversational language for non-numerical applications (In Czech). VTS FEL ČVUT, Praha, 1980.

[3] W. Clocksin: The Unix POP-2 System. Software Report 4, Dept. AI, Edinburgh University, June 1979.

[4] W. Clocksin: The RT-11 POP-2 System. Dept. of AI, Edinburgh University, June 1979.

IZKUŠNJA S PROLOGOM KOT JEZIKOM ZA SPECIFIKACIJO INFORMACIJSKIH SISTEMOV

DAMJAN BOJADŽIJEV,
NADA LAVRAČ,
IGOR MOZETIČ

UDK: 681.3.0.6:007

INSTITUT JOŽEF STEFAN, JAMOVA 39, LJUBLJANA

Visokonivojski programski jezik PROLOG smo preizkusili kot specifikacijsko orodje pri razvoju zahtevne aplikacije. Članek najprej opisuje problem specifikacije kompleksnih programskih paketov. Nato obravnava prednosti uporabe logike kot formalnega specifikacijskega jezika in PROLOGa kot enostavnega izvedljivega logičnega formalizma specifikacij. Opisano je naše izkušnje pri pretvorbi ISAC specifikacij računalniško podprte konkretnega skladiščnega sistema v PROLOGov program in povratni učinek testiranja PROLOGovega programa na spremenjanje in poglabljanje specifikacij. Podana je naša ocena PROLOGa kot zelo uporabnega specifikacijskega jezika - ob tem navajamo njegove konkretno ugotovljene prednosti in slabosti.

EXPERIENCE WITH PROLOG AS AN INFORMATION SYSTEMS SPECIFICATION LANGUAGE. We have used the high-level programming language PROLOG as a specification tool for the development of a complex application system. The article first describes the problem of specifying complex program packages. Then it describes the advantages of using logic as a formal specification language and PROLOG in particular as a simple runnable formalism. We describe our experience with transforming the ISAC specifications of a computerised warehouse system into a PROLOG program. We also mention the feedback from testing and demonstrating the PROLOG program to the change and further development of specification. The advantages and some weaknesses of PROLOG as a specification language are presented, supporting our opinion that PROLOG is a very useful and efficient specification tool.

1. UVODNI OPIS PROBLEMA

Programski jezik PROLOG smo preizkusili kot specifikacijsko orodje pri razvoju računalniškega sistema za podporo poslovanja skladiščno prodajnega centra Slovenijales v Črnučah. Serva smo se lotili programiranja poenostavljenega modela skladiščnega poslovanja v PROLOGu zaradi želje po testiranju hitrosti programiranja ter obnašanja PROLOGovega programa pri simulaciji nekega realnega procesa. Ker pa se je izkazalo, da lahko PROLOG uporabimo kot zelo učinkovit specifikacijski jezik (s stališča hitrosti programiranja), smo nadaljevali s programiranjem vse realnejšega modela skladišča. Pričakujemo, da nam bo podrobno razdelani program v PROLOGu trdno vodilo pri implementaciji sistema v izbranem programskem jeziku (PASCAL).

2. PROBLEM SPECIFIKACIJE PROGRAMSKIH SISTEMOV

Razvoj korektnega programskega paketa poteka vsaj skozi štiri faze [3]:

- razumevanje problema,
- formalna specifikacija,
- programiranje,
- preverjanje pravilnosti.

V fazi razumevanja problema si načrtovalec v interakciji z uporabnikom ustvari intuitivno sliko problema in izbere pristop k njegovemu reševanju. V naslednji fazi mora jasno in nedvoumno izraziti svojo intuitivno interpretacijo problema v izbranem specifikacijskem jeziku. Na osnovi specifikacije se izdela program, katerega pravilnost pa je potrebno preveriti.

V idealnem primeru bi potekala realizacija sistema zaporedno po opisanih fazah, pri čemer bi se v vsaki fazi vztrajalo pri testiranju in odpravljanju napak do take mere, da se ne bi bilo treba vračati na prejšnje. V praksi pa ne moremo dovolj podrobno vsebinsko preveriti formalnih specifikacij, da bi ugotovili, če ustrezajo uporabnikovim željam. Specifikacije namreč ponavadi niso izvedljive, ročno preverjanje pa je mukotrpen posel. Zato raje začnemo s programiranjem kljub zavesti, da specifikacije najbrž še niso dokončne. V primeru sprememb to povzroči preprogramiranje sistema, kar utegne biti zelo zamudno in neustvarjalno opravilo.

Dejansko je najtežja in najkreativnejša faza v razvoju programskega sistema prav izdelava sprejemljivih formalnih specifikacij iz intuitivne interpretacije problema. Zato je koristno izbrati tako orodje, s katerim je konstruiranje in preverjanje specifikacij čimbolj olajšano. To preverjanje seveda ne more pomeniti dokazovanja pravilnosti specifikacij glede na našo intuitivno sliko, temveč le ugotavljanje ali se na testiranih primerih obnašajo tako, kot smo si želeli in predvideli.

3. POMEN FORMALNIH IN IZVEDLJIVIH SPECIFIKACIJ

Uporaba formalnih specifikacij [10] navaja načrtovalca sistema k odstranjevanju nejasnosti, dvoumnosti in skritih protislovij prvotnih specifikacij, podanih v pogovornem jeziku. Zato ni čudno, da logika že dalj časa služi kot preverjeno formalno specifikacijsko orodje. Logika je lahko razumljiva, opisna in s svojim mehanizmom dokazovanja izrekov omogoča odkrivanje protislovij, ki so se ohranila v formalnih specifikacijah.

če je specifikacijski jezik tudi direktno (strojno) izvedljiv, lahko brez dodatnih naporov preverimo, kako se bo sistem obnašal v posameznih primerih. Tedaj postane metoda "pa poslejmo!" zares operativna, preverjanje intuitivnega razumevanja problema in korektnosti predlagane rešitve v komunikaciji z uporabnikom pa dobi konkretno, "otipljivo" izhodišče. Izvedljivost specifikacij tako omogoča simulacijo obnašanja sistema pred njegovo dokončno, "zaresno" implementacijo, kar prinese vrsto očitnih prednosti.

V primeru specifikacij, podanih v kakšnem logičnem formalizmu, je njihova izvedljivost dana z mehanizmom dokazovanja izrekov [5]. Vprašanje, kakšno bo obnašanje sistema v posameznem primeru, se namreč prevede na vprašanje, ali je formalni zapis pravtnega vprašanja izrek logičnega sistema, katerega tvorijo specifikacijski stavki. Tako lahko preverimo, ali se bo sistem v določenem primeru res obnašal tako, kot si to želimo:

(Ali) vhodu I ustreza izhod I ?

Na enak način lahko izvemo, kakšno obnašanje predvidevajo specifikacije pri določenem vhodu:

(Kateri so izhodi X, da) vhodu I ustreza X ?

Tako lahko enostavno odkrijemo nepravilnost ali nepopolnost specifikacij in jih ustrezno spremenimo.

Opisani način preverjanja specifikacij je sotovo hitrejša in učinkovitejša alternativa običajnemu preverjanju programov, saj

- od uporabnika dobimo povratne informacije dovolj zgodaj
- omejimo se zgolj na vsebinske spremembe
- pri tem se nam ni potrebno ozirati na učinkovitost.

Ker se v komunikaciji z uporabnikom prepričamo v ustreznost (popravljene) formalne rešitve, nam pri preverjanju pravilnosti končnega programa, v nasprotju z običajnim preverjanjem programa, preostane le še odkrivanje programerskih napak.

Uporaba logičnih formalizmov ne narekuje izbire programskega jezika pri končni implementaciji. Dovolj podrobna logična specifikacija se sicer s stališča uporabnika razlikuje od končne verzije sistema načelno samo po učinkovitosti. Drugače rečeno, logična specifikacija je že lahko znosno učinkovita programska rešitev načrtovanega sistema. Izvedljivost logičnih specifikacij tako zabriše tradicionalno razliko med specifikacijo in ustreznim programom [4], [11].

4. PROLOG KOT SPECIFIKACIJSKI JEZIK

Logični formalizmi, med njimi standardno predikatni račun, so bili, vsaj do pojava PROLOGa, širše sprejeti predvsem kot specifikacijski jeziki. Možnost direktne uporabe npr. predikatnega računa kot programskega jezika je v praksi ostala neizkoriščena zaradi neučinkovitosti ustreznih dokazovalcev izrekov. Ta situacija se je spremenila s pojavitvijo PROLOGa [2], ki implementira šibkejši, a izrazno dovolj naraven in uporaben podjezik predikatnega računa. Predikatni račun je v PROLOGu omejen na tim. Hornove stavke, ki izražajo posojne trditve (implikacije) tipa

če Pi in P2 in ... in Pn, potem P

kar se npr. v DEC-10 PROLOGovi sintaksi [6] zapiše kot

P :- P1, P2, . . . , Pn.

V posebnem primeru n=0 so to brezpogojne trditve ali dejstva, kar se v PROLOGovem zapisu izrazi s

P.

PROLOGova omejitev na Hornove stavke je koristna predvsem zato, ker ob enostavni kontrolni strukturi omogoča učinkovito izvajanje sklepov, ki sledijo iz postavljenih trditev. S tem pridobi Hornova logika status visoko-nivojskega, deklarativnega programskega jezika. Logika Hornovih stavkov je tudi bližja standardnim formalizmom računalniške znanosti po svoji proceduralni interpretaciji. Ker namreč Hornovi stavki reduciranje reševanje problema P na reševanje podproblemov P1,...,Pn, lahko v funkciji Pi-jev (in v načinu njihovega aktiviranja) prepoznamo elemente funkcije (in delno načina klicanja) podprogramov v klasičnem programskem jeziku. S tem se načelno zmanjša razlika med specifikacijo problema v PROLOGu in njegovo implementacijo v nekem drugem programskem jeziku.

5. OPIS PROBLEMA IN PROGRAMIRANJE POENOSTAVLJENEGA MODELA V PROLOGU

Skladiščno prodajni center Slovenijales v črnučah potrebuje računalniško podporo poslovanja velikega paletnega skladišča, v katerem se skladišči veliko število različnih vrst artiklov. To skladišče bo prvo v bodoči mreži Slovenijalesovih računalniško vodenih skladišč. Ker bo za komercialne funkcije ter za planiranje nabave in prodaje skrbel centralni informacijski sistem, je računalniška podpora skladišča omejena na vodenje prevzema in izdaje blaga, inventuro, obravnavanje reklamacij, vodenje zalog (količinsko in prostorsko po paletah) ter komunikacijo s centrom.

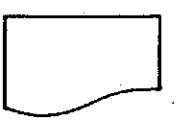
Pri razvoju sistema smo naleteli na precejšnje težave. Omenimo samo slabo definiranost načina poslovanja bodočega skladišča s strani uporabnika, kar je potegnilo za seboj veliko število sprememb specifikacij. Zato smo že po nekajkratnem spreminjanju specifikacij začutili potrebo po poenostavljenem in lahko spremenljivem modelu poslovanja skladišča.

V začetni fazi specificiranja problema smo kot orodje uporabili ISAC metodologijo. Ta metodologija se je izkazala kot koristna pri komunikaciji z uporabnikom zaradi možnosti grafične predstavitve problema. ISAC grafi namreč omogočajo strukturiran pregled nad aktivnostmi ter pretoki materialov in informacij v sistemu [9].

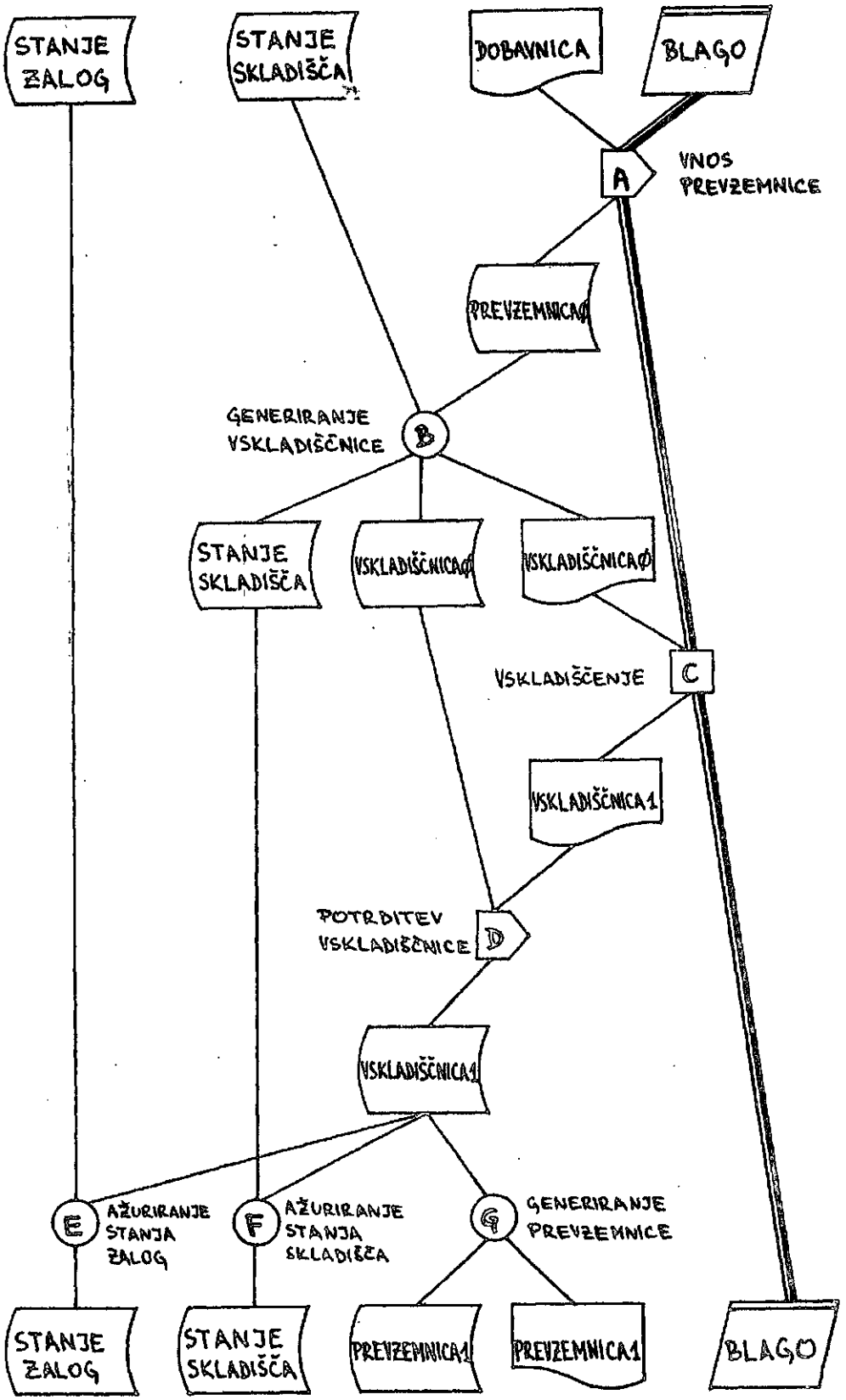
Slika prikazuje del ISAC grafa, ki poenostavljeno modelira prevzem blaga. V grafu smo uporabili naslednje simbole:

- za vhodno - izhodne množice:

datoteka



listina



V akciji A skladiščnik preko terminala vnese šifre prevzetih artiklov. Na osnovi vnesenih podatkov v akciji B sistem predlaga paletna mesta, na katera naj se blago vskladišči. Izpiše se dokument vskladiščnica, na podlagi katerega se blago fizično vskladišči. Če iz kakih razlogov vskladiščenje na predvidena paletna mesta ni bilo mogoče, se spremembe registrirajo v akciji D. Na podlagi tako popravljene vskladiščnice so avtomatsko ažurirata stanje zalog (kumulativne količine artiklov) in stanje skladišča (količine artiklov po paletah), generira in izpiše se prevzemnica.

- za aktivnosti:
  -  avtomatizem
  -  interakcija človek-stroj
  -  ročno

Zaradi nenehnega spreminjanja specifikacij je uporaba ISAC metodologije zahtevala poleg kreativnega še ogromno ročnega dela z dokumentiranjem razvoja v različnih besedilih, tabelah in grafih. Poleg tega nam s tako predstavljeno problematiko ni uspelo poglobiti specifikacij od nekega nivoja dalje.

V tej fazi razvoja sistema smo začutili potrebo po drugačni sistematizaciji funkcij in gradnikov sistema. Ob določitvi gradnikov smo usmerili, da se funkcije lahko realizirajo z različnimi kombinacijami teh gradnikov. Ker se je izkazalo, da je v PROLOGu enostavno realizirati in sestavljati osnovne gradnike, smo se lotili programiranja teh gradnikov in sestavljanja poenostavljenega modela sistema.

Za pisanje programa, ki je obsegal približno 150 vrstic PROLOGove kode, smo porabili pičla dva dneva. Zaradi hitrosti in preprostosti programiranja v PROLOGu smo model razvijali naprej. Osnovne postopke smo približali realnim zahtevam ter uvedli še preostale funkcije sistema. V naslednji fazi smo v programu uporabili tako organizacijo podatkov, kakršna bo realizirana z izbranim sistemom za delo s podatkovnimi bazami (TOTAL). V komunikaciji z uporabnikom smo se približali bodoči realni verziji in razdelali spremljajoče podatkovne strukture. Opisani model, katerega smo realizirali v približno enem mesecu, je obsegal približno 1000 vrstic PROLOGove kode. Pri demonstraciji delovanja modela uporabniku smo (po pričakovanju) dobili veliko novih informacij, ki so narekovale nadaljnje spreminjanje modela.

Naslednja verzija modela je obsegala približno 1400 vrstic programa, ko je pred njeno dokončno zaokrožitvijo (t.j. zamrznitvijo specifikacij) in demonstracijo uporabniku prišlo do novih, resnejših sprememb v zasnovi bodočega skladišča. Končna varianta fizične konstrukcije objekta je namreč narekovala nekatere drugačne rešitve.

Če bo PROLOGov model bodočega sistema dobro razdelan in bo ohranil sedanjo, po vsem sodeč dobro strukturiranost, pričakujemo, da se bomo ob implementaciji sistema v izbranem programskem jeziku lahko pretežno omejili na prepisovanje algoritmov, učinkovito realizacijo baze podatkov, programiranje vhodno - izhodnih procedur, dodajanje podrobnosti in eventualno povečanje učinkovitosti sistema. Program v PROLOGu bi moral služiti tudi kot solidna osnova za dokumentiranje programskega paketa.

6. PREDNOSTI IN SLABOSTI PROLOGA ZA HITRO PROGRAMIRANJE SPECIFIKACIJ

Pri implementaciji modela bodočega skladiščnega sistema so se pokazale marsikatere dobre lastnosti PROLOGa [8].

V veliki meri smo se naslanjali na usodnosti, ki sledijo iz načelne zadostnosti deklarativnega opisa problemov. PROLOG namreč omogoča povsem deklarativno programiranje, ker ima že vgrajeno kontrolno strukturo. Razen možnosti, da omejimo sicer nedeterministično izvajanje programa, PROLOG ne potrebuje nobenih kontrolnih konstruktov (npr. iteracije). Tako se pri definiciji algoritmov lahko ukvarjamo le z vsebino problema, saj je za postopek reševanja že poskrbljeno.

Ker delovanje PROLOG programa temelji na unifikaciji, eksplicitni prireditveni konstrukti niso potrebni (razen pri vrednotenju aritmetičnih izrazov), kar omogoča hitrejše pisanje jasnih in jedrnatih programov.

K jedrnatosti programov prispeva tudi dejstvo, da deklaracije podatkovnih struktur niso potrebne [1]. Oblika zapisa termov že sama implicitno definira enega od splošnih podatkovnih tipov, za dinamično dodeljevanje pomnilnika med izvajanje programa pa skrbi kontrolna struktura. S funktorji lahko terme povezujemo med seboj v kompleksne podatkovne strukture. Ker ni potrebna vnaprejšna definicija tipov podatkov, lahko z uporabo funktorjev strukture poljubno poslahljamo, ne da bi nam bilo zato potrebno spreminjati višje nivoje programov in ne da bi bilo treba strukturo predvideti v podrobnosti vnaprej, kar je za hitro programiranje bistvenega pomena.

Jedrnatost PROLOG programov in fleksibilnost omenjanja podatkovnih struktur ponazarja prepis grafa na sliki 1 v program:

```
prevzem :-
    vnos_prevzemnice (Prevzemnica_0),
    generiranje_vskladiscnice (Prevzemnica_0, Vskladiscnica_0),
    vskladiscenje,
    potrditev (Vskladiscnica_0, Vskladiscnica_1),
    azuriranje_stanja_skladisca (Vskladiscnica_1),
    azuriranje_stanja_zalog (Vskladiscnica_1),
    generiranje_prevzemnice (Vskladiscnica_1, Prevzemnica_1).
```

V njem so v PROLOGovi minimalni sintaksi posamezne aktivnosti grafa proglašene za procedure, oznake podatkovnih množic grafa pa so prevzete kot formalni parametri teh procedur. Ti označujejo še nespecificirane podatkovne strukture, ki se seveda razgradijo na nižjih nivojih programa. Tako smo npr. proceduro, ki iz prevzemnice generira vskladiščnico izrazili kot rekurzivno relacijo med seznamom artiklov s količinami in seznamom artiklov s pripadajočim seznamom palet in količin na njih:

```
generiranje_vskladiscnice ([],[]).

generiranje_vskladiscnice ([Artikel:Kolicina|Prevzemnica],
                           [Artikel:Palete_Kolicine|Vskladiscnica]) :-
    stanje_skladisca (Prazne_palete_0),
    predlog_praznih_palet (Artikel, Kolicina, Palete_Kolicine,
                           Prazne_palete_0, Prazne_palete_1),
    brisi (stanje_skladisca (Prazne_palete_0)),
    vrisi (stanje_skladisca (Prazne_palete_1)),
    generiranje_vskladiscnice (Prevzemnica, Vskladiscnica).
```

Ker so v PROLOGu vse spremenljivke lokalne stavku, v katerem se nahajajo, je preglednost in berljivost posameznih programov ustrezno večja kot v konvencionalnih programskih jezikih. Stavki ene procedure so med seboj neodvisni, kar pomeni, da jih lahko dodajamo in odvzemamo, ne da bi morali spreminjati preostale. Seveda pa to ne velja, če imamo pri programiranju posameznih stavkov procedure že v mislih njihovo postopkovno interpretacijo. Ta način programiranja smo sicer pogosto uporabljali, saj si z njim prihranimo dodatno pisanje in povečamo preglednost programov. Zaradi preglednosti programov in lokalnosti spremenljivk je lažje razbiti sistem v neodvisne module in ga strukturirati.

Pravilnost programov v PROLOGu je razmeroma enostavno empirično preverjati. Ker ni nobene

formalne razlike med programi in podprogrami, lahko neposredno testiramo in preverjamo posamezne dele sistema. Pri odkrivanju napak si lahko pomagamo z že vgrajenim mehanizmom za sledenje izvajanja programa, pri katerem je možno selektivno izločanje nezanimivih delov. V RT-11 PROLOGu [2], katerega smo uporabljali, je to možno le do neke mere.

Omejene vhodno-izhodne možnosti PROLOGa nas niso ovirale pri realizaciji modela, nasprotno, v začetni fazi smo izkoristili možnost preprostega izpisovanja poljubno kompleksnih podatkovnih struktur. Podatkovne strukture smo namreč pretežno realizirali v obliki termov, ki so direktno izpisljivi.
Omenimo še nekatere lastnosti PROLOGa, ki so se pri našem delu izkazale kot neugodne.

Čisti PROLOG, kot rečeno, nima globalnih spremenljivk (kar smo mu zgoraj štteli v dobro), dovoljuje pa njihovo simulacijo z dodatnimi meta-logičnimi konstrukti. Zanesljiva uporaba teh konstruktov zahteva določeno mero natančnega postopkovnega razmišljanja, ki lahko postane delikatno zlasti pri realizaciji podatkovnih struktur v obliki množice dejstev (namesto enega "seznamskega" dejstva).

Zasledovanje izvajanja programa pri odkrivanju napak je včasih nekoliko težje kot sledenje izvajanja programa v drugih programskih jezikih zaradi njegovega nedeterminizma.

Pri realnih aplikacijah bi se kot PROLOGova hiba izkazalo dejstvo, da PROLOG ne podpira realnih števil. Implementacija, katero smo uporabljali (RT-11 PROLOG), pa ne podpira niti negativnih celih števil, kar je zahtevalo nekaj dodatnega dela pri implementaciji modela.

7. ZAKLJUČEK

Iz povedanega je razvidno, da so naše izkušnje pri uporabi PROLOGa kot specifikacijskega jezika zelo pozitivne. Programiranje v PROLOGu je hitro in zanimivo. PROLOGov model omogoča vodeno poslabljanje specifikacij, simulacijo obnašanja sistema načrtovalcu in uporabniku ter jasno zasnovo realne programske rešitve. Pri bodočih aplikacijah se bomo gotovo mnogo prej lotili programiranja modela v PROLOGu, kar bo omogočilo učinkovitejšo interakcijo z uporabnikom in hitrejšo definicijo problema.

8. LITERATURA

1. I. Bratko, M. Gams: Prolog: Osnove in principi strukturiranja podatkov; Informatica, letnik 4, št. 4, 1980

2. W. F. Clocksin, C. S. Mellish: Programming in Prolog; Springer-Verlag, 1982

3. R. E. Davis: Runnable specification as a Design Tool; Proceedings of the Logic Programming Workshop, Debrecen, Hungary, 1980

4. R. Kowalski: Logic as a Computer Language; Dept. of Computing, Imperial College, London, 1981

5. R. Kowalski: Logic for Problem Solving; AI Series, Elsevier North Holland, 1979

6. L. M. Pereira, F. Pereira, D. Warren: User's Guide for DEC-10 PROLOG; Dept. of Artificial Intelligence, University of Edinburgh, 1978

7. E. Santane-Toth, P. Szeredi: PROLOG applications in Hungary; Proceedings of the Logic Programming Workshop, Debrecen, Hungary, 1980

8. P. Szeredi, K. Balogh, E. Santane-Toth, Zs. Farkas: LDM - a Logic Based Software Development Method; Proceedings of the Logic Programming Workshop, Debrecen, Hungary, 1980

9. P. Tancig et al.: Nekaj izkušenj z ISAC metodologija razvoja informacijskih sistemov - I. del (analiza IS); Zbornik radova IX Jugoslovenskog savetovanja o informacionim sistemima, Beograd, 1980

10. W. M. Turski: Design of Large Programs; Notes for seminar Informatica, Ljubljana, Yugoslavia, 1981

11. G. Winterstein, M. Dausmann, G. Persch: Deriving Different Unification Algorithms from a Specification in Logic; Proceedings of the Logic Programming Workshop, Debrecen, Hungary, 1980

INFORMATICA 3/1982

SINOPTIKA MIKRORAČUNALNIŠKO VODENEGA PROCESA

I. LESJAK,
R. TROBEC,
M. ŠUBELJ

UDK: 681.327.12

INSTITUT „JOŽEF STEFAN" LJUBLJANA

Opisana je interaktivna podporna programska oprema za realizacijo slike sinoptične sheme industrijskega procesa na CRT ekranu. Avtomatsko tvorjenje slike je zahteven posel, zato je ta programska oprema uresničena z mikroračunalnikom.

THE CONTROL PANEL OF CONTROL SYSTEM: A simple method of representing industrial systems via graphics CRT(s) is described.

UVOD

Naše vodilo je bila želja, da bi operater dokaj enostavno, s pomočjo nabora standardnih elementov in skice ustvaril sliko industrijskega procesa. V ta namen smo razvili:

- simbolni jezik, ki omogoča enostaven in skrčen slikovni opis standardnih elementov industrijskega procesa,
- knjižico slik standardnih elementov industrijskega procesa,
- program za oblikovanje in izdelavo poljubne slike,
- interpreter, ki preslika zapis o sliki industrijskega procesa v assemblerske tabele za mikroračunalnik.

OPREMA

Za prikazovalno enoto izberemo prikazovalnik, ki zadovoljuje naslednje zahteve:

- brisanje znaka, vrstice, ekrana, levo in desno od kurzorja,
- relativno in absolutno premikanje kurzorja po ekranu prikazovalnika,
- premikanje (drsenje) vidnega polja,
- risanje ravnih in vogalnih črt v grafičnem načinu,
- izpis nabora ASCII znakov z atributi (utrip,negativ, svetlobno poudarjanje, dvojna širina).

Grafičnemu prikazovalniku smo se izognili zaradi njegove cene in uvoza, medtem ko prikazovalnike z zgoraj navedenimi zahtevami ponujajo na domačem tržišču.

ENOTA IN STANDARDNI ELEMENTI

Prikazovalnik s svojimi dimenzijami (24 vrstic in 80 ali 132 stolpcev) in abeceda standardnih likov ter boljša preglednost vsebine slike, so nas privedli do izbire enote prikazovalnega polja na ekranu. Enoto predstavljajo po trije znaki v treh vrsticah, tako, da celotno sliko sestavimo iz 8 x 25 "najmanjših" enot. Če moramo lik ploskovno opisati z več kot devetimi znaki, to storimo tako, da ga rišemo preko meja enote. Nekaj likov iz abecede presega dimenzije enote in se ploskovno širijo preko dveh, štirih ali več enot v vse smeri. Rišemo ga tako, da dodajamo enote (3x3), dokler ni lik narisan.

Industrijski procesi, katerih sinoptične sheme smo želeli predočiti z grafično sliko, so vezani na predelavo odpadnih vod ali pa na upravljanje s pitno vodo (filterske postaje, čistilne naprave, kanalizacije, vodovodi). Zato smo s simbolnim jezikom opisali v knjižici standardnih likov elemente kot so:

- cevovod - hidrofor
- rezervoar - zasun
- črpalka - izvir
- vodnjak - zajem
- merilna točka.

Standardni elementi imajo različne oblike, tako da se lahko mozaično sestavljajo v celoto slike. Knjižnica vsebuje več različnih oblik cevovodov, rezervoarjev, ..., tako, da se povezujejo z okoliškimi liki v smiselno celoto slike. Abecedo standardnih likov se lahko dovolj enostavno spremeni ali dopolni s standardnimi liki drugih industrijskih procesov (kemična industrija,elektrodistribucija ...)

NAČRTOVANJE DIALOGA

Z upoštevanjem psihološke obravnave človeka operaterja lahko najbolje oblikujemo dialog. Občasen, aktiven uporabnik podporne programske opreme za realizacijo slik

sinoptične sheme mikroračunalniško vodenih procesov je bil naše vodilo pri oblikovanju dialoga. Dialog vodi operater sam. Omogočeno mu je, da prekine z delom in kasneje nadaljuje na prekinjenem mestu. Program za oblikovanje slik nudi možnost izstopa iz programa, ohranitev do tedaj oblikovane slike in parametrov ter možnost ponovnega zagona s privzeto shranjeno sliko in parametri. Operater komunicira s programom v "menu selection" tehniki, delno pa se dialog odvija z vprašanji in odgovori.

OBLIKOVANJE SLIKE

Operater oblikuje sliko tako, da v prikazan koordinatni sistem z enoto (3x3) vstavlja like iz knjižnice in tako jih kot lepljenko zloži v sinoptično shemo industrijskega procesa. Operater najprej izbere položaj v koordinatnem sistemu. Položaj izbere tako, da vnese x in y koordinato točke, ki predstavlja levi zgornji vogal enote. Slikanje lika poteka na enotskem prostoru desno in navzdol od izbrane izhodiščne točke. Lik, ki bi ga rad naslikal je določen z imenom lika (cevovod, hidrofor, črpalka ...). Oblika lika je v neposredni zvezi z njegovo zaporedno številko lege v tistem delu knjižnice, ki je določen z imenom lika. Ker je od operaterja nemogoče zahtevati, da pozna vse oblike likov s številkami, mu omogoči program listanje oblik izbranega lika na izbrano mesto v prikazanem koordinatnem sistemu na ekranu. Možnost za določitev oblike lika s številko je ponujena kot bljižnica v dialogu. Ob pogostejši uporabi programa, si lahko zapomnimo nekaj številk, ki predstavljajo določeno obliko lika.

```
3,4,5
lqk
$1,-3u& }tqqqqqq
$1,-9mq5
$-2,07
*$2,-5%
£
```

Slika 1. Zapis elementa v knjižnici

Nekateri liki se v industrijskih procesih in na sliki ponavljajo, zato ni dovolj, da so opisani samo z obliko, temveč potrebujejo za medsebojno razlikovanje še dodatne informacije:

- oznaka lika (Z-1 (zasun 1), R-3 (rezervoar 3), ... = tekst)

- oznaka meritve (P (pritisk), Q (pretok), T (temperatura), C (Klor), ... = znak)

- enota meritve(%,M, kg/cm3, mg/l, ... = znaki)

- ime terminala (poseben lik ... = tekst).

Pri avtomatskem izdelovanju slike mora operater sproti vnašati pravilne odgovore na zahteve, ki dodatno opisujejo lik. Pravilne odgovore na oznake likov, meritev in enot meritev poiče operater v skicah projektantov industrijskega procesa (npr. hidravlične sheme, projekt energetike, projekt avtomatike ...).

Iluzorno je, da bi lahko operater samo z naborom standardnih likov izrisal katerokoli shemo industrijskega procesa. Zato mu program nudi možnost interaktivnega risanja poljubnih likov v grafičnem načinu. Operater preko tastature vnaša črke, ki jih program odda prikazovalniku v grafičnem načinu. Ob interaktivnem risanju so možni tudi relativni in absolutni pomiki kurzorja, kar omogoča hitrejše risanje (premik s puščico samo za eno mesto: levo, desno, gor, dol). Prav tako program omogoča operaterju vstavljanje poljubnega pojasnjevalnega teksta v katerikoli del slike.

SIMBOLNI JEZIK

Program izpolnjuje zahteve s tem, ko prebere in pravilno odpošlje prikazovalniku zapis o liku iz knjižnice. Liki v knjižnici so opisani z znaki iz standardnega ascii znakovnega nabora. Nekateri od teh znakov imajo poseben pomen:

"ß" - oddaja niza za relativni pomik (ukaz: /ß X,Y/ ... relativni premik kurzorja za X in Y)
"≈" - oddaja niza za absolutni premik (ukaz: /≈ X,Y/... absolutni premik kurzorja na koordinate X,Y)
"o" - izpis vrednosti meritve (ukaz: /o/ ... štirimestno število izpisano v enem od formatov: 0.XXXX, X.XXX, XX.XX, XXX.X, XXXX)
"&" - izpis vrednosti signala (ukaz: /&/ ..., on-negativ, off-normalno, on in okvara - povečana intenziteta in utrip, off in okvara - utrip)
"Z" - izpis oznake lika (ukaz: /Z/ ... izpis oznake max. 5 znakov)
"{" - oddaja niza za atribut negativ (ukaz: /{/ ... znaki ki sledijo bodo izpisani reverzno kot ozadje)
"}" - oddaja niza za izklop atributov (ukaz: /}/ ... sledi normalen način izpisovanja znakov)
"<" - oddaja niza za dvojno širino znakov (ukaz: /</ ... znaki v vrstici so izpisani z dvojno širino)

Ostali znaki imajo običajen pomen interpretiran v grafičnem načinu (vodoravne in navpične črte, vogali, križ, ...)

ZAPIS O SLIKI

Organizacija operacijskega sistema in stališče preglednosti sta narekovala kratko, enostavno in pregledno izhodno datoteko. V njej so shranjeni kazalci na elemente v knjižnici in vnešene so izpolnjene dodatne zahtove za medsebojno razlikovanje likov (kazalcem so pripisano še oznake, tipi in enote meritev in ime terminala). Ob interaktivnem risanju poljubnih likov v grafičnem načinu, program shranjuje te like v posebno datoteko. V zapisu o sliku so tudi kazalci v to datoteko s poljubnimi liki. Ob ponovnem zagonu programa se lahko naslonimo na že ustvarjeno sliko, lahko jo privzamemo, lahko jo popravljamo, dopolnjujemo. Popravljanje pomeni samo spreminjanje vsebine zapisa o sliki (drugi kazalci ali drugačne oznake). Poznavanje slike, elementov knjižnice in oznak nam omogoča, da manjše popravke lahko opravimo kar s sistemskim editorjem v datoteki z zapisom o sliki.

```
CEV       1              1  1
...
0         0              2  2
REZM      8    R-1       2  3
0         0              2  4
0         0              2  5
CEV       7              2  6
CEV       1              2  7
CEV       1              2  8
CEV       6              2  9
0         0              2 10
0         0              2 11
CRP       3    C-1       2 12
0         0              2 13
...
0         0              7 21
OSTA01    1              7 22
0         0              7 23
0         0              7 24
0         0              7 25
0         0              8  1
0         0              8  2
0         0              8  3
0         0              8  4
IME       1SEVER-4       8  5
0         0              8  6
...
```

Slika 2. Zapis o sliki (prikaz dela datoteke).

INTERPRETER

Struktura programov in struktura tabel za risanje sinoptike mikroračunalniško vodenega procesa je opisana v prvem delu tega članka. Rezidentni program za risanje slike se naslanja s pomočjo imenika na tabele, v katerih so shranjeni vsi liki in oznake, ki predstavljajo sliko sinoptične sheme. Avtomatsko tvorjenje tabel opravi program interpreter, ki preslika zapis o sliki industrijskega procesa v assemblerske tabele za mikroračunalnik. Program iz vhodne datoteke najprej izlušči vse specialne znake in jih prepiše v ukaze rezidentnemu programu (npr. pogled v ram - spremenljive vrednosti). Operater interaktivno dopolnjuje ukaze. Vnašati mora naslove in maske spremenljivih vrednosti v ram pomnilniku. Z masko, ki jo oblikuje operater, omogočimo bitno predstavitev podatkov (maska za on-off, *maska za okvaro*). Pri ukazih za izpis merjenih vrednosti mora operater vnesti podatek o naslovu meritve in o tipu obdelave ali formatu izpisa. Tako program zgradi imenik in poimenuje vse ukaze za sliko na ekranu. Sliko program prevede v assemblerske tabele tako, da v ustreznem formatu in z ustreznimi tabelami iz imenika preslika znake iz knjižnice v izhodno datoteko. Rezidentni program za risanje sinoptike relativno (preko imenika) naslavlja ukaze in like, ki jih uresničuje in riše na prikazovalnik. Relativno naslavljanje preko imenika omogoča enostavno spreminjanje tabel v primerih povečevanja in spreminjanja slike ali dodajanja teksta...).

```
*
*
IMAUKO DC   2,UKO  (IMENIK UKAZOV SLIKE 'O')
*
*SEGMENT 1  LIK: REZM     OZNAKA:R-1
UKO     DC  D'1'         POLOZAJ PIKE
        DC  2,H'FC00'    AORESA MERITVE
*
*
*
*SEGMENT 1
SLIEKO DC   H'7E'        ABSOLUTNI POMIK "~"
       DC   H'0407'      Y,X
       DC   H'24'        RELATIVNI POMIK
       DC   H'00'        SMER
       DC   2,H'0101'    Y,X
       DC.  5,C'R-1      OZNAKA LIKA
       DC   H'24'        RELATIVNI POMIK
       DC   H'11'        SMER
       DC   2,H'0106'    Y,X
       DC   D'107'       k
       DC   D'120'       x
       DC   H'20'        SPACE
       DC   H'20'        SPACE
       DC   H'20'        SPACE
       DC   H'108'       l
       DC   H'24'        RELATIVNI POMIK
       DC   H'01'        SMER
       DC   2,H'0106'    Y,X
       DC   D'120'       x
       DC   D'123'       {
       DC   H'20'        SPACE
       DC   H'20'        SPACE
       DC   H'20'        SPACE
       DC   H'20'        SPACE
       DC   D'125'       }
       DC   D'120'       x
       DC   H'24'        RELATIVNI POMIK
       DC   H'01'        SMER
       DC   2,H'0106'    Y,X
       DC   D'109'       m
       DC   D'119'       w
       DC   D'113'       q
       DC   D'113'       q
       DC   D'119'       w
       DC   D'106'       j
       DC   H'24'        RELATIVNI POMIK
       DC   H'01'        SMER
       DC   2,H'0106'    Y,X
       DC   2,C'*'       IZPIS MERITVE
       DC   D'77'        m
*
       DC   H'40'        KONEC EKRANA '£'
*
```

Slika 3. Prikaz dela imenika in tabel

SLABOSTI OPISANE PODPORNE PROGRAMSKE OPREME

- Risanje od levega zgornjega kota navzdol (neenakomerna velikost likov);
- brez poševnih in okroglih črt;
- v koordinatnem sistemu naslavljamo samo skupine znakov (enota).

DOBRE LASTNOSTI

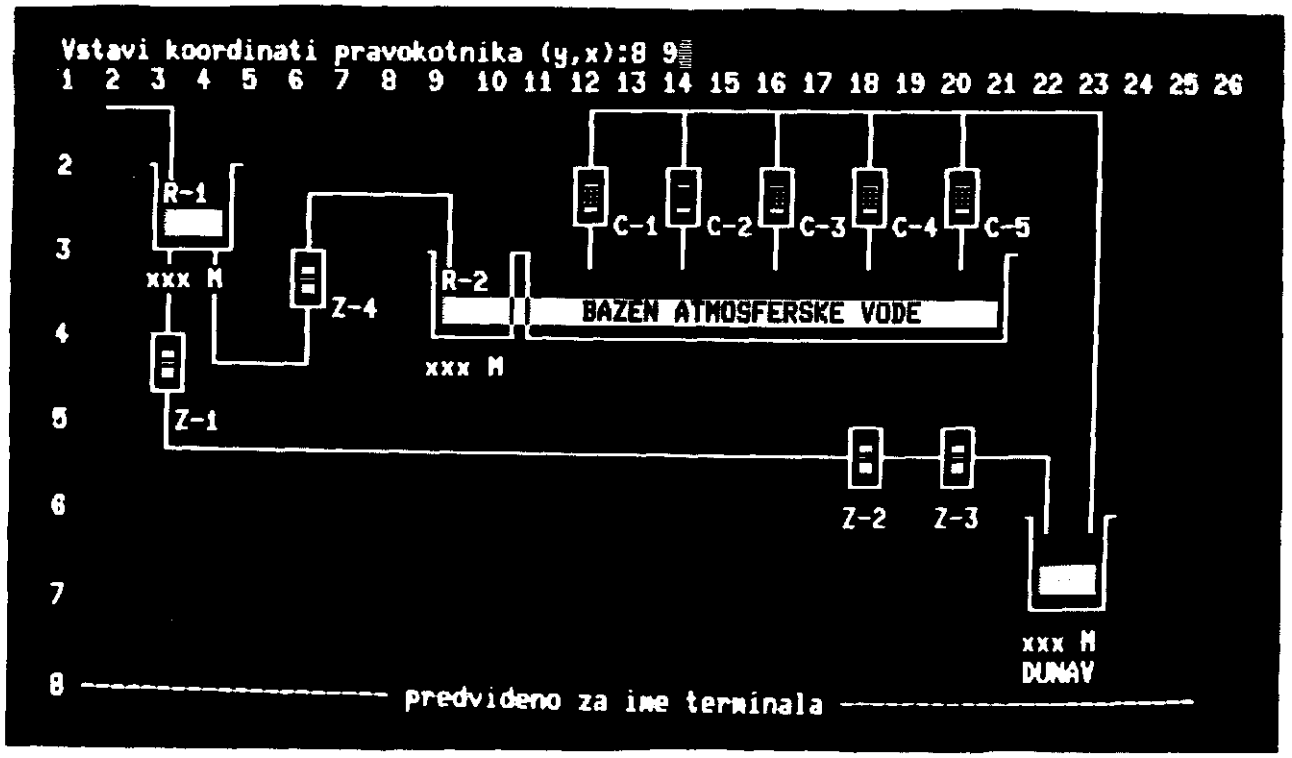
- Lahko poljubno spreminjamo like v knjižnici; (hitro in enostavno spreminjanje)
- lahko popravljamo in dopolnjujemo sliko, lahko prekinemo z delom;
- hitro in enostavno risanje slike;
- enostavna komunikacija s programi;
- ustvarjena slika je hkrati statična simulacija sinoptične sheme.

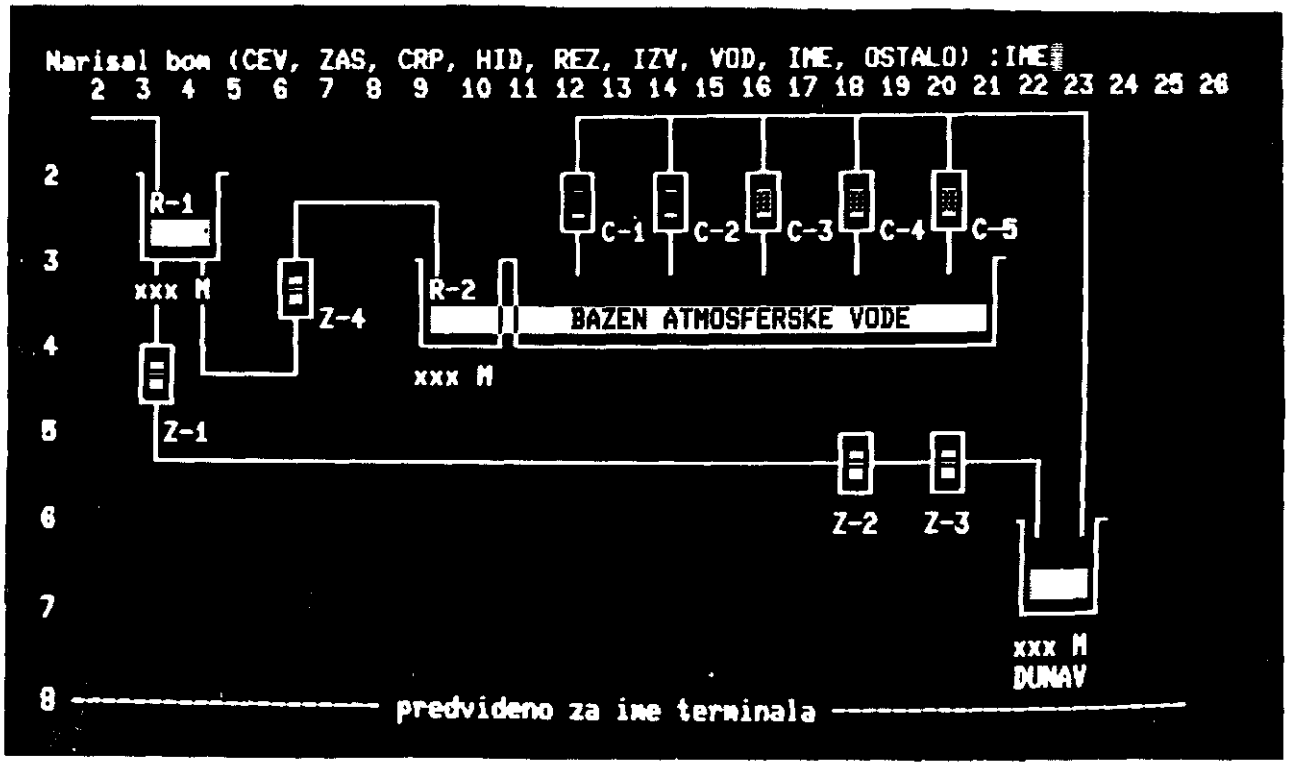
LITERATURA:

(1) A.B.Anue:
Status and trends in man-machine communication real-time data handling and process control, North-Holland, pp 145-151, Brussels and Luxemburg 1980

(2) I.Lesjak, M.Šubelj, R.Trobec
Načrtovanje dialoga človek računalnik pri vodenju procesov
Referat 120 - Knjiga I., V.Bosanskohercegovački simpozijum iz informatike,
Jahorina 1981, Jahorina, 23.-27.marca 1981

(3) Dokumentacija sistema za zbiranje podatkov o kanalizaciji Novi Sad
IJS 1981





Slika 5. Dialog s pomočjo prikazovalnika

UPORABNI PROGRAMI

Konverzija zbirne mnemonika procesorja 8080A v mnemonika procesorja Z80


```
**************************
*  Informatica UP 8      *
*  Primerjalna tabela    *
*  avgust 1982           *
*  Anton P. Železnikar   *
**************************
```


Področje uporabe

Pri programiranju v CP/M okolju, za katero sta predvidena procesorja 8080A in Z80, želimo večkrat pretvoriti obstoječo Intelovo zbirno mnemoniko v Zilogovo. Mnemonika Ziloga je namreč lažje razumljiva (si jo lažje zapomnimo). Večkrat želimo tudi obstoječe programe za procesor 8080A modificirati, imamo pa sistem s procesorjem Z80 (tudi zbirnik za njegovo mnemoniko). V takih primerih se modifikacije lotimo tako, da prevedemo obstoječi kod za 8080A z inverznim zbirnikom za procesor Z80. V tem primeru seveda nimamo težav, saj se to delo opravi avtomatično.

Na pravo potrebo za konverzijo mnemonike pa naletimo tedaj, ko so določeni programski segmenti napisani v zbirnem jeziku procesorja 8080A. In v teh primerih lahko pokličemo pri modifikaciji programa na pomoč našo tabelo.

Konverzijska tabela mnemonike

V prvem stolpcu spodnje tabele imamo Intelov zbirni format, v drugem stolpcu pa Zilogov.

ACI N	ADC A,N
ADC reg	ADC A,reg
ADC M	ADC A,(HL)
ADD reg	ADD A,reg
ADD M	ADD A,(HL)
ADI N	ADD A,N
ANA reg	AND reg
ANA M	AND (HL)
ANI N	AND N
CALL NN	CALL NN
CC NN	CALL C,NN
CM NN	CALL M,NN
CMA	CPL
CMC	CCF
CMP reg	CP reg
CMP M	CP (HL)
CNC NN	CALL NC,NN
CNZ NN	CALL NZ,NN
CP NN	CALL P,NN
CPE NN	CALL PE,NN
CPI N	CP N
CPO NN	CALL PO,NN
CZ NN	CALL Z,NN
DAA	DAA
DAD regpair	ADD HL,regpair
DCR reg	DEC reg
DCR M	DEC (HL)
DCX regpair	DEC regpair
DI	DI
EI	EI
HLT	HALT
IN N	IN a,(N)
INR reg	INC reg
INR M	INC (HL)
INX regpair	INC regpair
JC NN	JP C,NN
JM NN	JP M,NN
JMP NN	JP NN
JNC NN	JP NC,NN
JNZ NN	JP NZ,NN
JP NN	JP P,NN
JPE NN	JP PE,NN
JPO NN	JP PO,NN
JZ NN	JP Z,NN
LDA NN	LD A,(NN)
LDAX rgpair	LD A,(rgpair)
LHLD NN	LD HL,(NN)
LXI regpair,NN	LD regpair,NN
MOV reg,reg	LD reg,reg
MOV M,reg	LD (HL),reg
MOV reg,M	LD reg,(HL)
MVI reg,N	LD reg,N
MVI M,N	LD (HL),N
NOP	NOP
ORA reg	OR reg
ORA M	OR (HL)
ORI N	OR N
OUT N	OUT (N),A
PCHL	JP (HL)
POP regpair	POP regpair
PUSH regpair	PUSH regpair
RAL	RLA
RAR	RRA
RC	RET C
RET	RET
RIM	*
RLC	RLCA
RM	RET M
RNC	RET NC
RNZ	RET NZ
RP	RET P
RPE	RET PE
RPO	RET PO
RRC	RRCA
RST n	RST n
RZ	RET Z
SBB reg	SBC A,reg
SBB M	SBC A,(HL)
SBI N	SBC A,N
SHLD NN	LD (NN),HL
SIM	*
SPHL	LD SP,HL
STA NN	LD (NN),A
STAX rgpair	LD (rgpair),A
STC	SCF
SUB reg	SUB reg
SUB M	SUB (HL)
SUI N	SUB N
XCHG	EX DE,HL
XRA reg	XOR reg
XRA M	XOR (HL)
XRI N	XOR N
XTHL	EX (SP),HL

---

reg	...	A, B, C, D, E, H, L
regpair	...	BC, DE, HL, SP or (if PUSH/POP) PSW (AF in Z80)
rgpair	...	BC, DE
N	...	8-bitna vrednost (en zlog)
NN	...	16-bitna vrednost (dva zloga)
n	...	0, 8, 10H, 18H, 20H, 28H, 30H, 38H
M	...	(HL) (kazalec v pomnilnik)
*	...	ukaz procesorja 8085, ki nima Z80 predstavitve

MARIENBAD: igra za otroke, ki spoznavajo funkcije tastature in zaslona

```
Informatica UP 9
MARIENBAD
oktober 1982
B. Blatnik
sistem CP/M, PASCAL/Z
```

Marienbad je enostavna igra. Njen glavni namen je seznaniti računalniškega uporabnika-začetnika s funkcijami tastature in zaslona. Primerna je predvsem za osnovnošolsko mladino. Igralec igra proti računalniškemu programu. Igralec (računalnik) pobira palice iz vrstic (A,B,C,D). V eni potezi lahko vzame iz ene vrstice eno ali več palic. Zgubi tisti, ki mora pobrati zadnjo palico.

Uporabljeni računalniški program ni neoremagljiv. Kako bi morali spremeniti algoritem, da bi lahko igralec premagal računalnik le v primeru, če bi igro začel ?

```
Igralec izbere vrstico in vzame iz nje eno ali vec
palic. Kdor vzame zadnjo palico, zgubi.
A = /
B = ///
C = /////
D = ///////
Hoces prvo potezo? (D/N) = N
Racunalnikova poteza...
A = /
B = ///
C = /////
D = //////
Tvoja poteza...
Vrstica? = A
Stevilo palic? = 1
A =
B = ///
C = /////
D = //////
Racunalnikova poteza...
A =
B = ///
C = /////
D = /////
Tvoja poteza...
Vrstica? = D
Stevilo palic? = 5
```

```
A =
B = ///
C = /////
D =
Racunalnikova poteza...
A =
B = ///
C = ////
D =
Tvoja poteza...
Vrstica? = A
Stevilo palic? = 1
Pomota...Ponovi vnos...
Tvoja poteza...
Vrstica? = C
Stevilo palic? = 2
A =
B = ///
C = //
D =
Racunalnikova poteza...
A =
B = //
C = //
D =
Tvoja poteza...
Vrstica? = B
Stevilo palic? = 1
A =
B = /
C = //
D =
Racunalnikova poteza...
A =
B = /
C =
D =
Zgubil si...
Ali bi rad se igral? (D/N) = D

A = /
B = ///
C = /////
D = ///////
```

Lista 1. Izvajanje prevedenega programa (levo in zgoraj). Igro lahko začne igralec ali računalnik. Vprašanje: kakšen je program, s katerim vselej zmaga računalnik, če je z igro začel?

```
                                        end;
        state; {display state}
        while d=0 do begin if c=1 then player
                                  else computer;
                           state
                  end;
        if c=1 then begin write('Zgubil si...');writeln end
               else begin write('Zmagal si...');writeln end;
        write('Ali bi rad se igral? (D/N) = ');
        read(ch1); writeln;
        if ch1='D' then main
end; {main}


begin   writeln;
        write('Igralec izbere vrstico in vzame iz nje eno ali vec');
        writeln;
        write('palic. Kdor vzame zadnjo palico, zgubi.');
        writeln;
        main        {***** body of marienbad *****}
end.
```

Lista. Nadaljevanje programske liste iz naslednje strani (desno)

```
program marienbad;

                {programmed by B.Blatnik}
                {October 1982}


var     a:      {number of nonempty rows}
        i:      {row index}
        n:      {number of sticks to be taken away}
        c:      {indicator player/computer}
        d:      {indicator end of game}
        e:      {number of rows containing one stick}
        f       {number of rows containing two sticks}
                :integer;
        b:array[1..4]of integer;
        ch1,ch2         :char;


procedure player;
begin   write('Tvoja poteza...');writeln;
        write('Vrstica? = ');read(ch2);
        write('Stevilo palic? = ');read(n);
        case ch2 of
        'A': i:=1;
        'B': i:=2;
        'C': i:=3;
        'D': i:=4
        end;
        if (b[i]>=n)and(n>0) then b[i]:=b[i]-n
                    else begin write('Pomota...');
                               write('Ponovi vnos...');
                               writeln; player
                         end;
        c:=0
end; {player}


procedure state; {determine & display state of art}
begin   for i:=1 to 4 do
                begin   case i of
                        1: write('A = ');
                        2: write('B = ');
                        3: write('C = ');
                        4: write('D = ')
                        end;
                        case b[i] of
                        0: write(' ');
                        1: write('/');
                        2: write('//');
                        3: write('///');
                        4: write('////');
                        5: write('/////');
                        6: write('//////');
                        7: write('///////')
                        end;
                        writeln
                end; {for}
        d:=0;   a:=0;
        for i:=1 to 4 do if b[i]>0 then a:=a+1;
        if a=1 then for i:=1 to 4 do if b[i]=1 then d:=1
end; {state}


procedure decrement; {decrement number of sticks}
var     m,n     :integer;
begin   m:=1;   n:=b[1];
        for i:=2 to 4 do
                if b[i]>n then begin n:=b[i];
                                     m:=i
                               end;
        b[m]:=b[m]-1
end; {decrement}


procedure computer; {computer's turn}
begin
        write('Racunalnikova poteza...');writeln;
        e:=0;
        f:=0;
        for i:=1 to 4 do if b[i]=1 then e:=e+1;
        for i:=1 to 4 do if b[i]=2 then f:=f+1;
        case a of
        1: for i:=1 to 4 do if b[i]>1 then b[i]:=1;
        2:if e=1 then begin for i:=1 to 4 do
                                if b[i]>1 then b[i]:=0
                      end
                 else if f=1 then begin for i:=1 to 4 do
                                        if b[i]>2 then b[i]:=2
                                  end
                             else decrement;
        3:if e=2 then begin for i:=1 to 4 do
                                if b[i]>1 then b[i]:=1
                      end
                 else if f=2 then begin for i:=1 to 4 do
                                if (b[i]>0)and(b[i]<>2)then b[i]:=0
                                  end
                             else decrement;
        4:if e=3 then begin for i:=1 to 4 do
                                if b[i]>1 then b[i]:=0
                      end
                 else decrement
        end; {case}
        c:=1
end; {computer's turn}


procedure main;
begin   for i:=1 to 4 do b[i]:=2*(i-1)+1; {initialize}
        state;
        write('Hoces prvo potezo? (D/N) = ');
        read(ch1);
        if ch1='N' then computer
                   else if ch1='D' then player
                                   else begin
                                        write('Skoda...Napravil');
                                        write(' si napako...');
                                        writeln;
                                        main  {try again}
```

Lista. Programska lista na tej in prejšnji strani kaže pascalski program igre Marienbad. Prevajalnik za Pascal/Z se izvaja v okviru operacijskega sistema CP/M na mikroračunalnikih Iskra Delta.

NOVICE IN ZANIMIVOSTI

Japonski načrt računalniške dominacije

Desetletne raziskave in razvoj naj bi Japonsko pripeljale na vodilno mesto v računalniški industriji, in sicer s proizvodnjo računalniških sistemov pete generacije. Ta tehnologija naj bi se uveljavila na tržišču v začetku devetdesetih let. Japonci naj bi z njeno proizvodnjo dokončno prevzeli vodstvo na tem področju Združenim državam Amerike. Ideje za nov tehnološki preboj je pripravil Japonski razvojni center za obdelavo podatkov (JIPDEC), ki je izdelal tudi načrte in priporočila za projekte pete sistemske generacije v devetdesetih letih.

Načrt raziskav in razvoja za naslednje desetletje predvideva združitev in upoštevanje vrste inovativnih idej raziskovalcev iz ZDA, Japonske in ostalih dežel, s tem da te ideje razširja in jih vključuje v nov računalniški sistem. Ta cilj bo verjetno uresničljiv tudi zato, ker gre za japonski nacionalni projekt, za katerim se skriva nacionalna računalniška politika, ki bo materialno podprta s strani vlade, kar ni običajno za druge dežele.

MEHANIZEM ZA REŠEVANJE PROBLEMOV

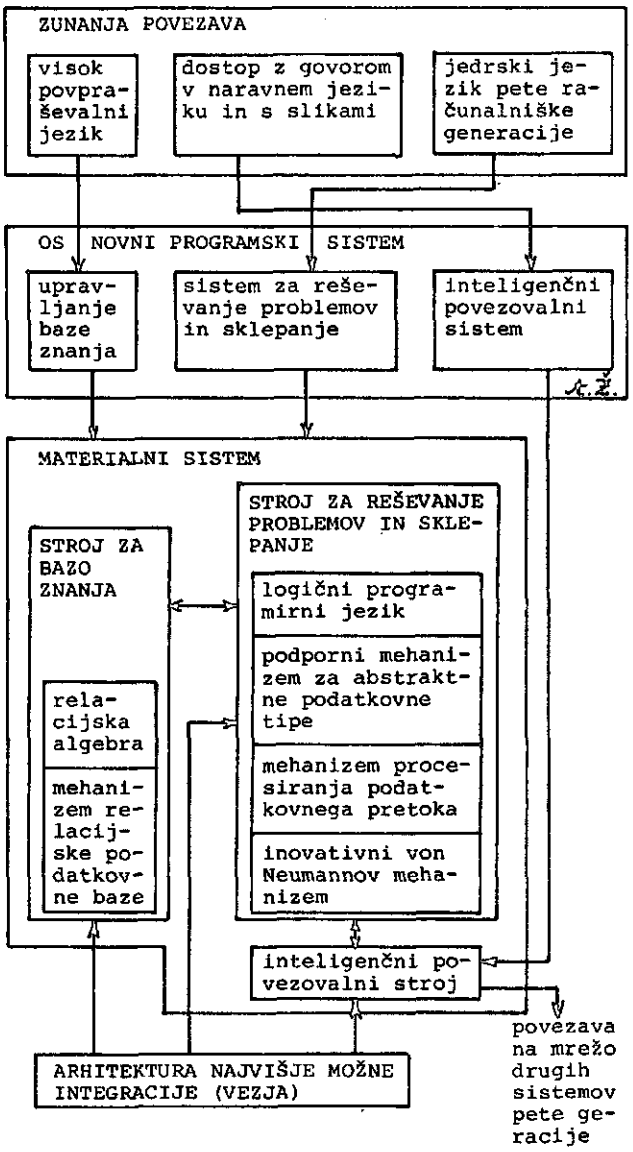
Računalnik prihodnosti bo pokril potrebe uporabnikov ne samo s povečano zmogljivostjo in nižjo ceno, marveč z možnostjo reševanja vrste splošnih problemov; tega današnji računalniki ne zmorejo. Razen tega bo nov sistem naraven za uporabnike, saj se bodo ti z njim lahko pogovarjali v naravnem jeziku pa tudi s pomočjo slik. Sistem, ki bo opravljal to vmesno funkcijo, se imenuje inteligenčni vmesni stroj (glej sliko).

Druga od treh osnovnih funkcij bo lastnost sistema za učenje, pridruževanje in sklepanje, podobno kot to dela človek. Računalnik bo sposoben razčiščevati zapletene zahteve in z uporabo svojega ogromnega informacijskega pomnilnika in pomnilnikov drugih računalnikov bo lahko izvajal sklepe in presoje, ki bodo znatno presegali obseg razmišljanja njegovega človeškega gospodarja. Računalnik bo sposoben voditi inteligentno razpravo z vprašanji in odgovori s poljubno osebo, ki bo imela poljubne interese. Osnovna funkcija, ki bo imela to lastnost, se imenuje sistem za reševanje problemov in sklepanje in ta sistem bo imel za opravljanje te svoje funkcije na voljo poseben stroj (glej sliko).

Tretja osnovna lastnost bo sposobnost uporabe shranjene informacije. Računalnik bo enostavno razumel vsebino podatkovne baze; v tem se bo razlikoval od sistemov, ki podatke le shranjujejo, razpoznavajo in prenašajo. To bodo baze znanja za razliko od današnjih baz, ki so le podatkovne baze in na bazah znanja bo osnovana funkcija reševanja problemov. Ta del trodelnih možganov se bo imenoval upravljavski sistem baze znanja in tudi ta del bo imel svoj poseben stroj (glej sliko) z visoko stopnjo integracije (integrirana vezja).

RAZLIČNE VELIKOSTI RAČUNALNIKOV

Računalniki pete generacije bodo imeli vse možne velikosti, od osebnih računalnikov do super računalnikov. Ti računalniki bodo praviloma povezani v lokalne in globalne mreže. Nekater nove metode bodo npr. nova arhitektura strojev za podatkovni pretok, zamisli umetne in naravne inteligence, novi jeziki in za njih optimizirani stroji (npr. jeziki podobni jeziku LISP, PROLOG, ADA itd.). JIPDEC je priporočil 26 raziskovalnih/razvojnih tem, od katerih ima vsaka po več projektov. Teme so združene v sedem kategorij, kot je prikazano v naslednji tabeli. Vsaka tema ima ciljne specifikacije.



Slika. NAČRTOVANI SISTEM: Inteligenčni vmesni stroj bo omogočal dostop v naravnem jeziku, imel bo možnost učenja in sklepanja ter razumevanja shranjenih podatkov

Npr. osebna delovna postaja bo lahko opravila 2 milijona ukazov v sekundi, imela bo pomnilnik za 0,5M do 5M zlogov in 100M-zložni disk s povprečnim časom dostopa 1ms. Druge specifikacije vsebujejo tkim. superhitri procesor, ki bo izvršil 1 milijardo do 100 milijard operacij s pomično vejico v sekundi in bo imel hitri pomnilnik obsega 8 do 160M zlogov.

PREGLED PROJEKTOV PETE GENERACIJE

Sistemi osnovnih aplikacij

* Sistem za strojno prevajanje
* Sistem za odgovarjanje na vprašanja
* Sistem za uporabno razumevanje govora
* Sistem za uporabno razumevanje slik in podob
* Sistem za uporabno reševanje problemov

Tehnologija za podporo razvoja

* Sistem za podporo razvoja

Sistematizacijska tehnologija

* Sistem za inteligentno programiranje
* Sistem za načrtovanje z bazo znanja
* Sistematizacijska tehnologija za računalniško arhitekturo
* Sistem podatkovne baze in porazdeljene podatkovne baze

Arhitektura porazdeljenih funkcij

* Mrežna arhitektura
* Stroj podatkovne baze
* Stroj za numerične izračune z visoko hitrostjo
* Sistem za komunikacijo človek-stroj na visoki ravnini

Nova vnaprejšnja arhitektura

* Stroj za logično programiranje
* Funkcionalni stroj
* Stroj za relacijsko algebro
* Stroj za podporo abstraktnih podatkovnih tipov
* Stroj za podatkovni pretok
* Inovativni von Neumannov stroj

Tehnologija zelo visoke integracije

* VLSI arhitektura
* Računalniško podprti načrtovalni sistem za inteligentno VSLI

Sistemi osnovne programske opreme

* Upravljavski sistem, osnovan na znanju
* Sistem za reševanje problemov in sklepanje
* Sistem za inteligenčno povezavo

VELIKE ZMOGLJIVOSTI

Načrtovalci predvidevajo, da bo mogoče oblikovati takšno funkcijo reševanja problemov in sklepanja, katere zmogljivost bo 100 milijonov do 1 milijarde logičnih/sklepnih operacij v sekundi (en logični sklep vsebuje od 100 do 1000 ukazov). Drugi primer je specifikacija sistema za procesiranje naravnega jezika. Razen tega naj bi funkcija za upravljanje baze znanja razpoznavala enoto znanja v nekaj sekundah uporabljajoč bazo znanja obsega 100 do 1000 giga zlogov.

Za te funkcije se bodo uporabljala nova integrirana vezja z milijon do 10 milijoni tranzistorji v enem samem vezju. Zato bo razvit sistem za avtomatično načrtovanje takih integriranih vezij.

Zaenkrat se v novi tehnologiji ne bodo uporabljali galijskoarsenidni in Josephsonovi spoji, saj raziskovalci predvidevajo, da ta tehnologija ne bo uporabna pred letom 1990. Vendar bodo budno spremljali razvoj in bodo te spoje uvrstili v novo tehnologijo, če bo doseženo stanje njihove praktične uporabnosti in superiorne zmogljivoati.

Sistem za avtomatično načrtovanje vezij bo sestavljen iz treh delov: iz programske opreme za avtomatično načrtovanje VLSI, iz računalniškega sistema, ki ga bo ta programska oprema poganjala (tkim. System 5G) ter iz 5G osebnega računalnika, ki bo logična programirna delovna postaja za načrtovalce. V prvi fazi (začetnih 5 let) projekta bo implementiran jezik za hierarhično specifikacijo, ki se uporablja v laboratorijih podjetja Musashino EC grupacije Nippon T&T Public Corp.

Sistem z jezikom hierarhične specifikacije ima več modulov, ki si združeni v totalni načrtovalni sistem. Vsebuje jezik za opis vezij, prevajalnik, bazo podatkov, simulator časovnih oblik, logični simulator, simulator vezij, generator preizkuševalnih vzorcev, program za razmestitev elementov in njihovo povezavo in preizkuševalnik pravil načrtovanja.

LOGIČNO NAČRTOVANJE

Del projekta za avtomatično načrtovanje je razvoj delovne postaje za logično programiranje. Noben obstoječi osebni računalnik ne izpolnjuje zahtev za hitro procesiranje govora, grafike, digitaliziranega vhoda podob in zmogljivosti osebnega sklepanja strojev, ki uporabljajo jezika LISP in PROLOG. Eden izmed najhitrejših navadnih univerzalnih računalnikov, ki zmore 40 milijonov ukazov v sekundi, je predviden kot gostiteljski računalnik sistema za avtomatizacijo načrtovanja, dokler ne bo dobavljiv nov sistem pete računalniške generacije.

A. P. Železnikar

****
Računalniška simulacija kriptografije
****

Računalniška simulacija klasičnih substitucijskih kriptografskih sistemov avtorja Rudolph F. Lauerja je zbrana v BASIC programih, ki omogočajo uporabniku simulacijo vsakega klasičnega substitucijskega šifrirnega sistema. Opisana je vrsta kriptoanalitičnih programov, ki omogočajo reševanje problemov (lomljenje šifre). Nadalje je dana informacija in seznam o ameriških kriptografskih patentih, predstavljena je kriptoanaliza in informacijsko ozadje različnih sistemov. Obravnavajo se tipični kriptografski problemi in njihovo reševanje. Naslov knjige je Computer Simulation of Classical Substitution Cryptographic Systems, cena broširane izdaje je $ 24,80 in naslov izdajatelja: Aegean Park Press, P.O.Box 2837, Laguna Hills, Ca 92653.

A. P. Železnikar

****
Miniaturni mikroprocesorski kristali
****

Podjetje Statek izdeluje kvarčne kristale v frekvenčnem intervalu 1MHz do 1,25MHz, ki so 48-krat manjši od industrijskega standarda HC-33. Ti kristali zasedejo na mikroprocesorski plošči le desetino prostora navadnega kvarčnega kristala. Izvedba teh kristalov je prilagojena za namestitev na ploščo s tiskanim vezjem. Podjetje Statek je znano po tem, da izdeluje tudi druge miniaturne kristale v simetričnih in asimetričnih izvedbah, od nizkih do visokih frekvenc, v ohišjih tipa TO-5. Naslov proizvajalca je: Statek Corp., 512 North Main St., Orange, Ca 92668.

A. P. Železnikar

****
Večnamensko upravljanje
****

Večnamensko upravljanje (Management by Multiple Objectives) je procedura in naslov nove knjige avtorja Sang M. Leeja. MBMO je pristop za integracijo upravljavskih sistemov in sodobna upravljavska tehnologija, s katero se doseže več upravljavskih namenov. MBMO obravnava slabosti upravljanja s cilji (nameni): kako obiti medsebojno izključujoče cilje (namene), da bi bili doseženi cilji celotne organizacije. Večnamensko upravljanje omogoča razumevanje konceptov, metod in virov, ki so potrebni pri oblikovanju in izvedbi take vrste upravljanja. Cena knjige je $ 17.50, izdajatelj pa je Van Nostrand Reinhold Co., 135 West 50th St., New York, NY 10020.

A. P. Železnikar

****
Perspektive umetne inteligence
****

Z japonskim projektom pete računalniške generacije se ponovno postavlja vprašanje aktualnosti umetne inteligence kot raziskovalne in razvojne dejavnosti. Uporaba računalnikov ima že danes na voljo vrsto pripomočkov s področja tkim. umetne inteligence. Philips Research Lab (Nizozemska) je dal pred kratkim v uporabo sistem za vpraševanje in odgovarjanje v naravnem jeziku z imenom PHILQA 1. Te vrste sistemi sodijo med najbolj uspešne na področju umetne inteligence (UI). ICLov center za vnaprejšnji razvoj in raziskave v Angliji se je prav tako začel intenzivno ukvarjati z metodami UI na področjih razpoznavanja govora in upravljanja podatkov.

XEROXov raziskovalni center v Palo Alto, ZDA je za več let zamrznil splošne raziskave UI v korist razvoja uporabnejšega pisarniškega sistema. Ta poteza se je izkazala kot zelo koristna, saj ima delovna postaja STAR visoke uporabnostne standarde. Inženirji podjetja DEC uporabljajo svoj UI sistem za pomoč pri konfiguriranju sistemov. Texas Instruments preučuje UI za izboljšavo učinkovitosti programirnega okolja in VSLI načrtovanja. Tudi IBM aktivno ocenjuje z UI povezane sisteme s ciljem obvladovanja različnih nalog, kot sta avtomatično generiranje računovodskih programov in diagnostika napak. Seveda pa IBM ni bil vedno naklonjen tem novim metodam in prejšnji predsednik Thomas J. Watson ml. je črtal UI zaradi možnih obrekovalskih posledic.

UI je bila eno redkih področij, ki je metodično proučevalo povezavo med človekom in strojem. Napredek na področju računalniških sistemov je mogoč na boljši povezavi uporabnika in sistema in seveda na novih računalniških funkcijah. Te funkcije pa so možnost učenja sistema, nove povezave in zanesljivi diagnostični sistemi. Teh novih funkcij pa ni moč doseči z modifikacijo starega: učljivost in uporabljivost je potrebno razvijati z novih položajev, od samega začetka.

Računalniški proizvajalci uporabljajo UI v svojih razvojnih dosežkih in nekateri se tega zavedajo, drugi pa tudi ne. Uporaba UI v novih sistemih je značilna v zadnjih petih letih, prej pa se je na UI gledalo kot na znanost izven realnih meja. V 60-ih letih je bil cilj UI tudi namera oblikovanja umetnega človeškega bitja in taki cilji so bili nerealistični ter so UI prav gotovo škodovali. Leta 1958 sta pionirja UI H. Simon in A. Newell zapisala: Na svetu že obstajajo stroji, ki mislijo, se učijo in so ustvarjalni. Še več: njihove zmogljivosti za opravljanje teh operacij se hitro povečujejo dokler v skorajšnji prihodnosti ne bodo zmogli reševanja problemov, ki so na stopnji reševanja človekovih možganov.

Ta skorajšnja prihodnost je še vedno oddaljena. V zadnjih dvajsetih letih je UI pomagala uresničiti vrsto inteligentnih človekovih dejavnosti na računalnikih, toda se praktično ni približala posnemanju človekovega genialnega razmišljanja. Za dosego tega cilja so potrebne še številne nove raziskave delovanja možganov in informacijskih procesov v njih, potrebno je poglobljeno znanje o nastajanju informacije v možganih, o njihovem paralelnem in diverznem delovanju, potrebni so natančnejši informacijski pojmi inteligence itd.

V Združenem kraljestvu je bilo v letu 1972 pripravljeno poročilo, ki je dobesedno izničilo UI in raziskave robotov v tef deželi. To poročilo je izdelal ugledni matematik Sir James Lighthill na osnovi raziskav UI pod okriljem Odbora za znanstvene raziskave in nekaterih univerz. V tem poročilu je Lighthill razkrinkal UI tako temeljito, da je bilo njeno raziskovanje skupaj z robotiko zaustavljeno do konca desetletja. Vendar je leta 1980 Odbor za znanstvene raziskave spremenil svoje stališče in namenil 5 milijonov dolarjev za raziskave robotike, nekaterih področij UI, zavzel pa se je tudi za ustanovitev posebnega raziskovalnega središča, ki se bo imenoval Turing Institute.

V 70-ih letih so raziskovalci UI znižali svoje teoretične poglede in povzdignili praktične elemente svojega dela. Niso več poskušali oblikovati univerzalni stroj za razmišljanje, ki bi bil podoben človeku, temveč so se lotili specifičnih nalog, ki naj bi človeku pomagale oblikovati in uporabljati računalnike. Ta sprememba je bila podprta z zahtevo uporabnikov in raziskovalcev, da naj se olajša uporaba računalnikov. Dotlej je bilo le malo raziskav za rešitev problema, ki so ga imenovali "uporabniška prijaznost." Tudi raziskovalci človekovega obnašanja, kot so psihologi, so namenjali le malo pozornosti uporabi računalnikov. Celo dobro organizirane dejavnosti s področja ergonomike in tehnike človeškega faktorja niso uvidele pomembnosti dela z računalniki.

Računalniški praktiki so bili v tem razdobju zaposleni s tehnologijo in posebno uporabo računalnikov. Zaradi tega se je umetna inteligenca (UI) razvijala kot posebna veja kibernetike, ki ni bila povezana z računalniško znanostjo. Kibernetika pa se je ukvarjala s krmilnimi mehanizmi, ki omogočajo biološkim, organizacijskim in umetnim sistemom, da delujejo uspešno. Uporaba računalnikov v raziskavah

UI je končno oddaljila UI od kibernetike, ki se je še vedno ukvarjala s preveč splošnimi sistemi. Vendar so z UI prodrle v računalniško znanost nekatere kibernetične, interdisciplinarne metode.

Razvoj komuniciranja v naravnih jezikih z računalniki kaže še eno vrsto uporabe UI. V 60-ih letih je bil poudarek raziskav na programirnih jezikih. V 70-ih letih je dominiral razvoj definicije podatkovne baze, jezikov za bazni dostop in metod za razpoznavanje podatkov; vsi ti napori so bili povezani z računalniško usmerjenim pogledom na uporabniško povezavo. UI si je prizadevala zgraditi komunikacijo z računalniki v naravnem jeziku. Tako je ameriški jezikoslovec Noam Chomsky preučeval možnost prevajalnika za angleški jezik. Vendar je Chomsky sklepal, da je učenje in razumevanje naravnega jezika preveč kompleksno za obravnavo na avtomatičnem sistemu in da mora temeljiti na podobnih lastnostih, ki so značilne za človekove možgane.

Večina raziskovalcev UI je priporočala opustitev nadaljnih raziskav v smeri razumevanja naravnega jezika z računalniki. Zato pa je bila večja pozornost usmerjena v kontekstno odvisne naravne jezike za komunikacijo z računalniki. V zgodnjih 70-ih letih je nastala vrsta interaktivnih dialognih sistemov v ZDA. Pri tem sta se pojavili dve glavni razvojni smeri: prva je temeljila na pojmu "skriptnega dialoga," druga pa na razpoznavanju informacije iz baz podatkov. Obe metodi sta spočetka temeljili na enostavnih, trivialnih poskusih, pozneje pa sta se razcveteli v visoko razvite komercialne sisteme.

V tkim. skriptnih sistemih je vnaprej definirana vrsta vprašanj in odgovorov, ki se lahko pojavijo na različnih stopnjah dobro strukturiranega dialoga. Npr. zdravnik-bolnik ali strežnik-porabnik sta tipični skriptni situaciji. Taki sistemi se uporabljajo le v okviru zelo specializiranih kontekstov. Povpraševalni sistemi podatkovnih baz pa uporabljajo UI pri posplošenem sistemu upravljanja baz podatkov. Sistemi za odgovarjanje na vprašanja, kot je Philipsov PHILQA 1 in IBMov QBE (Query By Example), so tako ali drugače povezani z metodami UI na področju naravnih jezikov.

V poznih 70-ih letih je postalo jasno, da je baza podatkov osrednji problem, oziroma da je baza znanja tisto, kar je treba temeljito raziskati. Program lahko razume naravni jezik z uporabo znanja "sveta", ia katerega prihaja informacija. Skriptno dialogni sistem je relativno tog za opisovanje modela sveta. Potrebne so bolj prožne metode za graditev in uporabo baze znanja. Baze znanja hranijo prečiščeno bistvo (jedro) človekovega strokovnega znanja v obliki pravil znanja, pri čemer so ta pravila strukturirana v vzorce človekovega sklepanja. Takšni sistemi z bazami znanja so znani kot izvedenski (ekspertni) sistemi. Takšen izvedenski sistem je npr. DECov sistem R1, ki vsebuje znanje tehniških izvedencev. Tudi IBM izdeluje izvedenski sistem DART na stanfordski univerzi. Cilj sistema DART je zajetje znanja za posebno oblikovanja in diagnostiko tistih izvedencev, ki se ukvarjajo z oblikovanjem sistemov. Drugi ekspertni sistemi sistemi se gradijo za medicinsko diagnostiko, strukturalno analizo in geološke raziskave. Za področje računalništva bi bil izredno zanimiv ekspertni sistem, ki bi zajel znanje s področja programiranja. R. Quinlan z univerze v Sydneyu je uporabil izvedenski sistem za pisanje programov, ki rešujejo posebne šahovske situacije. Ta program je bolj učinkovit kot program, ki bi ga napisal avtor. Seveda se bodo v tem desetletju pojavili avtomatični generatorji programov, polavtomatični generatorji pa se prodajajo že danes.

Polavtomatično programski generatorji ne opravijo sami vsega dela ter so interaktivno povezani s programerjem pri izdelavi programa. Ti programi postavljajo vprašanja uporabniku, večkrat pa tudi zahtevajo, da se določeni programski segmenti napišejo ročno. IBM raziskuje te metode s ciljem, da bi omogočil programiranje nestrokovnjakom pri oblikovanju računovodskih in poslovodskih programov, ki se prilegajo konkretnemu okolju. V zadnjih dveh letih se je pojavilo več polavtomatskih programskih generatorjev, ki generirajo programe v jezikih BASIC, COBOL in tudi v drugih jezikih (PEARL, THE LAST ONE itd.).

Ameriški futuristi verjamejo, da bo UI in še posebej izvedenski sistem uporabljen kot "ljudski" ojačevalnik. Te metode bodo skupaj z novimi visoko integriranimi vezji povzročile različne ljudske ojačevalnike, kot bodo npr. knjiga-v-vezju, učitelj-v-vezju, zdravnik-v-vezju itd. Ti tehnološki dosežki bodo omogočili ljudem, da si priredijo lastne izvedenske sisteme. Skeptiki napovedujejo za ta tehnološki prodor še nadaljnih 10 let.

Neglede na časovne zakasnitve in prehitevanja je več ali manj jasno, da prihodnost pripada ekspertnim sistemom: na tej predpostavki temelji tudi japonski državni projekt pete računalniške generacije. Ti sistemi bodo potrebni za golo preživetje človeštva, kot poudarja Donald Michie iz Enote za strojno inteligenco edinburške univerze. Vse večja odvisnost okolja od računalnikov povzroča nujnost uporabe teh sistemov. Ko bodo ti programirani za človeku podobno sklepanje, bodo po Michiju oblikovali "človekovo okno" v "skrivnostni" stroj. Ekspertni sistemi se bodo lahko uporabljali tudi kot inštruktorji, saj prečistijo strokovno znanje v jasno opredeljena pravila.

V prihodnosti bodo izvedenski sistemi pomagali razrešiti vrsto problemov povezave človek-stroj, ki so trenutno največja ovira za široko uporabo računalnikov. Tako bodo nastali uporabniško prijazni sistemi. Sistemi z bazami znanja bodo zaradi velikega uporabniškega interesa morali imeti nekatere nove, še ne odkrite lastnosti človekove inteligence; to pa je zagotovilo, da bo UI tudi v prihodnjem stoletju imela osrednjo vlogo na računalniškem prizorišču.

A. P. Železnikar

****
Svetovni računalniški center v Franciji
****

Francoska vlada je na pobudo J.-J. Servan-Schreiberja ustanovila tkim. svetovni računalniški center, katerega generalni direktor je postal Nicholas Negroponte, bivši direktor za računalnike in telekomunikacije na MIT. V okvir novega centra je bil postavljen tudi profesor Raj Reddy, bivši direktor instituta za robotiko na Carnegie-Mellon univerzi in Seymour Papert, avtor jezika Logo za poučevanje programiranja otrok. Ker gre za tri vodilne ameriške računalniške strokovnjake, ki so se preselili v Pariz, so v ZDA postavili vprašanje, zakaj podobnega projekta, ki ga je sprožil Servan-Schreiber, niso začeli v ZDA. Za kaj prevzaprav gre?

Vzroki za izselitev vodilnih ameriških strokovnjakov so preprosti. Francoska vlada je uvidela, da je ustanovitev svetovnega računalniškega centra povezana s posredno ali neposredno koristjo vsakega državljana sveta. V ZDA takšne uvidevnosti tipično ni in v Washingtonu že ugo-

tavljajo, da je zaradi odliva vrhunskih kadrov iz ZDA nekaj narobe z državno strategijo na področju računalništva. To pomeni, da obstaja za vrhunske strokovnjake močnejša motivacija izven ZDA. Američani ugotavljajo obstoj določene krize računalništva, ki bi jo bilo nemudoma potrebno identificirati.

Servan-Schreiber je v svojem predavanju v Washingtonu nakazal, kje vidi slabosti ameriškega tehnološkega napredka, ki je izredno občutljiv na računalniškem področju. Tako so npr. Japonci vstopili na tržišče motorjev na reakcijski pogon. Pred tremi leti so sklenili sporazum z British Rolls-Royce v okviru oddelka za reaktivne motorje. V zadnjem času postaja očitno, da prevzemajo japonska podjetja vodilno vlogo na področjih, ki predstavljajo pionirstvo ameriškega tehnološkega duha, spretnosti in podjetnosti. Tako se npr. tekoči kristali proizvajajo danes izključno na Japonskem, nadalje pa velja zadnja ugotovitev za avtomobilsko, kamersko, stereo, ladjedelniško industrijo in še posebej za industrijo osebnih računalnikov.

Svetovni računalniški center je nov, ambiciozen projekt, katerega cilj je predaja računalniške moči v roke ljudstva (tu se očitno v določeni meri zrcali francoska revolucionarna tradicija). Med predlogi novega centra je projekt za instalacijo osebnih računalnikov v 500 naseljih, ki naj bi bila s področja tretjega sveta, nekaj teh računalnikov pa bi postavili tudi na območje razvitih dežel. Servan-Schreiber, ki je trenutno še direktor centra, se je trdo in dolgo boril s francosko vladno birokracijo in končno dobil pristanek za ustanovitev centra v Parizu. Negroponte je prišel z MIT zaradi boljših delovnih pogojev, saj ima v centru na voljo dvakrat več ljudi, kot jih je imel na MIT v 15-letnem projektu.

Američani priznavajo, da je zamisel Svetovnega računalniškega centra prava in kot običajno utemeljujejo to trditev s številkami. Te številke se nanašajo na procente delovne sile, ki je zaposlena na področju informatike v posameznih državah:

Združene države Amerike	41,1 %
Kanada	39,9 %
Francija	32,1 %
Japonska	29,6 %
Venezuela	26,3 %

Zanimivo je, da dežele, kot je Venezuela, ki tehnološko ni preveč napredna, zaposljujejo znaten delež delovne sile na področju informatike.

Servan-Screiber poudarja nekoliko radikalno stališče, ko trdi, da ljudje že v nekaj letih ne bodo več delali na produktnih linijah. To delo bo v celoti prepuščeno robotom in strojem. V tej zvezi poudarja in svari, da se bodo ljudje morali naučiti procesirati informacije (biti v vlogi informacijskih procesorjev). Ta izjava se seveda nanaša na intenzivnost informacijskega procesiranja v možganih, saj bo po predvidevanjih Servan-Schreiberja prav ta intenzivnost znatno povečana.

Tudi roboti postajajo vse bolj zanimivi, saj postaja njihova uporaba cenejša, kot je človekovo delo. Cena robotove ure znaša $ 4,80, za živo delovno silo pa veljajo tele urne cene:

Združene države Amerike	$ 9,09
Zvezna republika Nemčija	$ 11,33
Japonska	$ 5,58
r o b o t i	$ 4,80

Iz tega izhaja, da ljudje morajo postati "računalniško pismeni," da bi v prihodnjem svetu lahko preživeli. Američani se jezijo, ker njihova vlada ne uvidi tega bistvenega pogoja preživetja.

Odgovor na omenjeno dilemo bi bili vladni in drugi projekti, ki bi vzpodbujali programe za računalniško pismenost. To pa pomeni, da se morajo osebni računalniki pojaviti v domovih in šolah. Pred ameriškim kongresom sta dva predloga za finančno podporo proizvajalcem, poslovnežem in šolam za povečanje naporov, s katerimi bi se uresničevali cilji računalniške pismenosti. Prvi predlog (sprožilo ga je podjetje Apple) je tale: v razdobju enega leta naj bi računalniški proizvajalci dali računalnike v šole s znatnim popustom. Toda eni šoli lahko da proizvajalec le en sistem, da s tem šole ne bi bile zasute z računalniškimi sistemi. Ti sistemi morajo biti najsodobnejši (ne starejši od dveh let).

Drugi predlog predvideva 100 dolarski letni kredit na člana družine do 50 % cene domačega računalniškega sistema, s petletno odplačilno dobo. Na ta način bi si družina 4 članov lahko nabavila sistem z vrednostjo $ 4000 pri kreditu $ 2000. Cilj teh predlogov je, da bi bil računalnik dosegljiv vsakemu Američanu.

A. P. Železnikar

"IBM/370 na enem integriranem vezju?"

To je geslo poročila, ki ga bo podal povabljeni strokovnjak H. Painke iz IBM laboratorijev v ZRN, na osmem simpoziju Evropskega združenja za mikroprocesorje in mikroprogramiranje (EUROMICRO), ki bo od 5. do 9. septembra v Haifi v Izraelu.

Vsakoletni simpozij EUROMICRO je vodilno evropsko srečanje strokovnjakov z univerz in iz industrije s področja mikroprocesorskih sistemov in mikroprogramiranja. Udeležijo se ga poročevalci in obiskovalci iz skoraj vseh evropskih dežel, ZDA in daljnega vzhoda.

Drugi povabljeni strokovnjak je R. Zaks, predsednik in ustanovitelj SYBEX-a. Njegovo poročilo ima naslov "Mikroračunalniška industrija: preteklih deset let, naslednjih pet let".

Po izkušnjah iz prejšnjih simpozijev je tudi letos organiziran dan seminarjev, poslušalci pa bodo lahko izbirali med dvema temama: " Firmware (vsebina pomnilnika ROM, ki določa lastnosti vezja), podpora za programsko opremo" od G. Chrousta iz IBM, Dunaj, Avstrija in "Distribuirani mikroprocesorski sistemi in lokalne mreže", H. Saal iz Nestar systems inc., Palo Alto, ZDA.

V znanstvenem programu EUROMICRO 82 bodo podani referati, ki so jih recenzirali trije vodilni strokovnjaki s posameznih področij. Te referate je programski odbor izbral iz velikega števila prispevkov. Podana bodo tudi kratka sporočila o opravljenem delu, na področju programske in aparaturne opreme.

Razstava proizvajalcev bo ob simpoziju, predstavljeni bodo mikroprocesorski sistemi, razvojni pripomočki, testni sistemi in periferne naprave. Razstavljalci bodo svoja poročila podali v okviru posebnega industrijskega programa.

Na EUROMICRO 82 bo finalno tekmovanje natečaja "EUROMOUSE o labirintu". Robot - miška z mikrokontrolo, se giblje v labirintu omejeni čas. Miška ima deset minut časa za raziskavo labirinta in učenje, nato pa mora v čim krajšem času najti izhod iz laborinta.

Neda Papić

OKROGLA MIZA

„INFORMACIJSKI SISTEMI ZA PRENOS ZNANJA"

LJUBLJANA, 12.5.1982 od 16.00–19.15

Zapisali: FRANC ŽERDIN, BORUT JUSTIN, SILVA PREDALIČ, BRANKA FRAS
Uredil: BORUT JUSTIN

Ob priliki simpozija in razstave INFORMATIKA '82 (10.14.5.1982) je bila organizirana okrogla miza, anmenjena izmenjavi mnenj o računalniško in mikrofilmsko podprtih informacijskih sistemih za iskanje znanstvenih, tehničnih in strokovnih informacij.

Okroglo mizo je vodil dr. Borut Justin, direktor Informacijskega centra, v sodelovanju s prof. dr. Aleksandro Kornhauser, predstojnico Specializiranega INDOK centra za kemijo (KEMINDOK) in mag. Valentinom Jarcem, vodjem specializiranega INDOK centra za strojništvo (SICS).

Na okrogli mizi je sodelovalo 60 udeležencev iz gospodarstva, raziskovalne, izobraževalne, upravne in informacijske dejavnosti, v razpravi pa je poleg voditeljev okrogle mize sodelovalo deset razpravljalcev (nekateri od njih po večkrat):
B. Grabnar, J. Španring, T. Banovec, M. Stele, L. Kristan, M. Šlajpah, J. Mogilnicki, V. Ambrožič, J. Kranjc, G. Galunič.

Predvsem dva razloga obstajata, da so se pred približno 20 leti začeli hitro razvijati in uveljavljati računalniško in mikrofilmsko podprti informacijski sistemi za iskanje in pretok znanstvenih tehničnih in strokovnih informacij in podatkov:

- prava eksplozija in poplava informacij kot posledica vse hitrejšega razvoja
- razvoj računalnikov in njihove uporabe ter mikrofilmskih sistemov, oboji pa so se izkazali kot izredno primerno orodje za uvajanje povsem novih prijemov, tehnologij in načinov dela

Zaradi prvega razloga, pa tudi zaradi čedalje večjega pritiska na produktivnost razsikovalnega, konstrukcijskega in podobnega intelektualnega ustvarjanja, predvsem v smislu potrebe za zmanjševanje porabe časa za informiranje po klasičnem načinu v strokovnih knjižnicah, se je pojavila akutna potreba po INDOK dejavnosti, kot jo danes pojmujemo: razbremeniti uporabnika informacij mučnega in zamudnega iskanja, ter vzpostaviti sisteme, ki mu bodo omogočili z veliko selektivnostjo, hitrostjo in točnostjo seznanjanje z novostmi in iskanje informacij na njegovem področju, ob pomoči specializiranih strokovnjakov in informacijskih služb.

Kot je v takme in podobnih primerih običajno so se najprej pojavili zametki informacijskih služb, procesov in sistemov znotraj velikih organizacij ali projektov (na primer: NASA), ki brez tovrstnih uslug niso mogli več normalno delovati. Kmalu se je pokazalo, da zaradi zelo hitro rastočega števila informacij potrebujejo tovrstni informacijski sistemi sorazmerno velike računalniške sisteme, zlasti na področju eksternega spomina (magnetni diski ali podobno), pokazala se je pa tudi kmalu omejena vrednost paketnih računalniških obdelav in nastopila potreba po modernejšem in zahtevnejšem načinu direktnega dialoga z računalnikom v realnem času - online delo. Ta zahteva je seveda dvignila celotno zahtevo na tovrstnih projektih še na dveh nivojih: potrebna je bila izdelava specializirane programske opreme in izgradnja telekomunikacijskih

možnosti za daljinski dostop do baz podatkov. Paralelno s tem se je spremenila tudi filozofija dosedanjega sekvenčnega pregledovanja baz podatkovv indekssekvenčno oziroma invertirano, za kar je bilo seveda potrebno razviti dodatne pripomočke.

Ob takih razmišljanjih je bil okoli leta 1965 v sodelovanju ameriške firme Lockheed in NASA razvit prvi online informacijski software pod imenom RECON, ki je omogočal online iskanje informacij. Približno istočasno pa se je po severno-ameriškem prostoru začelo širiti prvo javno omrežje za penos podatkov (ARPANET), kar je bil drugi pogoj, da je Lockheedov in formacijski sistem pod iemnom DIALOG začel prodirati v javno uporabo, ter v kratkem času postal komercialno dostopen - za uporabnike na severno - ameriškem področju.

Presenetljiva uspesnost tega projekta je kmalu rodila tudi konkurenčne proizvode, pa tudi sila hiter in uspešen nadaljnji razvoj DIALOG sistema, ki tudi danes še ni končan. Danes je mogoče v svetu najti tri kategorije programskih paketov za online uporabo znanstvenih, tehničnih in strokovnih informacij (Information Retrieval Software):

1. Specializirani programski paketi za komercialne operacije v velikem obsegu (praviloma mednarodne informacijske sisteme), kot na primer: DIALOG, ORBIT (oba USA), BRS - DATASTAR (USA - Švica), ESA - QUEST (evropska prostorska agencija, Frascati, Italija - nadaljni razvoj NASA - RECON softwarea)

2. Programski paketi proizvajalcev računalniške opreme, prirejen tudi - a ne samo - potrebam iskanja znanstvenih, tehničnih in strokovnih informacij online, kot na primer: IBM - Stairs, UNIVAC - Unidas ali SIEMENS - Golem.

3. Neodvisni programski paketi posameznih velikih družb, ki navadno poleg information retrievel funkcije zadostujejo tudi potrebam izmenjave informacij znotraj družb - na primer telekonferiranje. Primer: ICI - Assassin.

Seveda se vse te tri kategorije niso odvijale neodvisno ena od druge, saj je na primer BRS - DATASTAR spftware izvedenka IBMovega Stairsa.

Približno od leta 1970 dalje je bilo tudi pri nas precej naporov vloženih v razvoj informacijskega softwarea, ki bi ga pa le težko postavili ob bok zgoraj navedenim primerom, saj je v začetku bil pretežno paketno orientiran, danes sicer že omogoča določeno online delo, vendar na bistveno nižjem nivoju kot zlasti prva kategorija navedenih sistemov, kar pa je tudi povsem razumljivo in v celoti niti ni mogoče slabše funkcionalnosti pripisati zgolj softwareu, temveč tudi (ne)razpoložljivosti zadostnih strojnih računalniških kapacitet. Pri nas doma je prvi paket razvila ISKRA: SAIDC, nato Republiški računski center - DORS in kasneje še Računalniški center Univerze v Ljubljani - IBIS.

Ceni se, da dober informacijski software (Information Retrieval Software) zahteva okoli 60 človek - let(man years) dela zelo dobro kvalificiranega, specializiranega in kvalitetnega kadra, kupiti ga navadno ni mogoče, saj je vedno skoraj izključno razvit za potrebe lastnih komercialnih operacij v velikem obsegu. Upoštevajoč taka dejstva ob trenutnem stanju naše informacijske dejavnosti ter njenih perspektiv ne bi bilo realno pričakovati doseganja nivoja mednarodnih inforamcijskih sistemov v našem okolju.

Poleg obsežnosti in zahtevnosti programske opreme pa je potrebno pri zagotavljanju takega informacijskega sistema upoštevati tudi obsežnost in s tem ceno potrebne strojne opreme in telekomunikacijskih možnosti za uporabo sistema. Trenutni status DIALOG informacijskega sistema (začetkom 1982) je naslednji: stalno je online na razpolago približno 130 baz podatkov s skupno cca 60 milijoni pretežno bibliografskih referenc. Ob oceni, da ima povprečna referenca (bibliografski zapis) 800 znakov in ob upoštevanju potrebnega prostora na tiskovnih pogonih za indekse in kazalce, se da oceniti, da je potrebno za normalno delovanje isstema najmanj 100 verjetno pa več velikih diskovnih pogonov. Trenutno ima DIALOG informacijski sistem 20.000 upo-

rabnikov po celem svetu, tako, da je potrebno za zagotovitev normalne operativnosti omogočiti hkraten dostop (vsaj v konicah) več tisoč terminalov. Nadaljna ostra zahteva je postavljena na zanesljivost sistema, kar je v bistvu mogoče doseči le z dupliciranjem najpomembnejših funkcij in elementov, zato je normalno, da so veliki informacijski sistemi grajeni na osnovi dupleks računalniških sistemov (dvojni procesorji), od katerih je v sili vsak sposoben prevzeti celotno funkcijo - seveda z ustreznim podaljšanjem odzivnega časa.

Rezultat takih kalkulacij je nedvoumen sklep, da je smiselno graditi (pa tudi uporabljati) velike, v principu mednarodne informacijske sisteme, kar pa ne izključuje možnosti distribuirane gradnje posameznih baz podatkov. Razvoj v svetu, zlasti pa v Evropi ne gre povsem v smislu takih razmišljanj, to pa predvsem kot posledica različnih regionalnih in nacionalnih pogledov, značilnosti, opredelitev in apetitov, katerih razlogi pa so pretežno politične narave.

Za trenutek so uporabnikom v svetovnem merilu na razpolago trije veliki informacijski sistemi v Združenih državah Amerike, ter okoli 10 v Zahodni Evropi (Velika Britanija 2, Zahodna Nemčija 2, Italija 1, Francija 2 + 2 v gradnji, Švica 1). Številčno stanje teh sistemov daje precej nenormalno podobo ob upoštevanju dejstva, da se trenutno porabi nad 80% celotnih svetovnih informacijskih uslug na Severno-ameriškem področju, dobrih 10% v evropskem prostoru, preostanek pa odpade predvsem na Japonsko, Avstralijo in Novo Zelandijo, čeprav so ti sistemi dosegljivi in se - sicer marginalno - uporabljajo tudi v Afriki, Aziji in Južni Ameriki. Pred dobrim letom dni opravljena tržna analiza in študija do leta 1985 (B. Collier, Learned Information, Oxford, Velika Britanija) ugotavlja, da je v evropskem prostoru mesta za dva informacijska sistema, ali z drugimi besedami, da je trenutno stanje sila neracionalno in ekonomsko neupravičeno.

Glede na realno stanje pri nas, je povsem realno mogoče postaviti trditev, da je prepad med nami in Zahodno Evropo še bistveno večji kot med Zahodno Evropo in Ameriko, zato se sam od sebe vsiljuje zaključek, da je za nas edina alternativa (povsod tam, kjer je to politično in tehnično mogoče zagovarjati) čim direktnejša navezava na mednarodne informacijske sisteme in njihova trajna uporaba, saj so kakršnakoli druga razmišljanja brez vsake stvarne podlage in torej nerealna. Tak zaključek pa seveda s precejšnjo ostrino in takoj postavlja vprašanje, kaj storiti z našimi domačimi podatki, informacijami in bazami podatkov, ki jih delno že gradimo. Povsem nesprejemljivo bi namreč bilo, če bi se jim odpovedali, saj bi s tem potencirano in čedalje hitreje zapadali v vsevečjo in večjo odvisnost od tujine na tem verjetno najbolj perspektivno pomembnem področju bodočnosti. Zaradi tega seveda, iz nacionalnega stališča gledano, pri gradnji domačih baz podatkov in informacijskega sistema za njihovo uporabo ekonomski način razmišljanja ne sme biti merodajen, vsaj v tem trenutku ne. Ocenjujemo, da je brezpogojno potrebno ne le v istem tempu naprej graditi domače baze podatkov kot je to bil primer doslej, temveč te napore še bistveno ojačiti, tako v smislu obsega in tematskega pokrivanja baz podatkov kot njihove kvalitete (abstrakti), paralelno s tem pa je seveda potrebno posodobiti tudi metode uporabe teh baz podatkov, saj moramo stremeti k cilju, da čimbolj zmanjšamo razliko med tehnološkim nivojem uporabe tujih in domačih virov informacij.

Pet delujočih slovenskih INDOK centrov, ki že več let oskrbujejo uporabnike z informacijami iz računalniško ali mikrofilmsko podprtih informacijskih sistemov, se je v preteklem letu dogovorilo skupno pripraviti in kasneje realizirati projekt posodobitve informacijskega sistema za znanstvene tehnične in strokovne informacije. Sredstva za projekt bodo centri delno združili sami, seveda pa pričakujejo pomoč od pristojnih in odgovornih organov in organizacij v republiki, predvsem Raziskovalne skupnosti, Gospodarske zbornice, Izvršnega sveta, Univerze in proizvajalcev računalniške opreme. Glede na to, da je pretežni del današnjih uporabnikov informacijskih sistemovomenjenih petih centrov iz gospodarstva (okoli 90% vseh uslug, ki jih centri nudijo) in ob pričakovanju, da bo tak trend več ali manj nespremenjen tudi v bodoče (saj je v tujini tako), je prevzela vodilno vlogo pri organizacijskih pripravah projekta Gospodarska zbornica Slovenije. cilji projekta so zastavljeni v skladu z zgoraj navedenimi razmišljanji, trajal naj bi dve leti, program pa je sestavljen modularno, z namenom, da se delni rezultati čimpreje prenesejo v vsakodnevno prakso.

Rezultati projekta naj bi omogočili domačim uporabnikom online uporabljati domače baze podatkov, njegov dejanski uspeh pa je v precejšnji meri odvisen od nekaterih spremljajočih, predvsem infrastrukturnih faktorjev, ki niso sestavni del projekta, kot na primer:

- vzpostavitev javnega omrežja za prenos podatkov - vsaj v Sloveniji, če ne na področju celotne SFRJ, najkasneje do leta 1985
- razpoložljivost primerne računalniške opreme, na kateri bi bilo mogoče permanentno instalirati baze podatkov za online delo in, ki bi imela ustrezne telekomunikacijske možnosti za daljinsko online delo
- razpoložljivost (na tržišču) primernih terminalov, ki bi jih uporabniki za tako delo lahko kupili in uporabljali
- izboljšana knjižnična mreža, primerno oskrbljena z revijami, knjigami in drugim primarnim materialom, iz katere je v zadnji fazi (online) mogoče dovolj hitro in učinkovito pokrivati potrebe po naročilih kopij dokumentov. To organizacijsko že obstoja, problematična pa je iz znanih razlogov (restrikcija uvoza) preskrbljenost z dokumenti.

Trenutni problemi tovrstne informacijske dejavnosti pri nas so predvsem naslednji:

- izredno nizek nivo uporabe
- pomanjkljivo šoljanje uporabnikov, oziroma pomanjkanje kakršnekoli splošne izobrazbe v tej smeri
- slabo rešeno financiranje dejavnosti v celoti, kar nalaga redkim osveščenim uporabnikom, ki usluge izkoriščajo nesorazmerna finančna bremena
- slaba opremljenost informacijskih centrov in uporabnikov
- stalni in kroničnl problemi pri uvozu informacijskih uslug in materiala (plačevanje računov za baze podatkov, za online informacijske usluge in za dokumentacijo)
- slaba koordinacija in premalo sodelovanja med posameznimi INDOK centri
- slaba koordinacija in povezava struktur odgovornih za sfero in odsotnost kakršnekoli konkretnejše politike in ciljev na tem področju.

V resnici v zavest vodstvenih in strokovnih ljudi pri nas še ni prodrla resnica o neusmiljeni potrebi po sistematični gradnji in uporabi modernih informacijskih sistemov v celoti, sprotne gospodarske težave pa vsakodnevno ponovno odvračajo pozornost od te, za bodočnost ključne problematike. Zaradi praktično popolne odsotnosti tega problema v izobraževalnih programih povzroča zadrege neznanje na tem področju in sila meglene ter ter neizdelane predstave o možnostih, trenutnem stanju v svetu in perspektivi informacijskih sistemov. Mistifikacija računalništva je še vedno prisotna, nepoznavanje procesa in sistema pa uporabnikom v bistvu onemogoča ali vsaj otežuje primerno uporabo že danes razpoložljivih uslug.

Da se to stanje čimhitreje in čimmanj boleče preseže, je potrebno razmišljati o nekoncencionalnih prijemih, nekonvencionalnih predvsem za knjižnično in INDOK dejavnost. Pasivno nudenje uslug ne zadošča, potrebno je aktivno sodelovanje s potencialnim ali rednim uporabnikom, predvsem aktivno sodelovanje v smislu reševanja konkretnih uporabniških problemov, to pa implicitno vsebuje tudi sodelovanje uporabnika pri uporabi informacijskih sistemov in procesov, oziroma njegovo sprotno izobraževanje, ter seveda prilagajanje procesa dela v INDOK centrih konkretnim, včasih na žalost tudi kontradiktornim zahtevam uporabnikov. Profesionalci na področju informacijske dejavnosti moramo bolj upoštevati dejastvo, da je čas (vsaj v primeru gospodarstva) za uporabnike dobesedno denar, zato je potrebno informacijsko uslugo oplemenititi tudi v tej smeri, kar seveda pomeni več konkretnega intelektualnega sodelovanja ob konkretnih problemih in na konkretnem mestu (pri uporabniku).

V približno dve uri trajajoči razpravi je bilo iznešeno precej misli, sugestij in ugotovitev:

V zadnjih letih je v splošnem zelo veliko govora o informacijskih sistemih, takih in drugačnih, obstoječih in bodočih, vendar je povprečnemu občanu iz vseh teh razprav čedalje manj jasno kaj želimo in predvsem kako želimo. Bomo imeli en sam vseobsegajoč mamutski informacijski sistem ali tisoče informacijskih sistemčkov, bodo le ti med seboj sploh in kako povezani? V svetu se porajajo nove tehnologije in aplikacije, nekatere od teh izgledajo zlasti zanimive za naš družbeni

sistem, na primer VIDEOTEXT (PRESTEL, BILDSCHIRMTEXT, TELETEL). Videotext tehnologija daje možnosti za splošne informacijske sisteme za javno uporabo, saj je kot aparaturno opremo zanje mogoče uporabiti domač televizor in telefon, z dodatno vgraditivijo adapterja v televizor in uporabo tastarure. To omogoča uporabo računalniško podprtega informacijskega sistema vsakemu posamezniku, povsem fleksibilno glede na njegove želje, razpoložljiv čas in lokacijo. O tem in podobnih sistemih pa pri nas praktično ni ničesar znanega oziroma je poznavanje teh sistemov omejeno na ozek krog strokovnjakov. Potrebno bi torej bilo širšo javnost seznaniti s tovrstnim razvojem, pri razpravah o informacijskih sistemih pa od razpravljalcev in predlagateljev zahtevati več strokovnosti in poglobitev znanja.

Nakazano je bilo, da celo na področjih, kjer so vsaj po formalni plati doseženi določeni sporazumi in dogovori teh ne izvajamo, oziroma jih tolmačimo vsak po svoje, predvsem pa je mnogo premalo medrepubliške koordinacije.

Ločevati je počasi potreba začeti pojma baze podatkov in informacijskega sistema. Sicer je res, da enega brez drugega ne moremo vzpostaviti, drugi del resnice pa je, da operativno nimata praktično ničesar skupnega. Prav tako je povsem jasno, da v širšem kontekstu razprav o družbenem sistemu informiranja nedvomno vanj sodijo tudi informacijskih sistemi za znanstveno, tehnično in strokovno informiranje in prenos znanja in čas je, da nehamo razprave o DSI omejevati samo na informacijske sisteme statistike, SDK in morda še Narodne banke. Povsem jasno je tudi, da nam je projekt posodobitve informacijskih sistemov za znanstvene tehnične in strokovne informacije nujno potreben, in nikakor se ne sme zgoditi, da bi iz kakršnihkoli razlogov v neskončnost odlagali njegovo realizacijo.

Pri stikih z uporabniki se da mnogokrat ugotoviti, da za sistematičnejšo in obsežnejšo uporabo informacijskih sistemov za znanstvene in tehnične informacije ni prave motivacije, včasih pa je celo potreba po takih informacijah vprašljiva "saj shajamo tudi brez njih". treba bo torej temeljiteje oklestiti našo nagnjenost k improvizaciji in poslpoševanju, ki omogoča tako zgrešene zaključke. Poleg tega ne obstajajo norme za izvajanje določene aktivnosti (na primer investicijskih projektov, raziskovalnega dela ali licenčnih dogovorov in pogodb), ki bi eksplicitno zahtevale porabo informacijskih sistemov za nujno dviganje startne osnove teh aktivnosti, preprečevanje podvajanja, s tem pa zmanjševanje tehnoloških razlik med domovino in tujino. Ni se uveljavila potreba po povratnih informacijah od uporabnikov k informacišskim sistemom, kar bi zadnjim šele omogočilo prilagojevanje potrebam uporabnikom, uporabnikom pa dajalo občutek ali morda celo garancijo o smiselnosti in koristnosti informacijskega sistema (kakršnegakoli).

Opazno je kronično pomanjkanje znanja in razumevanja principov informacijskih sistemov, to pa povzroča kritične zaostanke, zlasti kadar se to neznanje pojavi na nivoju poslovodne, upravne alitehniške in tehnološke vodilne strukture kadrov. Zaradi tega je potrebno paralelno tudi za funkcionalno izobraževanje, predvsem vodilnih in tehniških in tehnoloških kadrov, na primer v zborničnem in izobraževalnem sistemu.

V primerjavi z ostalim svetom (predvsem razvitim, pa tudi že nekaterimi deželami v razvoju) mnogo premalo sistematično skrbimo za družbeno sofinanciranje te dejavnosti, pretirane zahteve po svobodni menjavi oziroma konkretnem plačevanju storitev s strani uporabnikov pa preveč obremenjuje tiste redke osveščene uporabnike in jih v bistvu odvračajo od uporabe informacijskih storitev. Poleg tega pa med uporabniki dodatno ustvarja slabo kri naravnost ignorantsko obnašanje določenega dela gospodarstva (pa tudi negospodarstva) za katerega izgleda, da ga informacijski sistemi preprosto sploh ne zanimajo. Resnici na ljubo pa je potrebno povedati, da redkim osveščenim uporabnikom tudi v lastnem okolju ni posilano z rožicami, saj vsakodnevni pereči, predvsem gospodarski problemi prevpijejo kakršnokoli dolgoročno načrtovanje in financiranje.

Izredno nizka informacijska kultura v naši deželi se kaže tudi skozi mnenja, ki jih je mngokrat slišati, da bi informacije (v splošnem) morale biti zastonj. Ob tem se navaja argument, da pregledovanje strokovnih knjig, revij in poročil v knjižnicah nič ne stane, pozablja pa se na dejstvo, da to stane ogromno časa, poleg tega pa je efekt takega ročnega dela slab, še zlasti ob upošte-

vanju trenutne suše pri uvozu literature in na splošno slabe založenosti knjižnic. Družba kot celota se še ne zaveda, da se danes v svetu informacijska storitev postavlja ob bok zagotovitvi energije, surovin in hrane, da pa se celo predvideva, da bo v bližnji bodočnosti ravno informacijska storitev postala glavni motor nadaljnjega razvoja in konkurenčnosti v mednarodnem pogledu.

Ob koncu okrogle mize so bili soglasno sprejeti naslednji zaključki:

1. Nikakor ni mogoče dovoliti nadaljnjega zaostajanja na področju znanstvenega in tehničnega informiranja. Potrebno se je čimprej lotiti prej predstavljenega projekta, ob tem pa selektivno - sredstvom, kadrovskim in drugim možnostim primerno ter čimbolj združeno stremeti k doseganju rezultatov na vnaprej zaokroženih celotah, ter jih čimprej sprovesti v vsakodnevno prakso.

2. Ni mogoče čakati na zamudnike, pač pa jim je potrebno ob primernih pogojih (ob tem jih je potrebno vzpodbujati) omogočiti vključevanje v delo pri razvoju in uporabi informacijskih sistemov.

3. Pri zagotavljanju informacijskih potreb uporabnikom na tem področju se je potrebno čimbolj svetovno orientirati in izrabiti možnosti, ki so nam ponujene, doma pa dograjevati tisto, česar v svetu ni mogoče dobiti oziroma je nacionalno pomembno.

4. Nespametno je delati dolgoročne vseobsegajoče programe, pač pa biti fleksibilen in odprt za nove alternative, potrebe in pojavljajoče se tehnološke možnosti.

5. Uvajati je potrebno določene standarde, predvsem za nivo storitev, formate zapisov, roke in podobno.

AVTORJI IN SODELAVCI

A. Milton Jenkins, M.B.A., Ph.D. is an Associate Professor of Management Information Systems at the Graduate School of Business, Indiana University where he has developed Graduate Programs in Management Information Systems. He has twelve years of industrial experience and is active as a consultant with various business and government organizations, and as a researcher in systems analysis and design, and MIS user systems interface. He has lectured on these topics in over thirty countries. He is a member of the American Institute for Decision Sciences, Society for Management Information Systems, Association for Computing Machinery, and Association for Systems Management,

Marko Kovačević (1953) je diplomiral na Fakulteti za elektrotehniko v Ljubljani (1976) na smeri Računalništvo in informatika. Že med diplómo se je zaposlil na Oddelku za elektroniko IJS, kjer je delal na nalogi Izdelava makroprocesorja za zbirni jezik mikroračunalnika. Kasneje je sodeloval pri razvoju za industrijo v okviru nalog Razvoj mikroračunalniškega sistema Iskra Data 1680 in Razvojni telefonski sistem TK 6800. V okviru teh nalog je delal na področjih operacijskih sistemov, materialne in programske računalniške opreme.

Leta 1981 se je zaposlil v DO Delta (kasneje Iskra Delta), kjer je razvil videoterminalni modul za mikroračunalnike ter sodeloval pri razvoju sistemske programske opreme za mikroračunalnike.

Marko Kovačević je sodeloval tudi pri vrsti raziskovalnih nalog s področja razvoja in uporabe mikroračunalniških sistemov. Vrsto svojih prispevkov je objavil v časopisu Informatica.

NAVODILO ZA PRIPRAVO ČLANKA

Avtorje prosimo, da pošljejo uredništvu naslov in kratek povzetek članka ter navedejo približen obseg članka (število strani A 4 formata). Uredništvo bo nato poslalo avtorjem ustrezno število formularjev z navodilom.

Članek tipkajte na priložene dvokolonske formularje. Če potrebujete dodatne formularje, lahko uporabite bel papir istih dimenzij. Pri tem pa se morate držati predpisanega formata, vendar pa ga ne vrišite na papir.

Bodite natančni pri tipkanju in temeljiti pri korigiranju. Vaš članek bo s foto postopkom pomanjšan in pripravljen za tisk brez kakršnihkoli dodatnik korektur.

Uporabljajte kvaliteten pisalni stroj. Če le tekst dopušča uporabljajte enojni presledek. Črni trak je obvezen.

Članek tipkajte v prostor obrobljen z modrimi črtami. Tipkajte do črt - ne preko njih. Odstavek ločite z dvojnim presledkom in brez zamikanja prve vrstice novega odstavka.

Prva stran članka:
a) v sredino zgornjega okvira na prvi strani napišite naslov članka z velikimi črkami;
b) v sredino pod naslov članka napišite imena avtorjev, ime podjetja, mesto, državo;
c) na označenem mestu čez oba stolpca napišite povzetek članka v jeziku, v katerem je napisan članek. Povzetek naj ne bo daljši od 10 vrst.
d) če članek ni v angleščini, ampak v katerem od jugoslovanskih jezikov izpustite 2 cm in napišite povzetek tudi v angleščini. Pred povzetkom napišite angleški naslov članka z velikimi črkami. Povzetek naj ne bo daljši od 10 vrst. Če je članek v tujem jeziku napišite povzetek tudi v enem od jugoslovanskih jezikov;
e) izpustite 2 cm in pričnite v levo kolono pisati članek.

Druga in naslednje strani članka:
Kot je označeno na formularju začnite tipkati tekst druge in naslednjih strani v zgornjem levem kotu,

Naslovi poglavij:
naslove ločuje od ostalega teksta dvojni presledek.

Če nekaterih znakov ne morete vpisati s strojem jih čitljivo vpišite s črnim črnilom ali svinčnikom. Ne uporabljajte modrega črnila, ker se z njim napisani znaki ne bodo preslikali.

Ilustracije morajo biti ostre, jasne in črno bele. Če jih vključite v tekst, se morajo skladati s predpisanim formatom. Lahko pa jih vstavite tudi na konec članka, vendar morajo v tem primeru ostati v mejah skupnega dvokolonskega formata. Vse ilustracije morate (nalepiti) vstaviti sami na ustrezno mesto.

Napake pri tipkanju se lahko popravljajo s korekrijsko folijo ali belim tušem. Napačne besede, stavke ali odstavke pa lahko ponovno natipkate na neprozoren papir in ga pazljivo nalepite na mesto napake.

V zgornjem desnem kotu izven modro označenega roba oštevilčite strani članka s svinčnikom, tako da jih je mogoče zbrisati.

Časopis INFORMATICA
Uredništvo, Parmova 41, 61000 Ljubljana

Naročam se na časopis INFORMATICA. Predplačilo bom izvršil po prejemu vaše položnice.

Cenik: letna naročnina za delovne organizacije 500,00 din, za posameznika 200,00/100,00/50,00 din

Časopis mi pošiljajte na naslov ☐ stanovanja ☐ delovne organizacije.

Priimek.....

Ime.....

Naslov stanovanja

Ulica.....

Poštna številka_____ Kraj.....

Naslov delovne organizacije

Delovna organizacija.....

.....

Ulica.....

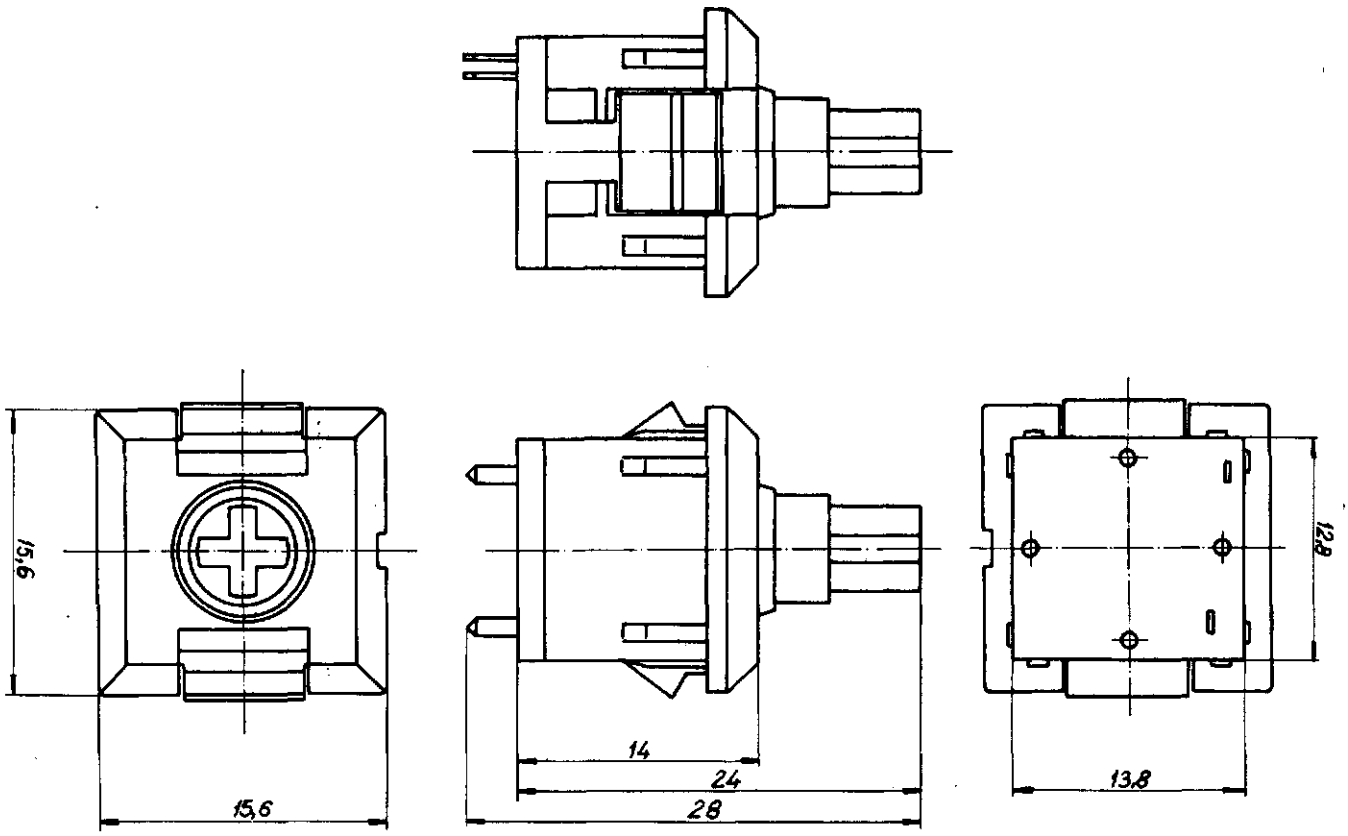
Poštna številka_____ Kraj.....

Datum..... Podpis:

.....

TIPKA SA KRIŽNIM KONTAKTOM (CROSS-POINT) — TX1

DIGITRON · BUJE



Tipka sa križnim kontaktom (cross-point) – TX1

Tipka TX1 ima kao kontaktni element ugrađen kontakt od difundiranog zlata. To joj omogućava kvalitetan i dugi vijek trajanja u najrazličitijim uvjetima rada.

Tipka je napravljena tako, da se postavlja u metalni raster, a kontakti se ostvaruju preko štampane ploče. Glava tipke sa svim pripadajućim znakovima omogućava veoma široku primjenu.

Tipka odgovara standardu: JUS N. 24. 030/31 (1969) i IEC Publ. 341-J/1A (1973).

Digitron Buje u suradnji sa institutom za elektroniko in vakuumsko tehniko – IEVT Ljubljana.

TEHNIČKE KARAKTERISTIKE

sila uključenja	0,8 ± 0,1	N
težina	3	gr.
kontaktni materijal	difundirano zlato	
materijal kućišta	pol. karborat	
najveći pomak klizača	3,2	mm
uključno-isključna histereza	1,5 ± 0,3	mm
isključenje	<0,3	mm
vijek trajanja	$>5 \cdot 10^6$	preklop.
radna temperatura	–10 do 60	°C
radna snaga	<5	VA
radni napon	<100	V=
struja prekida	<100	mA
probojni napon	>300	V=
kontaktni otpor	<300	mΩ
iztitravanje kontakta	3	m sec
vrsta kontakta	1^x radni ili 2^x radni	

DIGITRON
tvornica elektroničkih uređaja - 51460 buje - jugoslavija - tel.: (053) 71-202, 71-222, 71-242, tlx.: 25-128, 25-175 db yu
PREDSTAVNIŠTVA: BEOGRAD, Terazije 3/V, Tel.: (011) 320-039, Telex: 11-933 - SARAJEVO, Vase Pelagića br. 8, Tel.: (071) 24-329, Telex: 41-401 yu db - SKOPLJE, Gradski zid blok br. 3, Tel.: (091) 223-973 - TITOGRAD, Bulevar S. Kovačevića 143, Tel.: (081) 51-076 ZAGREB, Gajeva br. 59/I, (041) 440-844, 440-845, Telex: 21-119

Fotolit: "PLUTAL" TOZD GRAFIKA, Ilirska Bistrica – Tisek: "OFFSET TISKARA", Buje

COMPUTER CENTRE

Trst, Ul. F. Severo 89 – Tel. 574090

CENTER ZA VAŠO OSEBNO UPORABO

- ATOM – BBC z mikroprocesorjem 6502
- TEXAS TI 99/4 A 16-bitni, z najugodnejšo ceno
- EPSON HX-20 osebni procesni računalnik
- APPLE računalniki
- OLIVETTI M 20 T 16-bitni italijanski mikroračunalnik
- SIRIUS grafični računalnik
- ALTOS poslovni računalnik

Dodatna oprema:

tiskalniki	(EPSON, OKI, TALLY, CENTRONICS, HONEYWELL, OLYMPIA, ANTARES)
monitorji	9 in 12 colski, z zelenim fosforjem 14 colski, barvni
računalnika oprema	(FLOPPY DISKI, KASETE, magnetni trakovi, pohištvo za terminale in tiskalnike, pisalni trakovi)

Garantiramo tehnične usluge in servis za računalnike, kupljene pri nas. Smo avtorizirani za servis APPLE računalnikov.